MySQL 5.7 Reference Manual

MySQL 5.7 Reference Manual

Abstract

This is the MySQL™ Reference Manual. It documents MySQL 5.7 through 5.7.5.

MySQL Cluster is currently not supported in MySQL 5.7. For information about MySQL Cluster, please see MySQL Cluster NDB 7.3.

MySQL 5.7 features. This manual describes features that are not included in every edition of MySQL 5.7; such features may not be included in the edition of MySQL 5.7 licensed to you. If you have any questions about the features included in your edition of MySQL 5.7, refer to your MySQL 5.7 license agreement or contact your Oracle sales representative.

For release notes detailing the changes in each release, see the MySQL 5.7 Release Notes.

For legal information, see the Legal Notices.

Document generated on: 2014-04-11 (revision: 38404)

General	Administrators	MySQL Enterprise	Developers & Functionality	Connectors & APIs	HA/Scalability
Tutorial	Installation & Upgrades	MySQL Enterprise Edition	MySQL Workbench	Connectors and APIs	» HA/Scalability Guide
Server Administration	MySQL Yum Repository	MySQL Enterprise Monitor	Globalization	Connector/J	MySQL and DRBD
SQL Syntax	» MySQL Installer	MySQL Enterprise Backup	Optimization	Connector/ODBC	memcached
Storage Engines	» Security	MySQL Enterprise Security	Functions and Operators	Connector/Net	MySQL and Virtualization
Server Option / Variable Reference	» Startup / Shutdown	MySQL Enterprise Audit	Views and Stored Programs	Connector/Python	MySQL Proxy
» Release Notes	» Backup and Recovery Overview	MySQL Thread Pool	Partitioning	PHP	Replication
» MySQL Version Reference	» MySQL Utilities		Precision Math	C API	Semisynchronous Replication
FAQs	» Linux/Unix Platform Guide		Information Schema	Connector/C	
	» Windows Platform Guide		Performance Schema	Connector/C++	
	» Mac OS X Platform Guide		Spatial Extensions	» MySQL for Excel	
	» Solaris Platform Guide		Restrictions and Limitations		
	» Building from Source				

Table of Contents

Preface and Legal Notices

This is the Reference Manual for the MySQL Database System, version 5.7, through release 5.7.5. Differences between minor versions of MySQL 5.7 are noted in the present text with reference to release numbers (5.7.*x*). For license information, see the Legal Notices. This product may contain third-party code. For license information on third-party code, see Appendix A, *Licenses for Third-Party Components*.

This manual is not intended for use with older versions of the MySQL software due to the many functional and other differences between MySQL 5.7 and previous versions. If you are using an earlier release of the MySQL software, please refer to the appropriate manual. For example, *MySQL 5.6 Reference Manual* covers the 5.6 series of MySQL software releases.

Legal Notices

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this software or related documentation is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, the following notice is applicable:

U.S. GOVERNMENT RIGHTS Programs, software, databases, and related documentation and technical data delivered to U.S. Government customers are "commercial computer software" or "commercial technical data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, duplication, disclosure, modification, and adaptation shall be subject to the restrictions and license terms set forth in the applicable Government contract, and, to the extent applicable by the terms of the Government contract, the additional rights set forth in FAR 52.227-19, Commercial Computer Software License (December 2007). Oracle USA, Inc., 500 Oracle Parkway, Redwood City, CA 94065.

This software is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications which may create a risk of personal injury. If you use this software in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure the safe use of this software. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software in dangerous applications.

Oracle is a registered trademark of Oracle Corporation and/or its affiliates. MySQL is a trademark of Oracle Corporation and/or its affiliates, and shall not be used without Oracle's express written authorization. Other names may be trademarks of their respective owners.

This software and documentation may provide access to or information on content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services.

This documentation is in prerelease status and is intended for demonstration and preliminary use only. It may not be specific to the hardware on which you are using the software. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to this documentation and will not be responsible for any loss, costs, or damages incurred due to the use of this documentation.

The information contained in this document is for informational sharing purposes only and should be considered in your capacity as a customer advisory board member or pursuant to your beta trial agreement only. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described in this document remains at the sole discretion of Oracle.

This document in any form, software or printed matter, contains proprietary information that is the exclusive property of Oracle. Your access to and use of this material is subject to the terms and conditions of your Oracle Software License and Service Agreement, which has been executed and with which you agree to comply. This document and information contained herein may not be disclosed, copied, reproduced, or distributed to anyone outside Oracle without prior written consent of Oracle or as specifically provided below. This document is not part of your license agreement nor can it be incorporated into any contractual agreement with Oracle or its subsidiaries or affiliates.

This documentation is NOT distributed under a GPL license. Use of this documentation is subject to the following terms:

You may create a printed copy of this documentation solely for your own personal use. Conversion to other formats is allowed as long as the actual content is not altered or edited in any way. You shall not publish or distribute this documentation in any form or on any media, except if you distribute the documentation in a manner similar to how Oracle disseminates it (that is, electronically for download on a Web site with the software) or on a CD-ROM or similar medium, provided however that the documentation is disseminated together with the software on the same medium. Any other use, such as any dissemination of printed copies or use of this documentation, in whole or in part, in another publication, requires the prior written consent from an authorized representative of Oracle. Oracle and/or its affiliates reserve any and all rights to this documentation not expressly granted above.

For more information on the terms of this license, or for details on how the MySQL documentation is built and produced, please visit MySQL Contact & Questions.

For additional licensing information, including licenses for third-party libraries used by MySQL products, see Preface and Legal Notices.

For help with using MySQL, please visit either the MySQL Forums or MySQL Mailing Lists where you can discuss your issues with other MySQL users.

For additional documentation on MySQL products, including translations of the documentation into other languages, and downloadable versions in variety of formats, including HTML and PDF formats, see the MySQL Documentation Library.

Chapter 1 General Information

Table of Contents

The MySQL™ software delivers a very fast, multi-threaded, multi-user, and robust SQL (Structured Query Language) database server. MySQL Server is intended for mission-critical, heavy-load production systems as well as for embedding into mass-deployed software. Oracle is a registered trademark of Oracle Corporation and/or its affiliates. MySQL is a trademark of Oracle Corporation and/or its affiliates, and shall not be used by Customer without Oracle's express written authorization. Other names may be trademarks of their respective owners.

The MySQL software is Dual Licensed. Users can choose to use the MySQL software as an Open Source product under the terms of the GNU General Public License (http://www.fsf.org/licenses/) or can purchase a standard commercial license from Oracle. See http://www.mysql.com/company/legal/licensing/ for more information on our licensing policies.

The following list describes some sections of particular interest in this manual:

- For a discussion of MySQL Database Server capabilities, see Section 1.3.2, “The Main Features of MySQL”.
- For an overview of new MySQL features, see Section 1.4, “What Is New in MySQL 5.7”. For information about the changes in each version, see the Release Notes.
- For installation instructions, see Chapter 2, *Installing and Upgrading MySQL*. For information about upgrading MySQL, see Section 2.10.1, “Upgrading MySQL”.
- For a tutorial introduction to the MySQL Database Server, see Chapter 3, *Tutorial*.

- For information about configuring and administering MySQL Server, see Chapter 5, *MySQL Server Administration*.
- For information about security in MySQL, see Chapter 6, *Security*.
- For information about setting up replication servers, see Chapter 16, *Replication*.
- For information about MySQL Enterprise, the commercial MySQL release with advanced features and management tools, see Chapter 23, *MySQL Enterprise Edition*.
- For answers to a number of questions that are often asked concerning the MySQL Database Server and its capabilities, see Appendix B, *MySQL 5.7 Frequently Asked Questions*.
- For a history of new features and bug fixes, see the Release Notes.

⚠

Important

To report problems or bugs, please use the instructions at Section 1.7, “How to Report Bugs or Problems”. If you find a sensitive security bug in MySQL Server, please let us know immediately by sending an email message to `<secalert_us@oracle.com>`. Exception: Support customers should report all problems, including security bugs, to Oracle Support.

1.1 About This Manual

This is the Reference Manual for the MySQL Database System, version 5.7, through release 5.7.5. Differences between minor versions of MySQL 5.7 are noted in the present text with reference to release numbers (5.7.*x*). For license information, see the Legal Notices. This product may contain third-party code. For license information on third-party code, see Appendix A, *Licenses for Third-Party Components*.

This manual is not intended for use with older versions of the MySQL software due to the many functional and other differences between MySQL 5.7 and previous versions. If you are using an earlier release of the MySQL software, please refer to the appropriate manual. For example, *MySQL 5.6 Reference Manual* covers the 5.6 series of MySQL software releases.

Because this manual serves as a reference, it does not provide general instruction on SQL or relational database concepts. It also does not teach you how to use your operating system or command-line interpreter.

The MySQL Database Software is under constant development, and the Reference Manual is updated frequently as well. The most recent version of the manual is available online in searchable form at http://dev.mysql.com/doc/. Other formats also are available there, including HTML, PDF, and EPUB versions.

The Reference Manual source files are written in DocBook XML format. The HTML version and other formats are produced automatically, primarily using the DocBook XSL stylesheets. For information about DocBook, see http://docbook.org/

If you have questions about using MySQL, you can ask them using our mailing lists or forums. See Section 1.6.1, “MySQL Mailing Lists”, and Section 1.6.2, “MySQL Community Support at the MySQL Forums”. If you have suggestions concerning additions or corrections to the manual itself, please send them to the http://www.mysql.com/company/contact/.

This manual was originally written by David Axmark and Michael “Monty” Widenius. It is maintained by the MySQL Documentation Team, consisting of Paul DuBois, Stefan Hinz, Philip Olson, Daniel Price, Daniel So, Edward Gilmore, and Jon Stephens.

1.2 Typographical and Syntax Conventions

This manual uses certain typographical conventions:

- `Text in this style` is used for SQL statements; database, table, and column names; program listings and source code; and environment variables. Example: “To reload the grant tables, use the `FLUSH PRIVILEGES` statement.”

- **`Text in this style`** indicates input that you type in examples.

- `Text in this style` indicates the names of executable programs and scripts, examples being `mysql` (the MySQL command-line client program) and `mysqld` (the MySQL server executable).

- *`Text in this style`* is used for variable input for which you should substitute a value of your own choosing.

- *Text in this style* is used for emphasis.

- **Text in this style** is used in table headings and to convey especially strong emphasis.

- `Text in this style` is used to indicate a program option that affects how the program is executed, or that supplies information that is needed for the program to function in a certain way. *Example*: “The `--host` option (short form `-h`) tells the `mysql` client program the hostname or IP address of the MySQL server that it should connect to”.

- File names and directory names are written like this: “The global `my.cnf` file is located in the `/etc` directory.”

- Character sequences are written like this: “To specify a wildcard, use the ‘`%`’ character.”

When commands are shown that are meant to be executed from within a particular program, the prompt shown preceding the command indicates which command to use. For example, `shell>` indicates a command that you execute from your login shell, `root-shell>` is similar but should be executed as `root`, and `mysql>` indicates a statement that you execute from the `mysql` client program:

```
shell> type a shell command here
root-shell> type a shell command as root here
mysql> type a mysql statement here
```

In some areas different systems may be distinguished from each other to show that commands should be executed in two different environments. For example, while working with replication the commands might be prefixed with `master` and `slave`:

```
master> type a mysql command on the replication master here
slave> type a mysql command on the replication slave here
```

The “shell” is your command interpreter. On Unix, this is typically a program such as `sh`, `csh`, or `bash`. On Windows, the equivalent program is `command.com` or `cmd.exe`, typically run in a console window.

When you enter a command or statement shown in an example, do not type the prompt shown in the example.

Database, table, and column names must often be substituted into statements. To indicate that such substitution is necessary, this manual uses *`db_name`*, *`tbl_name`*, and *`col_name`*. For example, you might see a statement like this:

```
mysql> SELECT col_name FROM db_name.tbl_name;
```

This means that if you were to enter a similar statement, you would supply your own database, table, and column names, perhaps like this:

```
mysql> SELECT author_name FROM biblio_db.author_list;
```

SQL keywords are not case sensitive and may be written in any lettercase. This manual uses uppercase.

In syntax descriptions, square brackets (“[” and “]”) indicate optional words or clauses. For example, in the following statement, IF EXISTS is optional:

```
DROP TABLE [IF EXISTS] tbl_name
```

When a syntax element consists of a number of alternatives, the alternatives are separated by vertical bars (“|”). When one member from a set of choices *may* be chosen, the alternatives are listed within square brackets (“[” and “]”):

```
TRIM([[BOTH | LEADING | TRAILING] [remstr] FROM] str)
```

When one member from a set of choices *must* be chosen, the alternatives are listed within braces (“{” and “}”):

```
{DESCRIBE | DESC} tbl_name [col_name | wild]
```

An ellipsis (...) indicates the omission of a section of a statement, typically to provide a shorter version of more complex syntax. For example, SELECT ... INTO OUTFILE is shorthand for the form of SELECT statement that has an INTO OUTFILE clause following other parts of the statement.

An ellipsis can also indicate that the preceding syntax element of a statement may be repeated. In the following example, multiple *reset_option* values may be given, with each of those after the first preceded by commas:

```
RESET reset_option [,reset_option] ...
```

Commands for setting shell variables are shown using Bourne shell syntax. For example, the sequence to set the CC environment variable and run the configure command looks like this in Bourne shell syntax:

```
shell> CC=gcc ./configure
```

If you are using csh or tcsh, you must issue commands somewhat differently:

```
shell> setenv CC gcc
shell> ./configure
```

1.3 Overview of the MySQL Database Management System

1.3.1 What is MySQL?

MySQL, the most popular Open Source SQL database management system, is developed, distributed, and supported by Oracle Corporation.

The MySQL Web site (http://www.mysql.com/) provides the latest information about MySQL software.

- **MySQL is a database management system.**

  A database is a structured collection of data. It may be anything from a simple shopping list to a picture gallery or the vast amounts of information in a corporate network. To add, access, and process data stored in a computer database, you need a database management system such as MySQL Server. Since computers are very good at handling large amounts of data, database management systems play a central role in computing, as standalone utilities, or as parts of other applications.

- **MySQL databases are relational.**

  A relational database stores data in separate tables rather than putting all the data in one big storeroom. The database structures are organized into physical files optimized for speed. The logical model, with objects such as databases, tables, views, rows, and columns, offers a flexible programming environment. You set up rules governing the relationships between different data fields, such as one-to-one, one-to-many, unique, required or optional, and “pointers” between different tables. The database enforces these rules, so that with a well-designed database, your application never sees inconsistent, duplicate, orphan, out-of-date, or missing data.

  The SQL part of “MySQL” stands for “Structured Query Language”. SQL is the most common standardized language used to access databases. Depending on your programming environment, you might enter SQL directly (for example, to generate reports), embed SQL statements into code written in another language, or use a language-specific API that hides the SQL syntax.

  SQL is defined by the ANSI/ISO SQL Standard. The SQL standard has been evolving since 1986 and several versions exist. In this manual, “SQL-92” refers to the standard released in 1992, “SQL:1999” refers to the standard released in 1999, and “SQL:2003” refers to the current version of the standard. We use the phrase “the SQL standard” to mean the current version of the SQL Standard at any time.

- **MySQL software is Open Source.**

  Open Source means that it is possible for anyone to use and modify the software. Anybody can download the MySQL software from the Internet and use it without paying anything. If you wish, you may study the source code and change it to suit your needs. The MySQL software uses the GPL (GNU General Public License), http://www.fsf.org/licenses/, to define what you may and may not do with the software in different situations. If you feel uncomfortable with the GPL or need to embed MySQL code into a commercial application, you can buy a commercially licensed version from us. See the MySQL Licensing Overview for more information (http://www.mysql.com/company/legal/licensing/).

- **The MySQL Database Server is very fast, reliable, scalable, and easy to use.**

  If that is what you are looking for, you should give it a try. MySQL Server can run comfortably on a desktop or laptop, alongside your other applications, web servers, and so on, requiring little or no attention. If you dedicate an entire machine to MySQL, you can adjust the settings to take advantage of all the memory, CPU power, and I/O capacity available. MySQL can also scale up to clusters of machines, networked together.

  You can find a performance comparison of MySQL Server with other database managers on our benchmark page. See Section 8.12.2, “The MySQL Benchmark Suite”.

  MySQL Server was originally developed to handle large databases much faster than existing solutions and has been successfully used in highly demanding production environments for several years. Although under constant development, MySQL Server today offers a rich and useful set of functions. Its connectivity, speed, and security make MySQL Server highly suited for accessing databases on the Internet.

- **MySQL Server works in client/server or embedded systems.**

The MySQL Database Software is a client/server system that consists of a multi-threaded SQL server that supports different backends, several different client programs and libraries, administrative tools, and a wide range of application programming interfaces (APIs).

We also provide MySQL Server as an embedded multi-threaded library that you can link into your application to get a smaller, faster, easier-to-manage standalone product.

- **A large amount of contributed MySQL software is available.**

  MySQL Server has a practical set of features developed in close cooperation with our users. It is very likely that your favorite application or language supports the MySQL Database Server.

The official way to pronounce “MySQL” is “My Ess Que Ell” (not “my sequel”), but we do not mind if you pronounce it as “my sequel” or in some other localized way.

1.3.2 The Main Features of MySQL

This section describes some of the important characteristics of the MySQL Database Software. See also Section 1.5, “MySQL Development History”. In most respects, the roadmap applies to all versions of MySQL. For information about features as they are introduced into MySQL on a series-specific basis, see the “In a Nutshell” section of the appropriate Manual:

- MySQL 5.6: MySQL 5.6 in a Nutshell
- MySQL 5.5: MySQL 5.5 in a Nutshell
- MySQL 5.1: MySQL 5.1 in a Nutshell
- MySQL 5.0: MySQL 5.0 in a Nutshell

Internals and Portability:

- Written in C and C++.
- Tested with a broad range of different compilers.
- Works on many different platforms. See http://www.mysql.com/support/supportedplatforms/database.html.
- For portability, uses `CMake` in MySQL 5.5 and up. Previous series use GNU Automake, Autoconf, and Libtool.
- Tested with Purify (a commercial memory leakage detector) as well as with Valgrind, a GPL tool (http://developer.kde.org/~sewardj/).
- Uses multi-layered server design with independent modules.
- Designed to be fully multi-threaded using kernel threads, to easily use multiple CPUs if they are available.
- Provides transactional and nontransactional storage engines.
- Uses very fast B-tree disk tables (`MyISAM`) with index compression.
- Designed to make it relatively easy to add other storage engines. This is useful if you want to provide an SQL interface for an in-house database.

- Uses a very fast thread-based memory allocation system.
- Executes very fast joins using an optimized nested-loop join.
- Implements in-memory hash tables, which are used as temporary tables.
- Implements SQL functions using a highly optimized class library that should be as fast as possible. Usually there is no memory allocation at all after query initialization.
- Provides the server as a separate program for use in a client/server networked environment, and as a library that can be embedded (linked) into standalone applications. Such applications can be used in isolation or in environments where no network is available.

Data Types:

- Many data types: signed/unsigned integers 1, 2, 3, 4, and 8 bytes long, `FLOAT`, `DOUBLE`, `CHAR`, `VARCHAR`, `BINARY`, `VARBINARY`, `TEXT`, `BLOB`, `DATE`, `TIME`, `DATETIME`, `TIMESTAMP`, `YEAR`, `SET`, `ENUM`, and OpenGIS spatial types. See Chapter 11, *Data Types*.
- Fixed-length and variable-length string types.

Statements and Functions:

- Full operator and function support in the `SELECT` list and `WHERE` clause of queries. For example:

```
mysql> SELECT CONCAT(first_name, ' ', last_name)
    -> FROM citizen
    -> WHERE income/dependents > 10000 AND age > 30;
```

- Full support for SQL `GROUP BY` and `ORDER BY` clauses. Support for group functions (`COUNT()`, `AVG()`, `STD()`, `SUM()`, `MAX()`, `MIN()`, and `GROUP_CONCAT()`).
- Support for `LEFT OUTER JOIN` and `RIGHT OUTER JOIN` with both standard SQL and ODBC syntax.
- Support for aliases on tables and columns as required by standard SQL.
- Support for `DELETE`, `INSERT`, `REPLACE`, and `UPDATE` to return the number of rows that were changed (affected), or to return the number of rows matched instead by setting a flag when connecting to the server.
- Support for MySQL-specific `SHOW` statements that retrieve information about databases, storage engines, tables, and indexes. MySQL 5.0 adds support for the `INFORMATION_SCHEMA` database, implemented according to standard SQL.
- An `EXPLAIN` statement to show how the optimizer resolves a query.
- Independence of function names from table or column names. For example, `ABS` is a valid column name. The only restriction is that for a function call, no spaces are permitted between the function name and the "`(`" that follows it. See Section 9.3, "Reserved Words".
- You can refer to tables from different databases in the same statement.

Security:

- A privilege and password system that is very flexible and secure, and that enables host-based verification.
- Password security by encryption of all password traffic when you connect to a server.

Scalability and Limits:

- Support for large databases. We use MySQL Server with databases that contain 50 million records. We also know of users who use MySQL Server with 200,000 tables and about 5,000,000,000 rows.
- Support for up to 64 indexes per table (32 before MySQL 4.1.2). Each index may consist of 1 to 16 columns or parts of columns. The maximum index width is 767 bytes for `InnoDB` tables, or 1000 for `MyISAM`; before MySQL 4.1.2, the limit is 500 bytes. An index may use a prefix of a column for `CHAR`, `VARCHAR`, `BLOB`, or `TEXT` column types.

Connectivity:

- Clients can connect to MySQL Server using several protocols:
  - Clients can connect using TCP/IP sockets on any platform.
  - On Windows systems in the NT family (NT, 2000, XP, 2003, or Vista), clients can connect using named pipes if the server is started with the `--enable-named-pipe` option. In MySQL 4.1 and higher, Windows servers also support shared-memory connections if started with the `--shared-memory` option. Clients can connect through shared memory by using the `--protocol=memory` option.
  - On Unix systems, clients can connect using Unix domain socket files.
- MySQL client programs can be written in many languages. A client library written in C is available for clients written in C or C++, or for any language that provides C bindings.
- APIs for C, C++, Eiffel, Java, Perl, PHP, Python, Ruby, and Tcl are available, enabling MySQL clients to be written in many languages. See Chapter 21, *Connectors and APIs*.
- The Connector/ODBC (MyODBC) interface provides MySQL support for client programs that use ODBC (Open Database Connectivity) connections. For example, you can use MS Access to connect to your MySQL server. Clients can be run on Windows or Unix. Connector/ODBC source is available. All ODBC 2.5 functions are supported, as are many others. See MySQL Connector/ODBC Developer Guide.
- The Connector/J interface provides MySQL support for Java client programs that use JDBC connections. Clients can be run on Windows or Unix. Connector/J source is available. See MySQL Connector/J Developer Guide.
- MySQL Connector/Net enables developers to easily create .NET applications that require secure, high-performance data connectivity with MySQL. It implements the required ADO.NET interfaces and integrates into ADO.NET aware tools. Developers can build applications using their choice of .NET languages. MySQL Connector/Net is a fully managed ADO.NET driver written in 100% pure C#. See MySQL Connector/Net Developer Guide.

Localization:

- The server can provide error messages to clients in many languages. See Section 10.2, “Setting the Error Message Language”.
- Full support for several different character sets, including `latin1` (cp1252), `german`, `big5`, `ujis`, and more. For example, the Scandinavian characters “`å`”, “`ä`” and “`ö`” are permitted in table and column names. Unicode support is available as of MySQL 4.1.
- All data is saved in the chosen character set.
- Sorting and comparisons are done according to the chosen character set and collation (using `latin1` and Swedish collation by default). It is possible to change this when the MySQL server is started. To see

an example of very advanced sorting, look at the Czech sorting code. MySQL Server supports many different character sets that can be specified at compile time and runtime.

- As of MySQL 4.1, the server time zone can be changed dynamically, and individual clients can specify their own time zone. Section 10.6, “MySQL Server Time Zone Support”.

Clients and Tools:

- MySQL includes several client and utility programs. These include both command-line programs such as `mysqldump` and `mysqladmin`, and graphical programs such as MySQL Workbench.
- MySQL Server has built-in support for SQL statements to check, optimize, and repair tables. These statements are available from the command line through the `mysqlcheck` client. MySQL also includes `myisamchk`, a very fast command-line utility for performing these operations on `MyISAM` tables. See Chapter 4, *MySQL Programs*.
- MySQL programs can be invoked with the `--help` or `-?` option to obtain online assistance.

1.3.3 History of MySQL

We started out with the intention of using the `mSQL` database system to connect to our tables using our own fast low-level (ISAM) routines. However, after some testing, we came to the conclusion that `mSQL` was not fast enough or flexible enough for our needs. This resulted in a new SQL interface to our database but with almost the same API interface as `mSQL`. This API was designed to enable third-party code that was written for use with `mSQL` to be ported easily for use with MySQL.

MySQL is named after co-founder Monty Widenius's daughter, My.

The name of the MySQL Dolphin (our logo) is “Sakila,” which was chosen from a huge list of names suggested by users in our “Name the Dolphin” contest. The winning name was submitted by Ambrose Twebaze, an Open Source software developer from Swaziland, Africa. According to Ambrose, the feminine name Sakila has its roots in SiSwati, the local language of Swaziland. Sakila is also the name of a town in Arusha, Tanzania, near Ambrose's country of origin, Uganda.

1.4 What Is New in MySQL 5.7

This section summarizes what has been added to, deprecated in, and removed from MySQL 5.7.

Added Features

The following features have been added to MySQL 5.7:

- **Security improvements.** The server now requires account rows in the `mysql.user` table to have a nonempty `plugin` column value and disables accounts with an empty value. For server upgrade instructions, see Section 2.10.1.2, “Upgrading from MySQL 5.6 to 5.7”. DBAs are advised to also convert accounts that use the deprecated `mysql_old_password` authentication plugin to use `mysql_native_password` instead. For account upgrade instructions, see Section 6.3.9.3, “Migrating Away from Pre-4.1 Password Hashing and the `mysql_old_password` Plugin”.

  MySQL now enables database administrators to establish a policy for automatic password expiration: Any user who connects to the server using an account for which the password is past its permitted lifetime must change the password. For more information, see Section 6.3.6, “Password Expiration Policy”.

  MySQL deployments installed using RPM packages now are secure by default. The following changes have been implemented as the default deployment characteristics:

  - The installation process creates only a single `root` account, `'root'@'localhost'`, automatically generates a random password for this account, and marks the password expired. The MySQL administrator must connect as `root` using the random password and use `SET PASSWORD` to select a new password. (The random password is found in the `$HOME/.mysql_secret` file.)

  - Installation creates no anonymous-user accounts.

  - Installation creates no `test` database.

- **Online `ALTER TABLE`.** `ALTER TABLE` now supports a `RENAME INDEX` clause that renames an index. The change is made in place without a table-copy operation. It works for all storage engines. See Section 13.1.6, “`ALTER TABLE` Syntax”.

- **`InnoDB` enhancements.** These `InnoDB` enhancements were added:

  - `VARCHAR` size may be increased using an in-place `ALTER TABLE`, as in this example:

    ```
    ALTER TABLE t1 ALGORITHM=INPLACE, CHANGE COLUMN c1 c1 VARCHAR(255);
    ```

    This is true as long as the number of length bytes required by a `VARCHAR` column remains the same. For `VARCHAR` values of 0 to 255, one length byte is required to encode the value. For `VARCHAR` values of 256 bytes or more, two length bytes are required. As a result, in-place `ALTER TABLE` only supports increasing `VARCHAR` size from 0 to 255 bytes or increasing `VARCHAR` size from a value equal to or greater than 256 bytes.

    In-place `ALTER TABLE` does not support increasing `VARCHAR` size from less than 256 bytes to a value equal to or greater than 256 bytes. In this case, the number of required length bytes would change from 1 to 2, which is only supported by a table copy (`ALGORITHM=COPY`). For example, attempting to change `VARCHAR` column size from 255 to 256 using in-place `ALTER TABLE` would return an error:

    ```
    ALTER TABLE t1 ALGORITHM=INPLACE, CHANGE COLUMN c1 c1 VARCHAR(256);
    ERROR 0A000: ALGORITHM=INPLACE is not supported. Reason: Cannot change
    column type INPLACE. Try ALGORITHM=COPY.
    ```

    Decreasing `VARCHAR` size using in-place `ALTER TABLE` is not supported. Decreasing `VARCHAR` size requires a table copy (`ALGORITHM=COPY`).

  - DDL performance for `InnoDB` temporary tables is improved through optimization of `CREATE TABLE`, `DROP TABLE`, `TRUNCATE TABLE`, and `ALTER TABLE` statements.

  - InnoDB temporary table metadata is no longer stored to InnoDB system tables. Instead, a new table, `INNODB_TEMP_TABLE_INFO`, provides users with a snapshot of active temporary tables. The table contains metadata and reports on all user and system-created temporary tables that are active within a given InnoDB instance. The table is created when the first `SELECT` statement is run against it.

  - `InnoDB` now supports MySQL-supported spatial data types. Prior to this release, InnoDB would store spatial data as binary `BLOB` data. `BLOB` remains the underlying data type but spatial data types are now mapped to a new InnoDB internal data type, `DATA_GEOMETRY`.

  - There is now a separate tablespace for all non-compressed InnoDB temporary tables. The new tablespace is always recreated on server startup and is located in `DATADIR` by default. A newly added configuration file option, `innodb_temp_data_file_path`, allows for a user-defined temporary data file path.

  - In MySQL 5.7.2, innochecksum functionality is enhanced with several new options and extended capabilities. See Section 4.6.1, “innochecksum — Offline InnoDB File Checksum Utility”.

  - A new type of non-redo undo log for both normal and compressed temporary tables and related objects now resides in the temporary tablespace. For more information, see Section 14.2.2.13, “InnoDB Temporary Table Undo Logs”.

  - In MySQL 5.7.2, InnoDB buffer pool dump and load operations are enhanced. A new system variable, innodb_buffer_pool_dump_pct, allows you to specify the percentage of most recently used pages in each buffer pool to read out and dump. When there is other I/O activity being performed by InnoDB background tasks, InnoDB attempts to limit the number of buffer pool load operations per second using the innodb_io_capacity setting.

  - In MySQL 5.7.3, support is added to InnoDB for full-text parser plugins. For information about full-text parser plugins, see Section 22.2.3.2, “Full-Text Parser Plugins” and Section 22.2.4.4, “Writing Full-Text Parser Plugins”.

  - As of MySQL 5.7.4, InnoDB supports multiple page_cleaner threads for flushing dirty pages from buffer pool instances. A new system variable, innodb_page_cleaners, is used to specify the number of page_cleaner threads. The default value of 1 maintains the pre-MySQL 5.7.4 configuration in which there is a single page_cleaner thread. This enhancement builds on work completed in MySQL 5.6.2, which introduced a single page_cleaner thread to offload buffer pool flushing work from the InnoDB master thread.

  - As of MySQL 5.7.4, MySQL supports rebuilding regular and partitioned InnoDB tables using online DDL (ALGORITHM=INPLACE) for the following operations:

    - OPTIMIZE TABLE

    - ALTER TABLE ... FORCE

    - ALTER TABLE ... ENGINE=INNODB (when run on an InnoDB table)

    Online DDL support reduces table rebuild time and permits concurrent DML, which helps reduce user application downtime. For additional information, see Section 14.2.11.1, “Overview of Online DDL”.

  - The Fusion-io Non-Volatile Memory (NVM) file system on Linux provides atomic write capability, which makes the InnoDB doublewrite buffer redundant. In MySQL 5.7.4, the InnoDB doublewrite buffer is automatically disabled for system tablespace files (ibdata files) located on Fusion-io devices that support atomic writes.

  - As of MySQL 5.7.4, InnoDB supports the Transportable Tablespace feature for partitioned InnoDB tables and individual InnoDB table partitions. This enhancement eases backup procedures for partitioned tables and enables copying of partitioned tables and individual table partitions between MySQL instances. For additional information, see Section 14.2.5.5, “Copying Tablespaces to Another Server (Transportable Tablespaces)”.

- **MySQL Enterprise.** The format of the file generated by the audit log plugin was changed for better compatibility with Oracle Audit Vault. See Section 6.3.13, “MySQL Enterprise Audit Log Plugin”, and Section 6.3.13.3, “The Audit Log File”.

- **Condition handling.** MySQL now supports stacked diagnostics areas. When the diagnostics area stack is pushed, the first (current) diagnostics area becomes the second (stacked) diagnostics area and a new current diagnostics area is created as a copy of it. Within a condition handler, executed statements modify the new current diagnostics area, but GET STACKED DIAGNOSTICS can be used to inspect the stacked diagnostics area to obtain information about the condition that caused the handler to activate, independent of current conditions within the handler itself. (Previously, there was a single diagnostics area. To inspect handler-activating conditions within a handler, it was necessary to check this diagnostics area before executing any statements that could change it.) See Section 13.6.7.3, "GET DIAGNOSTICS Syntax", and Section 13.6.7.7, "The MySQL Diagnostics Area".

- **Optimizer.** EXPLAIN can now be used to obtain the execution plan for an explainable statement executing in a named connection:

  ```
  EXPLAIN [options] FOR CONNECTION connection_id;
  ```

  For more information, see Section 8.8.3, "Obtaining Execution Plan Information for a Named Connection".

- **Triggers.** Previously, a table could have at most one trigger for each combination of trigger event (INSERT, UPDATE, DELETE) and action time (BEFORE, AFTER). This limitation has been lifted and multiple triggers are permitted. For more information, see Section 18.3, "Using Triggers".

- **Logging.** The mysql client now has a --syslog option that causes interactive statements to be sent to the system syslog facility. Logging is suppressed for statements that match the default "ignore" pattern list ("*IDENTIFIED*:*PASSWORD*"), as well as statements that match any patterns specified using the --histignore option. See Section 4.5.1.3, "mysql Logging".

- **Test suite.** The MySQL test suite now uses InnoDB as the default storage engine.

- **mysql client.** Previously, **Control+C** in mysql interrupted the current statement if there was one, or exited mysql if not. Now **Control+C** interrupts the current statement if there was one, or cancels any partial input line otherwise, but does not exit.

- **Database name rewriting with mysqlbinlog.** Renaming of databases by mysqlbinlog when reading from binary logs written using the row-based format is now supported using the --rewrite-db option added in MySQL 5.7.1.

  This option uses the format --rewrite-db='*dboldname->dbnewname*'. You can implement multiple rewrite rules, by specifying the option multiple times.

- **HANDLER with partitioned tables.** The HANDLER statement may now be used with user-partitioned tables. Such tables may use any of the available partitioning types (see Section 17.2, "Partitioning Types").

- **Index condition pushdown support for partitioned tables.** In MySQL 5.7.3 and later, queries on partitioned tables using the InnoDB or MyISAM storage engine may employ the index condition pushdown optimization that was introduced in MySQL 5.6. See Section 8.2.1.6, "Index Condition Pushdown Optimization", for more information.

- **Master dump thread improvements.** The master dump thread was refactored to reduce lock contention and improve master throughput. Previous to MySQL 5.7.2, the dump thread took a lock on the binary log whenever reading an event; in MySQL 5.7.2 and later, this lock is held only while reading the position at the end of the last successfully written event. This means both that multiple dump threads are now able to read concurrently from the binary log file, and that dump threads are now able to read while clients are writing to the binary log.

- **Globalization improvements.** MySQL 5.7.4 includes a gb18030 character set that supports the China National Standard GB18030 character set. For more information about MySQL character set support, see Section 10.1, "Character Set Support".

- **Changing the replication master without STOP SLAVE.** In MySQL 5.7.4 and later, the strict requirement to execute STOP SLAVE prior to issuing any CHANGE MASTER TO statement is removed. Instead of depending on whether the slave is stopped, the behavior of CHANGE MASTER TO now depends on the states of the slave SQL thread and slave I/O threads; which of these threads is stopped or running now determines the options that can or cannot be used with a CHANGE MASTER TO statement at a given point in time. The rules for making this determination are listed here:

  - If the SQL thread is stopped, you can execute CHANGE MASTER TO using any combination of RELAY_LOG_FILE, RELAY_LOG_POS, and MASTER_DELAY options, even if the slave I/O thread is running. No other options may be used with this statement when the I/O thread is running.

  - If the I/O thread is stopped, you can execute CHANGE MASTER TO using any of the options for this statement (in any allowed combination) *except* RELAY_LOG_FILE, RELAY_LOG_POS, or MASTER_DELAY, even when the SQL thread is running. These three options may not be used when the I/O thread is running.

  - Both the SQL thread and the I/O thread must be stopped before issuing CHANGE MASTER TO ... MASTER_AUTO_POSITION = 1.

  You can check the current state of the slave SQL and I/O threads using SHOW SLAVE STATUS.

  If you are using statement-based replication and temporary tables, it is possible for a CHANGE MASTER TO statement following a STOP SLAVE statement to leave behind temporary tables on the slave. As part of this set of improvements, a warning is now issued whenever CHANGE MASTER TO is issued following STOP SLAVE when statement-based replication is in use and Slave_open_temp_tables remains greater than 0.

  For more information, see Section 13.4.2.1, “CHANGE MASTER TO Syntax”, and Section 16.3.6, “Switching Masters During Failover”.

Deprecated Features

The following features are deprecated in MySQL 5.7 and may be or will be removed in a future series. Where alternatives are shown, applications should be updated to use them.

- The ERROR_FOR_DIVISION_BY_ZERO, NO_ZERO_DATE, and NO_ZERO_IN_DATE SQL modes that were deprecated in MySQL 5.6 remain deprecated in 5.7 but do nothing. Instead, their previous effects are included in the effects of strict SQL mode (STRICT_ALL_TABLES or STRICT_TRANS_TABLES). In other words, strict mode now means the same thing as the previous meaning of strict mode plus the ERROR_FOR_DIVISION_BY_ZERO, NO_ZERO_DATE, and NO_ZERO_IN_DATE modes. This change reduces the number of SQL modes with an effect dependent on strict mode and makes them part of strict mode itself.

  To prepare for these SQL mode changes, it is advisable *before* upgrading to read SQL Mode Changes in MySQL 5.7. That discussion provides guidelines to assess whether your applications will be affected by these changes.

  The deprecated ERROR_FOR_DIVISION_BY_ZERO, NO_ZERO_DATE, and NO_ZERO_IN_DATE SQL modes are still recognized so that statements that name them do not produce an error, but will be removed in a future version of MySQL. To make advance preparation for versions of MySQL in which these modes do not exist, applications should be modified to not refer to those mode names.

- Relying on implicit GROUP BY sorting in MySQL 5.7 is deprecated. To achieve a specific sort order of grouped results, it is preferable to use an explicit ORDER BY clause. GROUP BY sorting is a MySQL extension that may change in a future release; for example, to make it possible for the optimizer to order groupings in whatever manner it deems most efficient and to avoid the sorting overhead.

- The EXTENDED and PARTITIONS keywords for the EXPLAIN statement. These keywords are still recognized but are now unnecessary because their effect is always enabled.
- The log_warnings system variable and --log-warnings server option. Use the log_error_verbosity system variable instead.
- The metadata_locks_cache_size and metadata_locks_hash_instances system variables. These do nothing as of MySQL 5.7.4.
- The timed_mutexes system variable. It does nothing and has no effect.
- The ENCODE() and DECODE() functions. Use AES_ENCRYPT() and AES_DECRYPT() instead.
- The INFORMATION_SCHEMA.PROFILING table. Use the Performance Schema instead; see Chapter 20, *MySQL Performance Schema*.

Removed Features

The following constructs are obsolete and have been removed in MySQL 5.7. Where alternatives are shown, applications should be updated to use them.

- The innodb_mirrored_log_groups system variable. The only supported value was 1, so it had no purpose.
- The storage_engine system variable. Use default_storage_engine instead.
- The thread_concurrency system variable.
- The IGNORE clause for ALTER TABLE.
- INSERT DELAYED is no longer supported. The server recognizes but ignores the DELAYED keyword, handles the insert as a nondelayed insert, and generates an ER_WARN_LEGACY_SYNTAX_CONVERTED warning. (“INSERT DELAYED is no longer supported. The statement was converted to INSERT.”) Similarly, REPLACE DELAYED is handled as a nondelayed replace. The DELAYED keyword will be removed in a future release.

  In addition, several DELAYED-related options or features were removed:

  - The --delayed-insert option for mysqldump.
  - The COUNT_WRITE_DELAYED, SUM_TIMER_WRITE_DELAYED, MIN_TIMER_WRITE_DELAYED, AVG_TIMER_WRITE_DELAYED, and MAX_TIMER_WRITE_DELAYED columns of the Performance Schema table_lock_waits_summary_by_table table.
  - mysqlbinlog no longer writes comments mentioning INSERT DELAYED.
- Database symlinking on Windows using for .sym files has been removed because it is redundant with native symlink support available using mklink. Any .sym file symbolic links will be ignored and should be replaced with symlinks created using mklink. See Using Symbolic Links for Databases on Windows.
- The unused --basedir and --datadir options for mysql_upgrade were removed.
- Previously, program options could be specified in full or as any unambiguous prefix. For example, the --compress option could be given to mysqldump as --compr, but not as --comp because the latter is ambiguous. Option prefixes are no longer supported; only full options are accepted. This is because prefixes can cause problems when new options are implemented for programs and a prefix that is currently unambiguous might become ambiguous in the future.

- SHOW ENGINE INNODB MUTEX output is removed in MySQL 5.7.2. Comparable information can be generated by creating views on Performance Schema tables.
- The InnoDB Tablespace Monitor and InnoDB Table Monitor are removed in MySQL 5.7.4. For the Tablespace Monitor, equivalent functionality will be introduced before the GA release of MySQL 5.7. For the Table Monitor, equivalent information can be obtained from InnoDB INFORMATION_SCHEMA tables.
- The specially named tables used to enable and disable the standard InnoDB Monitor and InnoDB Lock Monitor (innodb_monitor and innodb_lock_monitor) are removed in MySQL 5.7.4 and replaced by two dynamic system variables: innodb_status_output and innodb_status_output_locks. For additional information, see Section 14.2.12.4, “InnoDB Monitors”.
- The innodb_use_sys_malloc and innodb_additional_mem_pool_size system variables, which were deprecated in MySQL 5.6.3, are removed in MySQL 5.7.4.

1.5 MySQL Development History

This section describes the general MySQL development history, provides an overview about features that have been implemented in previous series and that are new in MySQL 5.7, the release series covered in this manual. The maturity level this release series is m15. Information about maturity levels can be found in Section 2.1.2.1, “Choosing Which Version of MySQL to Install”.

Before upgrading from one release series to the next, please see the notes in Section 2.10.1, “Upgrading MySQL”.

The most requested features and the versions in which they were implemented are summarized in the following table.

Feature	MySQL Series
Unions	4.0
Subqueries	4.1
R-trees	4.1 (for the MyISAM storage engine)
Stored procedures and functions	5.0
Views	5.0
Cursors	5.0
XA transactions	5.0
Triggers	5.0 and 5.1
Event scheduler	5.1
Partitioning	5.1
Pluggable storage engine API	5.1
Plugin API	5.1
Row-based replication	5.1
Server log tables	5.1
Scalability and performance improvements	5.1 (with InnoDB Plugin)
DTrace support	5.5
Semisynchronous replication	5.5
SIGNAL/RESIGNAL support in stored routines	5.5

Feature	MySQL Series
Performance Schema	5.5
Supplementary Unicode characters	5.5

1.6 MySQL Information Sources

This section lists sources of additional information that you may find helpful, such as the MySQL mailing lists and user forums, and Internet Relay Chat.

1.6.1 MySQL Mailing Lists

This section introduces the MySQL mailing lists and provides guidelines as to how the lists should be used. When you subscribe to a mailing list, you receive all postings to the list as email messages. You can also send your own questions and answers to the list.

To subscribe to or unsubscribe from any of the mailing lists described in this section, visit http://lists.mysql.com/. For most of them, you can select the regular version of the list where you get individual messages, or a digest version where you get one large message per day.

Please *do not* send messages about subscribing or unsubscribing to any of the mailing lists, because such messages are distributed automatically to thousands of other users.

Your local site may have many subscribers to a MySQL mailing list. If so, the site may have a local mailing list, so that messages sent from `lists.mysql.com` to your site are propagated to the local list. In such cases, please contact your system administrator to be added to or dropped from the local MySQL list.

To have traffic for a mailing list go to a separate mailbox in your mail program, set up a filter based on the message headers. You can use either the `List-ID:` or `Delivered-To:` headers to identify list messages.

The MySQL mailing lists are as follows:

- `announce`

  The list for announcements of new versions of MySQL and related programs. This is a low-volume list to which all MySQL users should subscribe.

- `mysql`

  The main list for general MySQL discussion. Please note that some topics are better discussed on the more-specialized lists. If you post to the wrong list, you may not get an answer.

- `bugs`

  The list for people who want to stay informed about issues reported since the last release of MySQL or who want to be actively involved in the process of bug hunting and fixing. See Section 1.7, “How to Report Bugs or Problems”.

- `internals`

  The list for people who work on the MySQL code. This is also the forum for discussions on MySQL development and for posting patches.

- `mysqldoc`

  The list for people who work on the MySQL documentation.

- `benchmarks`

  The list for anyone interested in performance issues. Discussions concentrate on database performance (not limited to MySQL), but also include broader categories such as performance of the kernel, file system, disk system, and so on.

- `packagers`

  The list for discussions on packaging and distributing MySQL. This is the forum used by distribution maintainers to exchange ideas on packaging MySQL and on ensuring that MySQL looks and feels as similar as possible on all supported platforms and operating systems.

- `java`

  The list for discussions about the MySQL server and Java. It is mostly used to discuss JDBC drivers such as MySQL Connector/J.

- `win32`

  The list for all topics concerning the MySQL software on Microsoft operating systems, such as Windows 9x, Me, NT, 2000, XP, and 2003.

- `myodbc`

  The list for all topics concerning connecting to the MySQL server with ODBC.

- `gui-tools`

  The list for all topics concerning MySQL graphical user interface tools such as MySQL Workbench.

- `cluster`

  The list for discussion of MySQL Cluster.

- `dotnet`

  The list for discussion of the MySQL server and the .NET platform. It is mostly related to MySQL Connector/Net.

- `plusplus`

  The list for all topics concerning programming with the C++ API for MySQL.

- `perl`

  The list for all topics concerning Perl support for MySQL with `DBD::mysql`.

If you're unable to get an answer to your questions from a MySQL mailing list or forum, one option is to purchase support from Oracle. This puts you in direct contact with MySQL developers.

The following MySQL mailing lists are in languages other than English. These lists are not operated by Oracle.

- `<mysql-france-subscribe@yahoogroups.com>`

  A French mailing list.

- `<list@tinc.net>`

A Korean mailing list. To subscribe, email `subscribe mysql your@email.address` to this list.

- `<mysql-de-request@lists.4t2.com>`

  A German mailing list. To subscribe, email `subscribe mysql-de your@email.address` to this list. You can find information about this mailing list at http://www.4t2.com/mysql/.

- `<mysql-br-request@listas.linkway.com.br>`

  A Portuguese mailing list. To subscribe, email `subscribe mysql-br your@email.address` to this list.

- `<mysql-alta@elistas.net>`

  A Spanish mailing list. To subscribe, email `subscribe mysql your@email.address` to this list.

1.6.1.1 Guidelines for Using the Mailing Lists

Please do not post mail messages from your browser with HTML mode turned on. Many users do not read mail with a browser.

When you answer a question sent to a mailing list, if you consider your answer to have broad interest, you may want to post it to the list instead of replying directly to the individual who asked. Try to make your answer general enough that people other than the original poster may benefit from it. When you post to the list, please make sure that your answer is not a duplication of a previous answer.

Try to summarize the essential part of the question in your reply. Do not feel obliged to quote the entire original message.

When answers are sent to you individually and not to the mailing list, it is considered good etiquette to summarize the answers and send the summary to the mailing list so that others may have the benefit of responses you received that helped you solve your problem.

1.6.2 MySQL Community Support at the MySQL Forums

The forums at http://forums.mysql.com are an important community resource. Many forums are available, grouped into these general categories:

- Migration
- MySQL Usage
- MySQL Connectors
- Programming Languages
- Tools
- 3rd-Party Applications
- Storage Engines
- MySQL Technology
- SQL Standards
- Business

1.6.3 MySQL Community Support on Internet Relay Chat (IRC)

In addition to the various MySQL mailing lists and forums, you can find experienced community people on Internet Relay Chat (IRC). These are the best networks/channels currently known to us:

freenode (see http://www.freenode.net/ for servers)

- `#mysql` is primarily for MySQL questions, but other database and general SQL questions are welcome. Questions about PHP, Perl, or C in combination with MySQL are also common.

If you are looking for IRC client software to connect to an IRC network, take a look at `xChat` (http://www.xchat.org/). X-Chat (GPL licensed) is available for Unix as well as for Windows platforms (a free Windows build of X-Chat is available at http://www.silverex.org/download/).

1.6.4 MySQL Enterprise

Oracle offers technical support in the form of MySQL Enterprise. For organizations that rely on the MySQL DBMS for business-critical production applications, MySQL Enterprise is a commercial subscription offering which includes:

- MySQL Enterprise Server
- MySQL Enterprise Monitor
- Monthly Rapid Updates and Quarterly Service Packs
- MySQL Knowledge Base
- 24x7 Technical and Consultative Support

MySQL Enterprise is available in multiple tiers, giving you the flexibility to choose the level of service that best matches your needs. For more information, see MySQL Enterprise.

1.7 How to Report Bugs or Problems

Before posting a bug report about a problem, please try to verify that it is a bug and that it has not been reported already:

- Start by searching the MySQL online manual at http://dev.mysql.com/doc/. We try to keep the manual up to date by updating it frequently with solutions to newly found problems. In addition, the release notes accompanying the manual can be particularly useful since it is quite possible that a newer version contains a solution to your problem. The release notes are available at the location just given for the manual.
- If you get a parse error for an SQL statement, please check your syntax closely. If you cannot find something wrong with it, it is extremely likely that your current version of MySQL Server doesn't support the syntax you are using. If you are using the current version and the manual doesn't cover the syntax that you are using, MySQL Server doesn't support your statement.

  If the manual covers the syntax you are using, but you have an older version of MySQL Server, you should check the MySQL change history to see when the syntax was implemented. In this case, you have the option of upgrading to a newer version of MySQL Server.
- For solutions to some common problems, see Section C.5, “Problems and Common Errors”.
- Search the bugs database at http://bugs.mysql.com/ to see whether the bug has been reported and fixed.
- Search the MySQL mailing list archives at http://lists.mysql.com/. See Section 1.6.1, “MySQL Mailing Lists”.

- You can also use http://www.mysql.com/search/ to search all the Web pages (including the manual) that are located at the MySQL Web site.

If you cannot find an answer in the manual, the bugs database, or the mailing list archives, check with your local MySQL expert. If you still cannot find an answer to your question, please use the following guidelines for reporting the bug.

The normal way to report bugs is to visit http://bugs.mysql.com/, which is the address for our bugs database. This database is public and can be browsed and searched by anyone. If you log in to the system, you can enter new reports.

Bugs posted in the bugs database at http://bugs.mysql.com/ that are corrected for a given release are noted in the release notes.

If you find a sensitive security bug in MySQL Server, please let us know immediately by sending an email message to `<secalert_us@oracle.com>`. Exception: Support customers should report all problems, including security bugs, to Oracle Support at http://support.oracle.com/.

To discuss problems with other users, you can use one of the MySQL mailing lists. Section 1.6.1, “MySQL Mailing Lists”.

Writing a good bug report takes patience, but doing it right the first time saves time both for us and for yourself. A good bug report, containing a full test case for the bug, makes it very likely that we will fix the bug in the next release. This section helps you write your report correctly so that you do not waste your time doing things that may not help us much or at all. Please read this section carefully and make sure that all the information described here is included in your report.

Preferably, you should test the problem using the latest production or development version of MySQL Server before posting. Anyone should be able to repeat the bug by just using `mysql test < script_file` on your test case or by running the shell or Perl script that you include in the bug report. Any bug that we are able to repeat has a high chance of being fixed in the next MySQL release.

It is most helpful when a good description of the problem is included in the bug report. That is, give a good example of everything you did that led to the problem and describe, in exact detail, the problem itself. The best reports are those that include a full example showing how to reproduce the bug or problem. See Section 22.4, “Debugging and Porting MySQL”.

Remember that it is possible for us to respond to a report containing too much information, but not to one containing too little. People often omit facts because they think they know the cause of a problem and assume that some details do not matter. A good principle to follow is that if you are in doubt about stating something, state it. It is faster and less troublesome to write a couple more lines in your report than to wait longer for the answer if we must ask you to provide information that was missing from the initial report.

The most common errors made in bug reports are (a) not including the version number of the MySQL distribution that you use, and (b) not fully describing the platform on which the MySQL server is installed (including the platform type and version number). These are highly relevant pieces of information, and in 99 cases out of 100, the bug report is useless without them. Very often we get questions like, “Why doesn't this work for me?” Then we find that the feature requested wasn't implemented in that MySQL version, or that a bug described in a report has been fixed in newer MySQL versions. Errors often are platform-dependent. In such cases, it is next to impossible for us to fix anything without knowing the operating system and the version number of the platform.

If you compiled MySQL from source, remember also to provide information about your compiler if it is related to the problem. Often people find bugs in compilers and think the problem is MySQL-related. Most compilers are under development all the time and become better version by version. To determine whether your problem depends on your compiler, we need to know what compiler you used. Note that every compiling problem should be regarded as a bug and reported accordingly.

If a program produces an error message, it is very important to include the message in your report. If we try to search for something from the archives, it is better that the error message reported exactly matches the one that the program produces. (Even the lettercase should be observed.) It is best to copy and paste the entire error message into your report. You should never try to reproduce the message from memory.

If you have a problem with Connector/ODBC (MyODBC), please try to generate a trace file and send it with your report. See How to Report Connector/ODBC Problems or Bugs.

If your report includes long query output lines from test cases that you run with the `mysql` command-line tool, you can make the output more readable by using the `--vertical` option or the `\G` statement terminator. The `EXPLAIN SELECT` example later in this section demonstrates the use of `\G`.

Please include the following information in your report:

- The version number of the MySQL distribution you are using (for example, MySQL 5.0.19). You can find out which version you are running by executing `mysqladmin version`. The `mysqladmin` program can be found in the `bin` directory under your MySQL installation directory.

- The manufacturer and model of the machine on which you experience the problem.

- The operating system name and version. If you work with Windows, you can usually get the name and version number by double-clicking your My Computer icon and pulling down the “Help/About Windows” menu. For most Unix-like operating systems, you can get this information by executing the command `uname -a`.

- Sometimes the amount of memory (real and virtual) is relevant. If in doubt, include these values.

- If you are using a source distribution of the MySQL software, include the name and version number of the compiler that you used. If you have a binary distribution, include the distribution name.

- If the problem occurs during compilation, include the exact error messages and also a few lines of context around the offending code in the file where the error occurs.

- If `mysqld` died, you should also report the statement that crashed `mysqld`. You can usually get this information by running `mysqld` with query logging enabled, and then looking in the log after `mysqld` crashes. See Section 22.4, “Debugging and Porting MySQL”.

- If a database table is related to the problem, include the output from the `SHOW CREATE TABLE` *`db_name.tbl_name`* statement in the bug report. This is a very easy way to get the definition of any table in a database. The information helps us create a situation matching the one that you have experienced.

- The SQL mode in effect when the problem occurred can be significant, so please report the value of the `sql_mode` system variable. For stored procedure, stored function, and trigger objects, the relevant `sql_mode` value is the one in effect when the object was created. For a stored procedure or function, the `SHOW CREATE PROCEDURE` or `SHOW CREATE FUNCTION` statement shows the relevant SQL mode, or you can query `INFORMATION_SCHEMA` for the information:

  ```
  SELECT ROUTINE_SCHEMA, ROUTINE_NAME, SQL_MODE
  FROM INFORMATION_SCHEMA.ROUTINES;
  ```

  For triggers, you can use this statement:

  ```
  SELECT EVENT_OBJECT_SCHEMA, EVENT_OBJECT_TABLE, TRIGGER_NAME, SQL_MODE
  ```

```
FROM INFORMATION_SCHEMA.TRIGGERS;
```

- For performance-related bugs or problems with SELECT statements, you should always include the output of EXPLAIN SELECT ..., and at least the number of rows that the SELECT statement produces. You should also include the output from SHOW CREATE TABLE *tbl_name* for each table that is involved. The more information you provide about your situation, the more likely it is that someone can help you.

  The following is an example of a very good bug report. The statements are run using the mysql command-line tool. Note the use of the \G statement terminator for statements that would otherwise provide very long output lines that are difficult to read.

  ```
  mysql> SHOW VARIABLES;
  mysql> SHOW COLUMNS FROM ...\G
         <output from SHOW COLUMNS>
  mysql> EXPLAIN SELECT ...\G
         <output from EXPLAIN>
  mysql> FLUSH STATUS;
  mysql> SELECT ...;
         <A short version of the output from SELECT,
         including the time taken to run the query>
  mysql> SHOW STATUS;
         <output from SHOW STATUS>
  ```

- If a bug or problem occurs while running mysqld, try to provide an input script that reproduces the anomaly. This script should include any necessary source files. The more closely the script can reproduce your situation, the better. If you can make a reproducible test case, you should upload it to be attached to the bug report.

  If you cannot provide a script, you should at least include the output from mysqladmin variables extended-status processlist in your report to provide some information on how your system is performing.

- If you cannot produce a test case with only a few rows, or if the test table is too big to be included in the bug report (more than 10 rows), you should dump your tables using mysqldump and create a README file that describes your problem. Create a compressed archive of your files using tar and gzip or zip. After you initiate a bug report for our bugs database at http://bugs.mysql.com/, click the Files tab in the bug report for instructions on uploading the archive to the bugs database.

- If you believe that the MySQL server produces a strange result from a statement, include not only the result, but also your opinion of what the result should be, and an explanation describing the basis for your opinion.

- When you provide an example of the problem, it is better to use the table names, variable names, and so forth that exist in your actual situation than to come up with new names. The problem could be related to the name of a table or variable. These cases are rare, perhaps, but it is better to be safe than sorry. After all, it should be easier for you to provide an example that uses your actual situation, and it is by all means better for us. If you have data that you do not want to be visible to others in the bug report, you can upload it using the Files tab as previously described. If the information is really top secret and you do not want to show it even to us, go ahead and provide an example using other names, but please regard this as the last choice.

- Include all the options given to the relevant programs, if possible. For example, indicate the options that you use when you start the mysqld server, as well as the options that you use to run any MySQL client programs. The options to programs such as mysqld and mysql, and to the configure script, are often key to resolving problems and are very relevant. It is never a bad idea to include them. If your problem involves a program written in a language such as Perl or PHP, please include the language processor's version number, as well as the version for any modules that the program uses. For example, if you have a Perl script that uses the DBI and DBD::mysql modules, include the version numbers for Perl, DBI, and DBD::mysql.

- If your question is related to the privilege system, please include the output of mysqlaccess, the output of mysqladmin reload, and all the error messages you get when trying to connect. When you test your privileges, you should first run mysqlaccess. After this, execute mysqladmin reload version and try to connect with the program that gives you trouble. mysqlaccess can be found in the bin directory under your MySQL installation directory.
- If you have a patch for a bug, do include it. But do not assume that the patch is all we need, or that we can use it, if you do not provide some necessary information such as test cases showing the bug that your patch fixes. We might find problems with your patch or we might not understand it at all. If so, we cannot use it.

  If we cannot verify the exact purpose of the patch, we will not use it. Test cases help us here. Show that the patch handles all the situations that may occur. If we find a borderline case (even a rare one) where the patch will not work, it may be useless.
- Guesses about what the bug is, why it occurs, or what it depends on are usually wrong. Even the MySQL team cannot guess such things without first using a debugger to determine the real cause of a bug.
- Indicate in your bug report that you have checked the reference manual and mail archive so that others know you have tried to solve the problem yourself.
- If your data appears corrupt or you get errors when you access a particular table, first check your tables with CHECK TABLE. If that statement reports any errors:
  - The InnoDB crash recovery mechanism handles cleanup when the server is restarted after being killed, so in typical operation there is no need to “repair” tables. If you encounter an error with InnoDB tables, restart the server and see whether the problem persists, or whether the error affected only cached data in memory. If data is corrupted on disk, consider restarting with the innodb_force_recovery option enabled so that you can dump the affected tables.
  - For non-transactional tables, try to repair them with REPAIR TABLE or with myisamchk. See Chapter 5, *MySQL Server Administration*.

  If you are running Windows, please verify the value of lower_case_table_names using the SHOW VARIABLES LIKE 'lower_case_table_names' statement. This variable affects how the server handles lettercase of database and table names. Its effect for a given value should be as described in Section 9.2.2, “Identifier Case Sensitivity”.
- If you often get corrupted tables, you should try to find out when and why this happens. In this case, the error log in the MySQL data directory may contain some information about what happened. (This is the file with the .err suffix in the name.) See Section 5.2.2, “The Error Log”. Please include any relevant information from this file in your bug report. Normally mysqld should *never* crash a table if nothing killed it in the middle of an update. If you can find the cause of mysqld dying, it is much easier for us to provide you with a fix for the problem. See Section C.5.1, “How to Determine What Is Causing a Problem”.
- If possible, download and install the most recent version of MySQL Server and check whether it solves your problem. All versions of the MySQL software thoroughly tested and should work without problems. We believe in making everything as backward-compatible as possible, and you should be able to switch MySQL versions without difficulty. See Section 2.1.2, “Choosing Which MySQL Distribution to Install”.

1.8 MySQL Standards Compliance

This section describes how MySQL relates to the ANSI/ISO SQL standards. MySQL Server has many extensions to the SQL standard, and here you can find out what they are and how to use them. You can also find information about functionality missing from MySQL Server, and how to work around some of the differences.

The SQL standard has been evolving since 1986 and several versions exist. In this manual, “SQL-92” refers to the standard released in 1992, “SQL:1999” refers to the standard released in 1999, “SQL:2003” refers to the standard released in 2003, and “SQL:2008” refers to the most recent version of the standard, released in 2008. We use the phrase “the SQL standard” or “standard SQL” to mean the current version of the SQL Standard at any time.

One of our main goals with the product is to continue to work toward compliance with the SQL standard, but without sacrificing speed or reliability. We are not afraid to add extensions to SQL or support for non-SQL features if this greatly increases the usability of MySQL Server for a large segment of our user base. The `HANDLER` interface is an example of this strategy. See Section 13.2.4, “`HANDLER` Syntax”.

We continue to support transactional and nontransactional databases to satisfy both mission-critical 24/7 usage and heavy Web or logging usage.

MySQL Server was originally designed to work with medium-sized databases (10-100 million rows, or about 100MB per table) on small computer systems. Today MySQL Server handles terabyte-sized databases, but the code can also be compiled in a reduced version suitable for hand-held and embedded devices. The compact design of the MySQL server makes development in both directions possible without any conflicts in the source tree.

Currently, we are not targeting real-time support, although MySQL replication capabilities offer significant functionality.

MySQL supports ODBC levels 0 to 3.51.

MySQL supports high-availability database clustering using the `NDBCLUSTER` storage engine. See MySQL Cluster NDB 7.3.

We implement XML functionality which supports most of the W3C XPath standard. See Section 12.11, “XML Functions”.

Selecting SQL Modes

The MySQL server can operate in different SQL modes, and can apply these modes differently for different clients, depending on the value of the `sql_mode` system variable. DBAs can set the global SQL mode to match site server operating requirements, and each application can set its session SQL mode to its own requirements.

Modes affect the SQL syntax MySQL supports and the data validation checks it performs. This makes it easier to use MySQL in different environments and to use MySQL together with other database servers.

For more information on setting the SQL mode, see Section 5.1.7, “Server SQL Modes”.

Running MySQL in ANSI Mode

To run MySQL Server in ANSI mode, start `mysqld` with the `--ansi` option. Running the server in ANSI mode is the same as starting it with the following options:

```
--transaction-isolation=SERIALIZABLE --sql-mode=ANSI
```

To achieve the same effect at runtime, execute these two statements:

```
SET GLOBAL TRANSACTION ISOLATION LEVEL SERIALIZABLE;
SET GLOBAL sql_mode = 'ANSI';
```

You can see that setting the `sql_mode` system variable to `'ANSI'` enables all SQL mode options that are relevant for ANSI mode as follows:

```
mysql> SET GLOBAL sql_mode='ANSI';
mysql> SELECT @@global.sql_mode;
        -> 'REAL_AS_FLOAT,PIPES_AS_CONCAT,ANSI_QUOTES,IGNORE_SPACE,ANSI'
```

Running the server in ANSI mode with `--ansi` is not quite the same as setting the SQL mode to `'ANSI'` because the `--ansi` option also sets the transaction isolation level.

See Section 5.1.3, “Server Command Options”.

1.8.1 MySQL Extensions to Standard SQL

MySQL Server supports some extensions that you probably won't find in other SQL DBMSs. Be warned that if you use them, your code won't be portable to other SQL servers. In some cases, you can write code that includes MySQL extensions, but is still portable, by using comments of the following form:

```
/*! MySQL-specific code */
```

In this case, MySQL Server parses and executes the code within the comment as it would any other SQL statement, but other SQL servers will ignore the extensions. For example, MySQL Server recognizes the `STRAIGHT_JOIN` keyword in the following statement, but other servers will not:

```
SELECT /*! STRAIGHT_JOIN */ col1 FROM table1,table2 WHERE ...
```

If you add a version number after the “`!`” character, the syntax within the comment is executed only if the MySQL version is greater than or equal to the specified version number. The `TEMPORARY` keyword in the following comment is executed only by servers from MySQL 3.23.02 or higher:

```
CREATE /*!32302 TEMPORARY */ TABLE t (a INT);
```

The following descriptions list MySQL extensions, organized by category.

- Organization of data on disk

  MySQL Server maps each database to a directory under the MySQL data directory, and maps tables within a database to file names in the database directory. This has a few implications:

  - Database and table names are case sensitive in MySQL Server on operating systems that have case-sensitive file names (such as most Unix systems). See Section 9.2.2, “Identifier Case Sensitivity”.

  - You can use standard system commands to back up, rename, move, delete, and copy tables that are managed by the `MyISAM` storage engine. For example, it is possible to rename a `MyISAM` table by renaming the `.MYD`, `.MYI`, and `.frm` files to which the table corresponds. (Nevertheless, it is preferable to use `RENAME TABLE` or `ALTER TABLE ... RENAME` and let the server rename the files.)

- General language syntax

  - By default, strings can be enclosed by either “`"`” or “`'`”, not just by “`'`”. (If the `ANSI_QUOTES` SQL mode is enabled, strings can be enclosed only by “`'`” and the server interprets strings enclosed by “`"`” as identifiers.)

  - "\" is the escape character in strings.
  - In SQL statements, you can access tables from different databases with the *db_name.tbl_name* syntax. Some SQL servers provide the same functionality but call this User space. MySQL Server doesn't support tablespaces such as used in statements like this: CREATE TABLE ralph.my_table ... IN my_tablespace.

- SQL statement syntax
  - The ANALYZE TABLE, CHECK TABLE, OPTIMIZE TABLE, and REPAIR TABLE statements.
  - The CREATE DATABASE, DROP DATABASE, and ALTER DATABASE statements. See Section 13.1.8, "CREATE DATABASE Syntax", Section 13.1.17, "DROP DATABASE Syntax", and Section 13.1.1, "ALTER DATABASE Syntax".
  - The DO statement.
  - EXPLAIN SELECT to obtain a description of how tables are processed by the query optimizer.
  - The FLUSH and RESET statements.
  - The SET statement. See Section 13.7.4, "SET Syntax".
  - The SHOW statement. See Section 13.7.5, "SHOW Syntax". The information produced by many of the MySQL-specific SHOW statements can be obtained in more standard fashion by using SELECT to query INFORMATION_SCHEMA. See Chapter 19, *INFORMATION_SCHEMA Tables*.
  - Use of LOAD DATA INFILE. In many cases, this syntax is compatible with Oracle's LOAD DATA INFILE. See Section 13.2.6, "LOAD DATA INFILE Syntax".
  - Use of RENAME TABLE. See Section 13.1.26, "RENAME TABLE Syntax".
  - Use of REPLACE instead of DELETE plus INSERT. See Section 13.2.8, "REPLACE Syntax".
  - Use of CHANGE *col_name*, DROP *col_name*, or DROP INDEX, IGNORE or RENAME in ALTER TABLE statements. Use of multiple ADD, ALTER, DROP, or CHANGE clauses in an ALTER TABLE statement. See Section 13.1.6, "ALTER TABLE Syntax".
  - Use of index names, indexes on a prefix of a column, and use of INDEX or KEY in CREATE TABLE statements. See Section 13.1.14, "CREATE TABLE Syntax".
  - Use of TEMPORARY or IF NOT EXISTS with CREATE TABLE.
  - Use of IF EXISTS with DROP TABLE and DROP DATABASE.
  - The capability of dropping multiple tables with a single DROP TABLE statement.
  - The ORDER BY and LIMIT clauses of the UPDATE and DELETE statements.
  - INSERT INTO *tbl_name* SET *col_name* = ... syntax.
  - The DELAYED clause of the INSERT and REPLACE statements.
  - The LOW_PRIORITY clause of the INSERT, REPLACE, DELETE, and UPDATE statements.
  - Use of INTO OUTFILE or INTO DUMPFILE in SELECT statements. See Section 13.2.9, "SELECT Syntax".

  - Options such as STRAIGHT_JOIN or SQL_SMALL_RESULT in SELECT statements.
  - You don't need to name all selected columns in the GROUP BY clause. This gives better performance for some very specific, but quite normal queries. See Section 12.17, "Functions and Modifiers for Use with GROUP BY Clauses".
  - You can specify ASC and DESC with GROUP BY, not just with ORDER BY.
  - The ability to set variables in a statement with the := assignment operator. See Section 9.4, "User-Defined Variables".
- Data types
  - The MEDIUMINT, SET, and ENUM data types, and the various BLOB and TEXT data types.
  - The AUTO_INCREMENT, BINARY, NULL, UNSIGNED, and ZEROFILL data type attributes.
- Functions and operators
  - To make it easier for users who migrate from other SQL environments, MySQL Server supports aliases for many functions. For example, all string functions support both standard SQL syntax and ODBC syntax.
  - MySQL Server understands the || and && operators to mean logical OR and AND, as in the C programming language. In MySQL Server, || and OR are synonyms, as are && and AND. Because of this nice syntax, MySQL Server doesn't support the standard SQL || operator for string concatenation; use CONCAT() instead. Because CONCAT() takes any number of arguments, it is easy to convert use of the || operator to MySQL Server.
  - Use of COUNT(DISTINCT *value_list*) where *value_list* has more than one element.
  - String comparisons are case-insensitive by default, with sort ordering determined by the collation of the current character set, which is latin1 (cp1252 West European) by default. If you don't like this, you should declare your columns with the BINARY attribute or use the BINARY cast, which causes comparisons to be done using the underlying character code values rather than a lexical ordering.
  - The % operator is a synonym for MOD(). That is, *N* % *M* is equivalent to MOD(*N*,*M*). % is supported for C programmers and for compatibility with PostgreSQL.
  - The =, <>, <=, <, >=, >, <<, >>, <=>, AND, OR, or LIKE operators may be used in expressions in the output column list (to the left of the FROM) in SELECT statements. For example:

```
mysql> SELECT col1=1 AND col2=2 FROM my_table;
```

  - The LAST_INSERT_ID() function returns the most recent AUTO_INCREMENT value. See Section 12.14, "Information Functions".
  - LIKE is permitted on numeric values.
  - The REGEXP and NOT REGEXP extended regular expression operators.
  - CONCAT() or CHAR() with one argument or more than two arguments. (In MySQL Server, these functions can take a variable number of arguments.)

- The BIT_COUNT(), CASE, ELT(), FROM_DAYS(), FORMAT(), IF(), PASSWORD(), ENCRYPT(), MD5(), ENCODE(), DECODE(), PERIOD_ADD(), PERIOD_DIFF(), TO_DAYS(), and WEEKDAY() functions.
- Use of TRIM() to trim substrings. Standard SQL supports removal of single characters only.
- The GROUP BY functions STD(), BIT_OR(), BIT_AND(), BIT_XOR(), and GROUP_CONCAT(). See Section 12.17, “Functions and Modifiers for Use with GROUP BY Clauses”.

1.8.2 MySQL Differences from Standard SQL

We try to make MySQL Server follow the ANSI SQL standard and the ODBC SQL standard, but MySQL Server performs operations differently in some cases:

- There are several differences between the MySQL and standard SQL privilege systems. For example, in MySQL, privileges for a table are not automatically revoked when you delete a table. You must explicitly issue a REVOKE statement to revoke privileges for a table. For more information, see Section 13.7.1.6, “REVOKE Syntax”.
- The CAST() function does not support cast to REAL or BIGINT. See Section 12.10, “Cast Functions and Operators”.

1.8.2.1 SELECT INTO TABLE Differences

MySQL Server doesn't support the SELECT ... INTO TABLE Sybase SQL extension. Instead, MySQL Server supports the INSERT INTO ... SELECT standard SQL syntax, which is basically the same thing. See Section 13.2.5.1, “INSERT ... SELECT Syntax”. For example:

```
INSERT INTO tbl_temp2 (fld_id)
    SELECT tbl_temp1.fld_order_id
    FROM tbl_temp1 WHERE tbl_temp1.fld_order_id > 100;
```

Alternatively, you can use SELECT ... INTO OUTFILE or CREATE TABLE ... SELECT.

You can use SELECT ... INTO with user-defined variables. The same syntax can also be used inside stored routines using cursors and local variables. See Section 13.2.9.1, “SELECT ... INTO Syntax”.

1.8.2.2 UPDATE Differences

If you access a column from the table to be updated in an expression, UPDATE uses the current value of the column. The second assignment in the following statement sets col2 to the current (updated) col1 value, not the original col1 value. The result is that col1 and col2 have the same value. This behavior differs from standard SQL.

```
UPDATE t1 SET col1 = col1 + 1, col2 = col1;
```

1.8.2.3 Transaction and Atomic Operation Differences

MySQL Server (version 3.23-max and all versions 4.0 and above) supports transactions with the InnoDB transactional storage engine. In MySQL 5.5 and up, newly created tables use InnoDB by default, as explained in Section 14.2.1.1, “InnoDB as the Default MySQL Storage Engine”. By default, InnoDB provides *full* ACID compliance; see Section 14.2.2.1, “MySQL and the ACID Model” for ways that you can adjust settings to balance ACID compliance with raw performance. For information about InnoDB differences from standard SQL with regard to treatment of transaction errors, see Section 14.2.17.4, “InnoDB Error Handling”.

The nontransactional storage engines in MySQL Server (such as MyISAM) follow a different paradigm for data integrity called “atomic operations”. MyISAM tables effectively always operate in autocommit = 1 mode. Because changed data is written to disk one statement at a time, it is harder to guarantee the consistency of a sequence of related DML operations, which could be interrupted partway through. Thus, this mode is suitable for read-mostly workloads. In transactional terms, while each specific update is running, no other user can interfere with it, there can never be an automatic rollback, and there are no dirty reads. However, these features apply to single operations, not related updates that succeed or fail as a unit. Workarounds such as the LOCK TABLES statement limit concurrent write access to nontransactional tables.

You can choose which paradigm to use, even for different tables within the same application: transactional features for reliability combined with high performance, or atomic operations for non-critical, read-mostly data (for example, on replication slave servers).

Transactional storage engines such as InnoDB offer many significant features to support high reliability for heavy read/write workloads. As a result, transactional tables can have higher memory and disk space requirements, and more CPU overhead. MySQL Server's modular design enables the concurrent use of different storage engines to suit different requirements and deliver optimum performance in all situations.

Workarounds for Reliability with Non-Transactional Tables

But how do you use the features of MySQL Server to maintain integrity even with the nontransactional MyISAM tables, and how do these features compare with the transactional storage engines?

- If your applications are written in a way that is dependent on being able to call ROLLBACK rather than COMMIT in critical situations, transactions are more convenient. Transactions also ensure that unfinished updates or corrupting activities are not committed to the database; the server is given the opportunity to do an automatic rollback and your database is saved.

  If you use nontransactional tables, you must resolve potential problems at the application level by including checks before updates and by running scripts that check the databases for inconsistencies and automatically repair or warn if such an inconsistency occurs. You can normally fix tables with no data integrity loss by using the MySQL log or even adding one extra log.

- Sometimes, critical transactional updates can be rewritten to be atomic. Multiple DML operations can be done with LOCK TABLES or atomic updates, ensuring that there are no deadlocks by limiting concurrent write access. If you obtain a READ LOCAL lock (as opposed to a write lock) for a table that enables concurrent inserts at the end of the table, reads are permitted, as are inserts by other clients. The newly inserted records are not be seen by the client that has the read lock until it releases the lock. With INSERT DELAYED, you can write inserts that go into a local queue until the locks are released, without having the client wait for the insert to complete. See Section 8.10.3, “Concurrent Inserts”, and Section 13.2.5.2, “INSERT DELAYED Syntax”.

- To be safe with MySQL Server, regardless of what kinds of tables you use, make regular backups and have binary logging turned on. It is always good to have backups, regardless of which database system you use.

Following are some techniques for working with nontransactional tables:

- Loops that need transactions normally can be coded with the help of LOCK TABLES, and you don't need cursors to update records on the fly.

- To avoid using ROLLBACK, you can employ the following strategy:

  1. Use LOCK TABLES to lock all the tables you want to access.

  2. Test the conditions that must be true before performing the update.

3. Update if the conditions are satisfied.

4. Use UNLOCK TABLES to release your locks.

> **Note**
>
> This solution does not handle the situation when someone kills the threads in the middle of an update. In that case, all locks are released but some of the updates may not have been executed.

- You can also use functions to update records in a single operation, using the following techniques:

  - Modify columns relative to their current value. This makes the update correct even if another client has changed the column values in the meantime.

  - Update only those columns that actually have changed. This is a good database practice in general.

- When managing unique identifiers, you can avoid statements such as LOCK TABLES or ROLLBACK by using an AUTO_INCREMENT column and either the LAST_INSERT_ID() SQL function or the mysql_insert_id() C API function. See Section 12.14, "Information Functions", and Section 21.8.7.38, "mysql_insert_id()".

  For situations that require row-level locking, use InnoDB tables. Otherwise, with MyISAM tables, you can use a flag column in the table and do something like the following:

  ```
  UPDATE tbl_name SET row_flag=1 WHERE id=ID;
  ```

  MySQL returns 1 for the number of affected rows if the row was found and row_flag wasn't 1 in the original row. You can think of this as though MySQL Server changed the preceding statement to:

  ```
  UPDATE tbl_name SET row_flag=1 WHERE id=ID AND row_flag <> 1;
  ```

1.8.2.4 Foreign Key Differences

The InnoDB storage engine supports checking of foreign key constraints, including CASCADE, ON DELETE, and ON UPDATE. See Section 14.2.6.6, "InnoDB and FOREIGN KEY Constraints".

For storage engines other than InnoDB, MySQL Server parses the FOREIGN KEY syntax in CREATE TABLE statements, but does not use or store it. This information is also present in mysqldump, and can be retrieved using Connector/ODBC. You can see which tables have foreign key constraints by checking the INFORMATION_SCHEMA.TABLE_CONSTRAINTS table in the INFORMATION_SCHEMA information database. You can obtain more detailed information about foreign keys from the INFORMATION_SCHEMA.REFERENTIAL_CONSTRAINTS table. In addition, InnoDB provides a number of INFORMATION_SCHEMA tables containing information about foreign keys on InnoDB tables; see Section 19.30, "INFORMATION_SCHEMA Tables for InnoDB".

Foreign key enforcement offers several benefits to database developers:

- Assuming proper design of the relationships, foreign key constraints make it more difficult for a programmer to introduce an inconsistency into the database.

- Centralized checking of constraints by the database server makes it unnecessary to perform these checks on the application side. This eliminates the possibility that different applications may not all check the constraints in the same way.

- Using cascading updates and deletes can simplify the application code.

- Properly designed foreign key rules aid in documenting relationships between tables.

Foreign keys in SQL are used to check and enforce referential integrity, not to join tables. If you want to get results from multiple tables from a SELECT statement, you do this by performing a join between them:

```
SELECT * FROM t1 INNER JOIN t2 ON t1.id = t2.id;
```

See Section 13.2.9.2, “JOIN Syntax”, and Section 3.6.6, “Using Foreign Keys”.

The FOREIGN KEY syntax without ON DELETE ... is often used by ODBC applications to produce automatic WHERE clauses.

1.8.2.5 '--' as the Start of a Comment

Standard SQL uses the C syntax /* this is a comment */ for comments, and MySQL Server supports this syntax as well. MySQL also support extensions to this syntax that enable MySQL-specific SQL to be embedded in the comment, as described in Section 9.6, “Comment Syntax”.

Standard SQL uses “--” as a start-comment sequence. MySQL Server uses “#” as the start comment character. MySQL Server 3.23.3 and up also supports a variant of the “--” comment style. That is, the “--” start-comment sequence must be followed by a space (or by a control character such as a newline). The space is required to prevent problems with automatically generated SQL queries that use constructs such as the following, where we automatically insert the value of the payment for payment:

```
UPDATE account SET credit=credit-payment
```

Consider about what happens if payment has a negative value such as -1:

```
UPDATE account SET credit=credit--1
```

credit--1 is a valid expression in SQL, but “--” is interpreted as the start of a comment, part of the expression is discarded. The result is a statement that has a completely different meaning than intended:

```
UPDATE account SET credit=credit
```

The statement produces no change in value at all. This illustrates that permitting comments to start with “--” can have serious consequences.

Using our implementation requires a space following the “--” for it to be recognized as a start-comment sequence in MySQL Server 3.23.3 and newer. Therefore, credit--1 is safe to use.

Another safe feature is that the mysql command-line client ignores lines that start with “--”.

The following information is relevant only if you are running a MySQL version earlier than 3.23.3:

If you have an SQL script in a text file that contains “--” comments, you should use the replace utility as follows to convert the comments to use “#” characters before executing the script:

```
shell> replace " --" " #" < text-file-with-funny-comments.sql \
         | mysql db_name
```

That is safer than executing the script in the usual way:

```
shell> mysql db_name < text-file-with-funny-comments.sql
```

You can also edit the script file “in place” to change the “--” comments to “#” comments:

```
shell> replace " --" " #" -- text-file-with-funny-comments.sql
```

Change them back with this command:

```
shell> replace " #" " --" -- text-file-with-funny-comments.sql
```

See Section 4.8.2, “replace — A String-Replacement Utility”.

1.8.3 How MySQL Deals with Constraints

MySQL enables you to work both with transactional tables that permit rollback and with nontransactional tables that do not. Because of this, constraint handling is a bit different in MySQL than in other DBMSs. We must handle the case when you have inserted or updated a lot of rows in a nontransactional table for which changes cannot be rolled back when an error occurs.

The basic philosophy is that MySQL Server tries to produce an error for anything that it can detect while parsing a statement to be executed, and tries to recover from any errors that occur while executing the statement. We do this in most cases, but not yet for all.

The options MySQL has when an error occurs are to stop the statement in the middle or to recover as well as possible from the problem and continue. By default, the server follows the latter course. This means, for example, that the server may coerce invalid values to the closest valid values.

Several SQL mode options are available to provide greater control over handling of bad data values and whether to continue statement execution or abort when errors occur. Using these options, you can configure MySQL Server to act in a more traditional fashion that is like other DBMSs that reject improper input. The SQL mode can be set globally at server startup to affect all clients. Individual clients can set the SQL mode at runtime, which enables each client to select the behavior most appropriate for its requirements. See Section 5.1.7, “Server SQL Modes”.

The following sections describe how MySQL Server handles different types of constraints.

1.8.3.1 PRIMARY KEY and UNIQUE Index Constraints

Normally, errors occurs for data-change statements (such as INSERT or UPDATE) that would violate primary-key, unique-key, or foreign-key constraints. If you are using a transactional storage engine such as InnoDB, MySQL automatically rolls back the statement. If you are using a nontransactional storage engine, MySQL stops processing the statement at the row for which the error occurred and leaves any remaining rows unprocessed.

MySQL supports an IGNORE keyword for INSERT, UPDATE, and so forth. If you use it, MySQL ignores primary-key or unique-key violations and continues processing with the next row. See the section for the statement that you are using (Section 13.2.5, “INSERT Syntax”, Section 13.2.11, “UPDATE Syntax”, and so forth).

You can get information about the number of rows actually inserted or updated with the mysql_info() C API function. You can also use the SHOW WARNINGS statement. See Section 21.8.7.36, “mysql_info()”, and Section 13.7.5.39, “SHOW WARNINGS Syntax”.

Currently, only InnoDB tables support foreign keys. See Section 14.2.6.6, "InnoDB and FOREIGN KEY Constraints".

1.8.3.2 FOREIGN KEY Constraints

Foreign keys let you cross-reference related data across tables, and foreign key constraints help keep this spread-out data consistent.

MySQL supports ON UPDATE and ON DELETE foreign key references in CREATE TABLE and ALTER TABLE statements. The available referential actions are RESTRICT (the default), CASCADE, SET NULL, and NO ACTION.

SET DEFAULT is also supported by the MySQL Server but is currently rejected as invalid by InnoDB. Since MySQL does not support deferred constraint checking, NO ACTION is treated as RESTRICT. For the exact syntax supported by MySQL for foreign keys, see Section 13.1.14.2, "Using FOREIGN KEY Constraints".

MATCH FULL, MATCH PARTIAL, and MATCH SIMPLE are allowed, but their use should be avoided, as they cause the MySQL Server to ignore any ON DELETE or ON UPDATE clause used in the same statement. MATCH options do not have any other effect in MySQL, which in effect enforces MATCH SIMPLE semantics full-time.

MySQL requires that foreign key columns be indexed; if you create a table with a foreign key constraint but no index on a given column, an index is created.

You can obtain information about foreign keys from the INFORMATION_SCHEMA.KEY_COLUMN_USAGE table. An example of a query against this table is shown here:

```
mysql> SELECT TABLE_SCHEMA, TABLE_NAME, COLUMN_NAME, CONSTRAINT_NAME
     > FROM INFORMATION_SCHEMA.KEY_COLUMN_USAGE
     > WHERE REFERENCED_TABLE_SCHEMA IS NOT NULL;
+--------------+---------------+-------------+-----------------+
| TABLE_SCHEMA | TABLE_NAME    | COLUMN_NAME | CONSTRAINT_NAME |
+--------------+---------------+-------------+-----------------+
| fk1          | myuser        | myuser_id   | f               |
| fk1          | product_order | customer_id | f2              |
| fk1          | product_order | product_id  | f1              |
+--------------+---------------+-------------+-----------------+
3 rows in set (0.01 sec)
```

Information about foreign keys on InnoDB tables can also be found in the INNODB_SYS_FOREIGN and INNODB_SYS_FOREIGN_COLS tables, in the INFORMATION_SCHEMA database.

Currently, only InnoDB tables support foreign keys. See Section 14.2.6.6, "InnoDB and FOREIGN KEY Constraints", for information specific to foreign key support in InnoDB.

Deviations from SQL Standards

MySQL's implementation of foreign keys differs from the SQL standard in the following key respects:

- If there are several rows in the parent table that have the same referenced key value, InnoDB acts in foreign key checks as if the other parent rows with the same key value do not exist. For example, if you have defined a RESTRICT type constraint, and there is a child row with several parent rows, InnoDB does not permit the deletion of any of those parent rows.

  InnoDB performs cascading operations through a depth-first algorithm, based on records in the indexes corresponding to the foreign key constraints.

- A FOREIGN KEY constraint that references a non-UNIQUE key is not standard SQL but rather an InnoDB extension.
- If ON UPDATE CASCADE or ON UPDATE SET NULL recurses to update the *same table* it has previously updated during the same cascade, it acts like RESTRICT. This means that you cannot use self-referential ON UPDATE CASCADE or ON UPDATE SET NULL operations. This is to prevent infinite loops resulting from cascaded updates. A self-referential ON DELETE SET NULL, on the other hand, is possible, as is a self-referential ON DELETE CASCADE. Cascading operations may not be nested more than 15 levels deep.
- In an SQL statement that inserts, deletes, or updates many rows, foreign key constraints (like unique constraints) are checked row-by-row. When performing foreign key checks, InnoDB sets shared row-level locks on child or parent records that it must examine. MySQL checks foreign key constraints immediately; the check is not deferred to transaction commit. According to the SQL standard, the default behavior should be deferred checking. That is, constraints are only checked after the *entire SQL statement* has been processed. This means that it is not possible to delete a row that refers to itself using a foreign key.

For information how InnoDB foreign keys differ from the SQL standard, see Section 14.2.6.6, “InnoDB and FOREIGN KEY Constraints”.

1.8.3.3 Constraints on Invalid Data

By default, MySQL is forgiving of invalid or improper data values and coerces them to valid values for data entry. However, you can change the server SQL mode to select more traditional treatment of bad values such that the server rejects them and aborts the statement in which they occur. See Section 5.1.7, “Server SQL Modes”.

This section describes the default (forgiving) behavior of MySQL, as well as the strict SQL mode and how it differs.

If you are not using strict mode, then whenever you insert an “incorrect” value into a column, such as a NULL into a NOT NULL column or a too-large numeric value into a numeric column, MySQL sets the column to the “best possible value” instead of producing an error: The following rules describe in more detail how this works:

- If you try to store an out of range value into a numeric column, MySQL Server instead stores zero, the smallest possible value, or the largest possible value, whichever is closest to the invalid value.
- For strings, MySQL stores either the empty string or as much of the string as can be stored in the column.
- If you try to store a string that doesn't start with a number into a numeric column, MySQL Server stores 0.
- Invalid values for ENUM and SET columns are handled as described in Section 1.8.3.4, “ENUM and SET Constraints”.
- MySQL enables you to store certain incorrect date values into DATE and DATETIME columns (such as '2000-02-31' or '2000-02-00'). The idea is that it is not the job of the SQL server to validate dates. If MySQL can store a date value and retrieve exactly the same value, MySQL stores it as given. If the date is totally wrong (outside the server's ability to store it), the special “zero” date value '0000-00-00' is stored in the column instead.
- If you try to store NULL into a column that doesn't take NULL values, an error occurs for single-row INSERT statements. For multiple-row INSERT statements or for INSERT INTO ... SELECT statements, MySQL Server stores the implicit default value for the column data type. In general, this is

0 for numeric types, the empty string ('') for string types, and the “zero” value for date and time types. Implicit default values are discussed in Section 11.5, “Data Type Default Values”.

- If an INSERT statement specifies no value for a column, MySQL inserts its default value if the column definition includes an explicit DEFAULT clause. If the definition has no such DEFAULT clause, MySQL inserts the implicit default value for the column data type.

The reason for using the preceding rules in nonstrict mode is that we can't check these conditions until the statement has begun executing. We can't just roll back if we encounter a problem after updating a few rows, because the storage engine may not support rollback. The option of terminating the statement is not that good; in this case, the update would be “half done,” which is probably the worst possible scenario. In this case, it is better to “do the best you can” and then continue as if nothing happened.

In MySQL 5.0.2 and up, you can select stricter treatment of input values by using the STRICT_TRANS_TABLES or STRICT_ALL_TABLES SQL modes:

```
SET sql_mode = 'STRICT_TRANS_TABLES';
SET sql_mode = 'STRICT_ALL_TABLES';
```

STRICT_TRANS_TABLES enables strict mode for transactional storage engines, and also to some extent for nontransactional engines. It works like this:

- For transactional storage engines, bad data values occurring anywhere in a statement cause the statement to abort and roll back.

- For nontransactional storage engines, a statement aborts if the error occurs in the first row to be inserted or updated. (When the error occurs in the first row, the statement can be aborted to leave the table unchanged, just as for a transactional table.) Errors in rows after the first do not abort the statement, because the table has already been changed by the first row. Instead, bad data values are adjusted and result in warnings rather than errors. In other words, with STRICT_TRANS_TABLES, a wrong value causes MySQL to roll back all updates done so far, if that can be done without changing the table. But once the table has been changed, further errors result in adjustments and warnings.

For even stricter checking, enable STRICT_ALL_TABLES. This is the same as STRICT_TRANS_TABLES except that for nontransactional storage engines, errors abort the statement even for bad data in rows following the first row. This means that if an error occurs partway through a multiple-row insert or update for a nontransactional table, a partial update results. Earlier rows are inserted or updated, but those from the point of the error on are not. To avoid this for nontransactional tables, either use single-row statements or else use STRICT_TRANS_TABLES if conversion warnings rather than errors are acceptable. To avoid problems in the first place, do not use MySQL to check column content. It is safest (and often faster) to let the application ensure that it passes only valid values to the database.

With either of the strict mode options, you can cause errors to be treated as warnings by using INSERT IGNORE or UPDATE IGNORE rather than INSERT or UPDATE without IGNORE.

1.8.3.4 ENUM and SET Constraints

ENUM and SET columns provide an efficient way to define columns that can contain only a given set of values. See Section 11.4.4, “The ENUM Type”, and Section 11.4.5, “The SET Type”. However, before MySQL 5.0.2, ENUM and SET columns do not provide true constraints on entry of invalid data:

- ENUM columns always have a default value. If you specify no default value, then it is NULL for columns that can have NULL, otherwise it is the first enumeration value in the column definition.

- If you insert an incorrect value into an ENUM column or if you force a value into an ENUM column with IGNORE, it is set to the reserved enumeration value of 0, which is displayed as an empty string in string context.

- If you insert an incorrect value into a SET column, the incorrect value is ignored. For example, if the column can contain the values 'a', 'b', and 'c', an attempt to assign 'a,x,b,y' results in a value of 'a,b'.

As of MySQL 5.0.2, you can configure the server to use strict SQL mode. See Section 5.1.7, “Server SQL Modes”. With strict mode enabled, the definition of a ENUM or SET column does act as a constraint on values entered into the column. An error occurs for values that do not satisfy these conditions:

- An ENUM value must be one of those listed in the column definition, or the internal numeric equivalent thereof. The value cannot be the error value (that is, 0 or the empty string). For a column defined as ENUM('a','b','c'), values such as '', 'd', or 'ax' are invalid and are rejected.
- A SET value must be the empty string or a value consisting only of the values listed in the column definition separated by commas. For a column defined as SET('a','b','c'), values such as 'd' or 'a,b,c,d' are invalid and are rejected.

Errors for invalid values can be suppressed in strict mode if you use INSERT IGNORE or UPDATE IGNORE. In this case, a warning is generated rather than an error. For ENUM, the value is inserted as the error member (0). For SET, the value is inserted as given except that any invalid substrings are deleted. For example, 'a,x,b,y' results in a value of 'a,b'.

1.9 Credits

The following sections list developers, contributors, and supporters that have helped to make MySQL what it is today.

1.9.1 Contributors to MySQL

Although Oracle Corporation and/or its affiliates own all copyrights in the MySQL server and the MySQL manual, we wish to recognize those who have made contributions of one kind or another to the MySQL distribution. Contributors are listed here, in somewhat random order:

- Gianmassimo Vigazzola <qwerg@mbox.vol.it> or <qwerg@tin.it>

  The initial port to Win32/NT.

- Per Eric Olsson

  For constructive criticism and real testing of the dynamic record format.

- Irena Pancirov <irena@mail.yacc.it>

  Win32 port with Borland compiler. mysqlshutdown.exe and mysqlwatch.exe.

- David J. Hughes

  For the effort to make a shareware SQL database. At TcX, the predecessor of MySQL AB, we started with mSQL, but found that it couldn't satisfy our purposes so instead we wrote an SQL interface to our application builder Unireg. mysqladmin and mysql client are programs that were largely influenced by their mSQL counterparts. We have put a lot of effort into making the MySQL syntax a superset of mSQL. Many of the API's ideas are borrowed from mSQL to make it easy to port free mSQL programs to the MySQL API. The MySQL software doesn't contain any code from mSQL. Two files in the distribution (client/insert_test.c and client/select_test.c) are based on the corresponding (noncopyrighted) files in the mSQL distribution, but are modified as examples showing the changes necessary to convert code from mSQL to MySQL Server. (mSQL is copyrighted David J. Hughes.)

- Patrick Lynch

For helping us acquire http://www.mysql.com/.

- Fred Lindberg

  For setting up qmail to handle the MySQL mailing list and for the incredible help we got in managing the MySQL mailing lists.

- Igor Romanenko `<igor@frog.kiev.ua>`

  `mysqldump` (previously `msqldump`, but ported and enhanced by Monty).

- Yuri Dario

  For keeping up and extending the MySQL OS/2 port.

- Tim Bunce

  Author of `mysqlhotcopy`.

- Zarko Mocnik `<zarko.mocnik@dem.si>`

  Sorting for Slovenian language.

- "TAMITO" `<tommy@valley.ne.jp>`

  The `_MB` character set macros and the ujis and sjis character sets.

- Joshua Chamas `<joshua@chamas.com>`

  Base for concurrent insert, extended date syntax, debugging on NT, and answering on the MySQL mailing list.

- Yves Carlier `<Yves.Carlier@rug.ac.be>`

  `mysqlaccess`, a program to show the access rights for a user.

- Rhys Jones `<rhys@wales.com>` (And GWE Technologies Limited)

  For one of the early JDBC drivers.

- Dr Xiaokun Kelvin ZHU `<X.Zhu@brad.ac.uk>`

  Further development of one of the early JDBC drivers and other MySQL-related Java tools.

- James Cooper `<pixel@organic.com>`

  For setting up a searchable mailing list archive at his site.

- Rick Mehalick `<Rick_Mehalick@i-o.com>`

  For `xmysql`, a graphical X client for MySQL Server.

- Doug Sisk `<sisk@wix.com>`

  For providing RPM packages of MySQL for Red Hat Linux.

- Diemand Alexander V. `<axeld@vial.ethz.ch>`

  For providing RPM packages of MySQL for Red Hat Linux-Alpha.

- Antoni Pamies Olive <toni@readysoft.es>

  For providing RPM versions of a lot of MySQL clients for Intel and SPARC.

- Jay Bloodworth <jay@pathways.sde.state.sc.us>

  For providing RPM versions for MySQL 3.21.

- David Sacerdote <davids@secnet.com>

  Ideas for secure checking of DNS host names.

- Wei-Jou Chen <jou@nematic.ieo.nctu.edu.tw>

  Some support for Chinese(BIG5) characters.

- Wei He <hewei@mail.ied.ac.cn>

  A lot of functionality for the Chinese(GBK) character set.

- Jan Pazdziora <adelton@fi.muni.cz>

  Czech sorting order.

- Zeev Suraski <bourbon@netvision.net.il>

  FROM_UNIXTIME() time formatting, ENCRYPT() functions, and bison advisor. Active mailing list member.

- Luuk de Boer <luuk@wxs.nl>

  Ported (and extended) the benchmark suite to DBI/DBD. Have been of great help with crash-me and running benchmarks. Some new date functions. The mysql_setpermission script.

- Alexis Mikhailov <root@medinf.chuvashia.su>

  User-defined functions (UDFs); CREATE FUNCTION and DROP FUNCTION.

- Andreas F. Bobak <bobak@relog.ch>

  The AGGREGATE extension to user-defined functions.

- Ross Wakelin <R.Wakelin@march.co.uk>

  Help to set up InstallShield for MySQL-Win32.

- Jethro Wright III <jetman@li.net>

  The libmysql.dll library.

- James Pereria <jpereira@iafrica.com>

  Mysqlmanager, a Win32 GUI tool for administering MySQL Servers.

- Curt Sampson <cjs@portal.ca>

  Porting of MIT-pthreads to NetBSD/Alpha and NetBSD 1.3/i386.

- Martin Ramsch <m.ramsch@computer.org>

Examples in the MySQL Tutorial.

- Steve Harvey

  For making `mysqlaccess` more secure.

- Konark IA-64 Centre of Persistent Systems Private Limited

  http://www.pspl.co.in/konark/. Help with the Win64 port of the MySQL server.

- Albert Chin-A-Young.

  Configure updates for Tru64, large file support and better TCP wrappers support.

- John Birrell

  Emulation of `pthread_mutex()` for OS/2.

- Benjamin Pflugmann

  Extended `MERGE` tables to handle `INSERTS`. Active member on the MySQL mailing lists.

- Jocelyn Fournier

  Excellent spotting and reporting innumerable bugs (especially in the MySQL 4.1 subquery code).

- Marc Liyanage

  Maintaining the Mac OS X packages and providing invaluable feedback on how to create Mac OS X packages.

- Robert Rutherford

  Providing invaluable information and feedback about the QNX port.

- Previous developers of NDB Cluster

  Lots of people were involved in various ways summer students, master thesis students, employees. In total more than 100 people so too many to mention here. Notable name is Ataullah Dabaghi who up until 1999 contributed around a third of the code base. A special thanks also to developers of the AXE system which provided much of the architectural foundations for NDB Cluster with blocks, signals and crash tracing functionality. Also credit should be given to those who believed in the ideas enough to allocate of their budgets for its development from 1992 to present time.

- Google Inc.

  We wish to recognize Google Inc. for contributions to the MySQL distribution: Mark Callaghan's SMP Performance patches and other patches.

Other contributors, bugfinders, and testers: James H. Thompson, Maurizio Menghini, Wojciech Tryc, Luca Berra, Zarko Mocnik, Wim Bonis, Elmar Haneke, `<jehamby@lightside>`, `<psmith@BayNetworks.com>`, `<duane@connect.com.au>`, Ted Deppner `<ted@psyber.com>`, Mike Simons, Jaakko Hyvatti.

And lots of bug report/patches from the folks on the mailing list.

A big tribute goes to those that help us answer questions on the MySQL mailing lists:

- Daniel Koch <dkoch@amcity.com>

  Irix setup.

- Luuk de Boer <luuk@wxs.nl>

  Benchmark questions.

- Tim Sailer <tps@users.buoy.com>

  DBD::mysql questions.

- Boyd Lynn Gerber <gerberb@zenez.com>

  SCO-related questions.

- Richard Mehalick <RM186061@shellus.com>

  xmysql-related questions and basic installation questions.

- Zeev Suraski <bourbon@netvision.net.il>

  Apache module configuration questions (log & auth), PHP-related questions, SQL syntax-related questions and other general questions.

- Francesc Guasch <frankie@citel.upc.es>

  General questions.

- Jonathan J Smith <jsmith@wtp.net>

  Questions pertaining to OS-specifics with Linux, SQL syntax, and other things that might need some work.

- David Sklar <sklar@student.net>

  Using MySQL from PHP and Perl.

- Alistair MacDonald <A.MacDonald@uel.ac.uk>

  Is flexible and can handle Linux and perhaps HP-UX.

- John Lyon <jlyon@imag.net>

  Questions about installing MySQL on Linux systems, using either .rpm files or compiling from source.

- Lorvid Ltd. <lorvid@WOLFENET.com>

  Simple billing/license/support/copyright issues.

- Patrick Sherrill <patrick@coconet.com>

  ODBC and VisualC++ interface questions.

- Randy Harmon <rjharmon@uptimecomputers.com>

  DBD, Linux, some SQL syntax questions.

1.9.2 Documenters and translators

The following people have helped us with writing the MySQL documentation and translating the documentation or error messages in MySQL.

- Paul DuBois

  Ongoing help with making this manual correct and understandable. That includes rewriting Monty's and David's attempts at English into English as other people know it.

- Kim Aldale

  Helped to rewrite Monty's and David's early attempts at English into English.

- Michael J. Miller Jr. <mke@terrapin.turbolift.com>

  For the first MySQL manual. And a lot of spelling/language fixes for the FAQ (that turned into the MySQL manual a long time ago).

- Yan Cailin

  First translator of the MySQL Reference Manual into simplified Chinese in early 2000 on which the Big5 and HK coded (http://mysql.hitstar.com/) versions were based. Personal home page at linuxdb.yeah.net.

- Jay Flaherty <fty@mediapulse.com>

  Big parts of the Perl DBI/DBD section in the manual.

- Paul Southworth <pauls@etext.org>, Ray Loyzaga <yar@cs.su.oz.au>

  Proof-reading of the Reference Manual.

- Therrien Gilbert <gilbert@ican.net>, Jean-Marc Pouyot <jmp@scalaire.fr>

  French error messages.

- Petr Snajdr, <snajdr@pvt.net>

  Czech error messages.

- Jaroslaw Lewandowski <jotel@itnet.com.pl>

  Polish error messages.

- Miguel Angel Fernandez Roiz

  Spanish error messages.

- Roy-Magne Mo <rmo@www.hivolda.no>

  Norwegian error messages and testing of MySQL 3.21.xx.

- Timur I. Bakeyev <root@timur.tatarstan.ru>

  Russian error messages.

- <brenno@dewinter.com> & Filippo Grassilli <phil@hyppo.com>

  Italian error messages.

- Dirk Munzinger <dirk@trinity.saar.de>

German error messages.

- Billik Stefan <billik@sun.uniag.sk>

  Slovak error messages.

- Stefan Saroiu <tzoompy@cs.washington.edu>

  Romanian error messages.

- Peter Feher

  Hungarian error messages.

- Roberto M. Serqueira

  Portuguese error messages.

- Carsten H. Pedersen

  Danish error messages.

- Arjen Lentz

  Dutch error messages, completing earlier partial translation (also work on consistency and spelling).

1.9.3 Packages that support MySQL

The following is a list of creators/maintainers of some of the most important API/packages/applications that a lot of people use with MySQL.

We cannot list every possible package here because the list would then be way to hard to maintain. For other packages, please refer to the software portal at http://solutions.mysql.com/software/.

- Tim Bunce, Alligator Descartes

  For the DBD (Perl) interface.

- Andreas Koenig <a.koenig@mind.de>

  For the Perl interface for MySQL Server.

- Jochen Wiedmann <wiedmann@neckar-alb.de>

  For maintaining the Perl DBD::mysql module.

- Eugene Chan <eugene@acenet.com.sg>

  For porting PHP for MySQL Server.

- Georg Richter

  MySQL 4.1 testing and bug hunting. New PHP 5.0 mysqli extension (API) for use with MySQL 4.1 and up.

- Giovanni Maruzzelli <maruzz@matrice.it>

  For porting iODBC (Unix ODBC).

- Xavier Leroy `<Xavier.Leroy@inria.fr>`

  The author of LinuxThreads (used by the MySQL Server on Linux).

1.9.4 Tools that were used to create MySQL

The following is a list of some of the tools we have used to create MySQL. We use this to express our thanks to those that has created them as without these we could not have made MySQL what it is today.

- Free Software Foundation

  From whom we got an excellent compiler (`gcc`), an excellent debugger (`gdb` and the `libc` library (from which we have borrowed `strto.c` to get some code working in Linux).

- Free Software Foundation & The XEmacs development team

  For a really great editor/environment.

- Julian Seward

  Author of `valgrind`, an excellent memory checker tool that has helped us find a lot of otherwise hard to find bugs in MySQL.

- Dorothea Lütkehaus and Andreas Zeller

  For `DDD` (The Data Display Debugger) which is an excellent graphical front end to `gdb`).

1.9.5 Supporters of MySQL

Although Oracle Corporation and/or its affiliates own all copyrights in the `MySQL server` and the `MySQL manual`, we wish to recognize the following companies, which helped us finance the development of the `MySQL server`, such as by paying us for developing a new feature or giving us hardware for development of the `MySQL server`.

- VA Linux / Andover.net

  Funded replication.

- NuSphere

  Editing of the MySQL manual.

- Stork Design studio

  The MySQL Web site in use between 1998-2000.

- Intel

  Contributed to development on Windows and Linux platforms.

- Compaq

  Contributed to Development on Linux/Alpha.

- SWSoft

  Development on the embedded `mysqld` version.

- FutureQuest

  The `--skip-show-database` option.

Chapter 2 Installing and Upgrading MySQL

Table of Contents

This chapter describes how to obtain and install MySQL. A summary of the procedure follows and later sections provide the details. If you plan to upgrade an existing version of MySQL to a newer version rather than install MySQL for the first time, see Section 2.10.1, “Upgrading MySQL”, for information about upgrade procedures and about issues that you should consider before upgrading.

If you are interested in migrating to MySQL from another database system, you may wish to read Section B.8, “MySQL 5.7 FAQ: Migration”, which contains answers to some common questions concerning migration issues.

Installation of MySQL generally follows the steps outlined here:

1. **Determine whether MySQL runs and is supported on your platform.**

   Please note that not all platforms are equally suitable for running MySQL, and that not all platforms on which MySQL is known to run are officially supported by Oracle Corporation:

2. **Choose which distribution to install.**

   Several versions of MySQL are available, and most are available in several distribution formats. You can choose from pre-packaged distributions containing binary (precompiled) programs or source code. When in doubt, use a binary distribution. We also provide public access to our current source tree for those who want to see our most recent developments and help us test new code. To determine which version and type of distribution you should use, see Section 2.1.2, “Choosing Which MySQL Distribution to Install”.

3. **Download the distribution that you want to install.**

   For instructions, see Section 2.1.3, “How to Get MySQL”. To verify the integrity of the distribution, use the instructions in Section 2.1.4, “Verifying Package Integrity Using MD5 Checksums or GnuPG”.

4. **Install the distribution.**

   To install MySQL from a binary distribution, use the instructions in Section 2.2, “Installing MySQL on Unix/Linux Using Generic Binaries”.

   To install MySQL from a source distribution or from the current development source tree, use the instructions in Section 2.8, “Installing MySQL from Source”.

5. **Perform any necessary postinstallation setup.**

   After installing MySQL, see Section 2.9, “Postinstallation Setup and Testing” for information about making sure the MySQL server is working properly. Also refer to the information provided in Section 2.9.2, “Securing the Initial MySQL Accounts”. This section describes how to secure the initial MySQL user accounts, *which have no passwords* until you assign passwords. The section applies whether you install MySQL using a binary or source distribution.

6. If you want to run the MySQL benchmark scripts, Perl support for MySQL must be available. See Section 2.12, “Perl Installation Notes”.

Instructions for installing MySQL on different platforms and environments is available on a platform by platform basis:

- **Unix, Linux, FreeBSD**

  For instructions on installing MySQL on most Linux and Unix platforms using a generic binary (for example, a `.tar.gz` package), see Section 2.2, "Installing MySQL on Unix/Linux Using Generic Binaries".

  For information on building MySQL entirely from the source code distributions or the source code repositories, see Section 2.8, "Installing MySQL from Source"

  For specific platform help on installation, configuration, and building from source see the corresponding platform section:

  - Linux, including notes on distribution specific methods, see Section 2.5, "Installing MySQL on Linux".

  - Solaris and OpenSolaris, including PKG and IPS formats, see Section 2.6, "Installing MySQL on Solaris and OpenSolaris".

  - IBM AIX, see Section 2.6, "Installing MySQL on Solaris and OpenSolaris".

  - FreeBSD, see Section 2.7, "Installing MySQL on FreeBSD".

- **Microsoft Windows**

  For instructions on installing MySQL on Microsoft Windows, using either a Zipped binary or an MSI package, see Section 2.3, "Installing MySQL on Microsoft Windows".

  For information on using the MySQL Server Instance Config Wizard, see MySQL Server Instance Configuration Wizard.

  For details and instructions on building MySQL from source code using Microsoft Visual Studio, see Section 2.8, "Installing MySQL from Source".

- **Mac OS X**

  For installation on Mac OS X, including using both the binary package and native PKG formats, see Section 2.4, "Installing MySQL on Mac OS X".

  For information on making use of the MySQL Startup Item to automatically start and stop MySQL, see Section 2.4.3, "Installing the MySQL Startup Item".

  For information on the MySQL Preference Pane, see Section 2.4.4, "Installing and Using the MySQL Preference Pane".

2.1 General Installation Guidance

The immediately following sections contain the information necessary to choose, download, and verify your distribution. The instructions in later sections of the chapter describe how to install the distribution that you choose. For binary distributions, see the instructions at Section 2.2, "Installing MySQL on Unix/Linux Using Generic Binaries" or the corresponding section for your platform if available. To build MySQL from source, use the instructions in Section 2.8, "Installing MySQL from Source".

2.1.1 Operating Systems Supported by MySQL Community Server

MySQL is available on many operating systems and platforms. For information about platforms supported by GA releases of MySQL, see http://www.mysql.com/support/supportedplatforms/database.html. For development versions of MySQL, builds are available for a number of platforms at http://dev.mysql.com/downloads/mysql/5.7.html. To learn more about MySQL Support, see http://www.mysql.com/support/.

2.1.2 Choosing Which MySQL Distribution to Install

When preparing to install MySQL, you should decide which version to use. MySQL development occurs in several release series, and you can pick the one that best fits your needs. After deciding which version to install, you can choose a distribution format. Releases are available in binary or source format.

2.1.2.1 Choosing Which Version of MySQL to Install

The first decision to make is whether you want to use a production (stable) release or a development release. In the MySQL development process, multiple release series co-exist, each at a different stage of maturity.

Production Releases

- MySQL 5.6: Latest General Availability (Production) release
- MySQL 5.5: Previous General Availability (Production) release
- MySQL 5.1: Older General Availability (Production) release
- MySQL 5.0: Older Production release nearing the end of the product lifecycle

MySQL 4.1, 4.0, and 3.23 are old releases that are no longer supported.

See http://www.mysql.com/about/legal/lifecycle/ for information about support policies and schedules.

Normally, if you are beginning to use MySQL for the first time or trying to port it to some system for which there is no binary distribution, use the most recent General Availability series listed in the preceding descriptions. All MySQL releases, even those from development series, are checked with the MySQL benchmarks and an extensive test suite before being issued.

If you are running an older system and want to upgrade, but do not want to take the chance of having a nonseamless upgrade, you should upgrade to the latest version in the same release series you are using (where only the last part of the version number is newer than yours). We have tried to fix only fatal bugs and make only small, relatively “safe” changes to that version.

If you want to use new features not present in the production release series, you can use a version from a development series. Be aware that development releases are not as stable as production releases.

We do not use a complete code freeze because this prevents us from making bugfixes and other fixes that must be done. We may add small things that should not affect anything that currently works in a production release. Naturally, relevant bugfixes from an earlier series propagate to later series.

If you want to use the very latest sources containing all current patches and bugfixes, you can use one of our source code repositories (see Section 2.8.3, “Installing MySQL Using a Development Source Tree”). These are not “releases” as such, but are available as previews of the code on which future releases are to be based.

The naming scheme in MySQL 5.7 uses release names that consist of three numbers and a suffix; for example, **mysql-5.7.1-m1**. The numbers within the release name are interpreted as follows:

- The first number (**5**) is the major version and describes the file format. All MySQL 5 releases have the same file format.

- The second number (**7**) is the release level. Taken together, the major version and release level constitute the release series number.
- The third number (**1**) is the version number within the release series. This is incremented for each new release. Usually you want the latest version for the series you have chosen.

For each minor update, the last number in the version string is incremented. When there are major new features or minor incompatibilities with previous versions, the second number in the version string is incremented. When the file format changes, the first number is increased.

Release names also include a suffix to indicates the stability level of the release. Releases within a series progress through a set of suffixes to indicate how the stability level improves. The possible suffixes are:

- **mN** (for example, **m1**, **m2**, **m3**, ...) indicate a milestone number. MySQL development uses a milestone model, in which each milestone proceeds through a small number of versions with a tight focus on a small subset of thoroughly tested features. Following the releases for one milestone, development proceeds with another small number of releases that focuses on the next small set of features, also thoroughly tested. Features within milestone releases may be considered to be of pre-production quality.
- **rc** indicates a Release Candidate. Release candidates are believed to be stable, having passed all of MySQL's internal testing, and with all known fatal runtime bugs fixed. However, the release has not been in widespread use long enough to know for sure that all bugs have been identified. Only minor fixes are added.
- If there is no suffix, it indicates that the release is a General Availability (GA) or Production release. GA releases are stable, having successfully passed through all earlier release stages and are believed to be reliable, free of serious bugs, and suitable for use in production systems. Only critical bugfixes are applied to the release.

All releases of MySQL are run through our standard tests and benchmarks to ensure that they are relatively safe to use. Because the standard tests are extended over time to check for all previously found bugs, the test suite keeps getting better.

All releases have been tested at least with these tools:

- **An internal test suite.** The `mysql-test` directory contains an extensive set of test cases. We run these tests for every server binary. See Section 22.1.2, “The MySQL Test Suite”, for more information about this test suite.
- **The MySQL benchmark suite.** This suite runs a range of common queries. It is also a test to determine whether the latest batch of optimizations actually made the code faster. See Section 8.12.2, “The MySQL Benchmark Suite”.

We also perform additional integration and nonfunctional testing of the latest MySQL version in our internal production environment. Integration testing is done with different connectors, storage engines, replication modes, backup, partitioning, stored programs, and so forth in various combinations. Additional nonfunctional testing is done in areas of performance, concurrency, stress, high volume, upgrade and downgrade.

2.1.2.2 Choosing a Distribution Format

After choosing which version of MySQL to install, you should decide whether to use a binary distribution or a source distribution. In most cases, you should probably use a binary distribution, if one exists for your platform. Binary distributions are available in native format for many platforms, such as RPM packages for Linux, DMG packages for Mac OS X, and PKG packages for Solaris. Distributions are also available in more generic formats such as Zip archives or compressed `tar` files.

Reasons to choose a binary distribution include the following:

- Binary distributions generally are easier to install than source distributions.
- To satisfy different user requirements, we provide several servers in binary distributions. `mysqld` is an optimized server that is a smaller, faster binary. `mysqld-debug` is compiled with debugging support.

  Each of these servers is compiled from the same source distribution, though with different configuration options. All native MySQL clients can connect to servers from either MySQL version.

Under some circumstances, you may be better off installing MySQL from a source distribution:

- You want to install MySQL at some explicit location. The standard binary distributions are ready to run at any installation location, but you might require even more flexibility to place MySQL components where you want.
- You want to configure `mysqld` to ensure that features are available that might not be included in the standard binary distributions. Here is a list of the most common extra options that you may want to use to ensure feature availability:
  - `-DWITH_LIBWRAP=1` for TCP wrappers support.
  - `-DWITH_ZLIB={system|bundled}` for features that depend on compression
  - `-DWITH_DEBUG=1` for debugging support

  For additional information, see Section 2.8.4, “MySQL Source-Configuration Options”.
- You want to configure `mysqld` without some features that are included in the standard binary distributions. For example, distributions normally are compiled with support for all character sets. If you want a smaller MySQL server, you can recompile it with support for only the character sets you need.
- You want to use the latest sources from one of the Bazaar repositories to have access to all current bugfixes. For example, if you have found a bug and reported it to the MySQL development team, the bugfix is committed to the source repository and you can access it there. The bugfix does not appear in a release until a release actually is issued.
- You want to read (or modify) the C and C++ code that makes up MySQL. For this purpose, you should get a source distribution, because the source code is always the ultimate manual.
- Source distributions contain more tests and examples than binary distributions.

2.1.2.3 How and When Updates Are Released

MySQL is evolving quite rapidly and we want to share new developments with other MySQL users. We try to produce a new release whenever we have new and useful features that others also seem to have a need for.

We also try to help users who request features that are easy to implement. We take note of what our licensed users want, and we especially take note of what our support customers want and try to help them in this regard.

No one is *required* to download a new release. The Release Notes help you determine whether the new release has something you really want.

We use the following policy when updating MySQL:

- Enterprise Server releases are meant to appear every 18 months, supplemented by quarterly service packs and monthly rapid updates. Community Server releases are meant to appear 2 to 3 times per year.

- Releases are issued within each series. For each release, the last number in the version is one more than the previous release within the same series.
- Binary distributions for some platforms are made by us for major releases. Other people may make binary distributions for other systems, but probably less frequently.
- We make fixes available as soon as we have identified and corrected small or noncritical but annoying bugs. The fixes are available in source form immediately from our public Bazaar repositories, and are included in the next release.
- If by any chance a security vulnerability or critical bug is found in a release, our policy is to fix it in a new release as soon as possible. (We would like other companies to do this, too!)

2.1.3 How to Get MySQL

Check our downloads page at http://dev.mysql.com/downloads/ for information about the current version of MySQL and for downloading instructions. For a complete up-to-date list of MySQL download mirror sites, see http://dev.mysql.com/downloads/mirrors.html. You can also find information there about becoming a MySQL mirror site and how to report a bad or out-of-date mirror.

For RPM-based Linux platforms that use Yum as their package management system, MySQL can be installed using the MySQL Yum repository. See Section 2.5.1, “Installing MySQL on Linux Using the MySQL Yum Repository” for details.

To obtain the latest development source, see Section 2.8.3, “Installing MySQL Using a Development Source Tree”.

2.1.4 Verifying Package Integrity Using MD5 Checksums or `GnuPG`

After you have downloaded the MySQL package that suits your needs and before you attempt to install it, you should make sure that it is intact and has not been tampered with. There are three means of integrity checking:

- MD5 checksums
- Cryptographic signatures using `GnuPG`, the GNU Privacy Guard
- For RPM packages, the built-in RPM integrity verification mechanism

The following sections describe how to use these methods.

If you notice that the MD5 checksum or GPG signatures do not match, first try to download the respective package one more time, perhaps from another mirror site.

2.1.4.1 Verifying the MD5 Checksum

After you have downloaded a MySQL package, you should make sure that its MD5 checksum matches the one provided on the MySQL download pages. Each package has an individual checksum that you can verify against the package that you downloaded. The correct MD5 checksum is listed on the downloads page for each MySQL product, and you will compare it against the MD5 checksum of the file (product) that you download.

Each operating system and setup offers its own version of tools for checking the MD5 checksum. Typically the command is named `md5sum`, or it may be named `md5`, and some operating systems do not ship it at all. On Linux, it is part of the **GNU Text Utilities** package, which is available for a wide range of platforms. You can also download the source code from http://www.gnu.org/software/textutils/. If you have OpenSSL installed, you can use the command `openssl md5 *package_name*` instead. A Windows implementation of the `md5` command line utility is available from http://www.fourmilab.ch/md5/. `winMd5Sum` is a graphical

MD5 checking tool that can be obtained from http://www.nullriver.com/index/products/winmd5sum. Our Microsoft Windows examples will assume the name `md5.exe`.

Linux and Microsoft Windows examples:

```
shell> md5sum mysql-standard-5.7.5-linux-i686.tar.gz
aaab65abbec64d5e907dcd41b8699945  mysql-standard-5.7.5-linux-i686.tar.gz
```

```
shell> md5.exe mysql-installer-community-5.7.5.msi
aaab65abbec64d5e907dcd41b8699945  mysql-installer-community-5.7.5.msi
```

You should verify that the resulting checksum (the string of hexadecimal digits) matches the one displayed on the download page immediately below the respective package.

Note

Make sure to verify the checksum of the *archive file* (for example, the `.zip`, `.tar.gz`, or `.msi` file) and not of the files that are contained inside of the archive. In other words, verify the file before extracting its contents.

2.1.4.2 Signature Checking Using GnuPG

Another method of verifying the integrity and authenticity of a package is to use cryptographic signatures. This is more reliable than using MD5 checksums, but requires more work.

We sign MySQL downloadable packages with `GnuPG` (GNU Privacy Guard). `GnuPG` is an Open Source alternative to the well-known Pretty Good Privacy (`PGP`) by Phil Zimmermann. See http://www.gnupg.org/ for more information about `GnuPG` and how to obtain and install it on your system. Most Linux distributions ship with `GnuPG` installed by default. For more information about `GnuPG`, see http://www.openpgp.org/.

To verify the signature for a specific package, you first need to obtain a copy of our public GPG build key, which you can download from http://pgp.mit.edu/. The key that you want to obtain is named `mysql-build@oss.oracle.com`. Alternatively, you can cut and paste the key directly from the following text:

```
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v1.4.9 (SunOS)

mQGiBD4+owwRBAC14GIfUfCyEDSIePvEW3SAFUdJBtoQHH/nJKZyQT7h9bPlUWC3
RODjQReyCITRrdwyrKUGku2FmeVGwn2u2WmDMNABLnpprWPkBdCk96+OmSLN9brZ
fw2vOUgCmYv2hW0hyDHuvYlQA/BThQoADgj8AW6/0Lo7V1W9/8VuHP0gQwCgvzV3
BqOxRznNCRCRxAuAuVztHRcEAJooQK1+iSiunZMYD1WufeXfshc57S/+yeJkegNW
hxwR9pRWVArNYJdDRT+rf2RUe3vpquKNQU/hnEIUHJRQqYHo8gTxvxXNQc7fJYLV
K2HtkrPbP72vwsEKMYhhr0eKCbtLGfls9krjJ6sBgACyP/Vb7hiPwxh6rDZ7ITnE
kYpXBACmWpP8NJTkamEnPCia2ZoOHODANwpUkP43I7jsDmgtobZX9qnrAXw+uNDI
QJEXM6FSbi0LLtZciNlYsafwAPEOMDKpMqAK6IyisNtPvaLd8lH0bPAnWqcyefep
rv0sxxqUEMcM3o7wwgfN83POkDasDbs3pjwPhxvhz6//62zQJ7Q2TXlTUUwgUmVs
ZWFzZSBFbmdpbmVlcmluZyA8bXlzcWwtYnVpbGRAb3NzLm9yYWNsZS5jb20+iGkE
ExECACkCGyMGCwkIBwMCBBUCCAMEFgIDAQIeAQIXgAIZAQUCUwHUZgUJGmbLywAK
CRCMcY07UHLh9V+DAKCjS1gGwgVI/eut+5L+l2v3ybl+ZgCcD7ZoA341HtoroV3U
6xRD09fUgeq0O015U1FMIFBhY2thZ2Ugc2lnbmluZyBrZXkgKHd3dy5teXNxbC5j
b20pIDxidWlsZEBteXNxbC5jb20+iG8EMBECAC8FAk53Pa0oHSBidWlsZEBteXNx
bC5jb20gd2lsbCBzdG9wIHdvcmtpbmcgc29vbgAKCRCMcY07UHLh9bU9AJ9xDK0o
xJFL9vTl9OSZC4lX0K9AzwCcCrS9cnJyz79eaRjL0s2r/CcljdyIZQQTEQIAHQUC
R6yUtAUJDTBYqAULBwoDBAMVAwIDFgIBAheAABIJEIxxjTtQcuH1B2VHUEcAAQGu
kgCffz4GUEjzXkOi71VcwgCxASTgbe0An34LPr1j9fCbrXWXO14msIADfb5piEwE
ExECAAwFAj4+o9EFgwlmALsACgkQSVDhKrJykfIk4QCfWbEeKN+3TRspe+5xKj+k
QJSammIAnjUz0xFWPlVx0f8o38qNG1bq0cU9iEwEExECAAwFAj5CggMFgwliIokA
CgkQtvXNTca6JD+WkQCgiGmnoGjMojynp5ppvMXkyUkfnykAoK79E6h8rwkSDZou
iz7nMRisH8uyiEYEEBECAAYFAj+s468ACgkQr8UjSHiDdA/2lgCg21IhIMMABTYd
p/IBiUsP/JQLiEoAnRzMywEtujQz/E9ono7H1DkebDa4iEYEEBECAAYFAj+0Q3cA
CgkQhZavqzBzTmbGwwCdFqD1frViC7WRt8GKoOS7hzNN32kAnirlbwpnT7a6NOsQ
```

```
83nk11a2dePhiEYEEBECAAYFAkNbs+oACgkQi9gubzC5S1x/dACdELKoXQKkwJN0
gZztsM7kjsIgyFMAnRRMbHQ7V39XC90OIpaPjk3a01tgiEYEExECAAYFAkTxMyYA
CgkQ9knE9GCTUwwKcQCgibak/SwhxWH1ijRhgYCo5GtM4vcAnAhtzL57wcw1Kg1X
m7nVGetUqJ7fiEwEEBECAAwFAkGBywEFgwYi2YsACgkQGFnQH2d7oexCjQCcD8sJ
NDc/mS8m8OGDUOx9VMWcnGkAnj1YWOD+Qhxo3mI/Ul9oEAhNkjcfiEwEEBECAAwF
AkGByzQFgwYi2VgACgkQgcL36+ITtpIiIwCdFVNVUB8xe8mFXoPm4d9Z54PTjpMA
niSPA/ZsfJ3oOMLKar4F0QPPrdrGiEwEEBECAAwFAkGBy2IFgwYi2SoACgkQa3Ds
2V3D9HMJqgCbBYzr5GPXOXgP88jKzmdbjweqXeEAnRss4G2G/3qD7uhTL1SPT1SH
jWUXiEwEEBECAAwFAkHQkyQFgwXUEWgACgkQfSXKCsEpp8JiVQCghvWvkPqowsw8
w7WSseTcw1tflvkAni+vLHl/DqIly0LkZYn5jzK1dpvfiEwEEBECAAwFAkIrW7oF
gwV5SNIACgkQ5hukiRXruavzEwCgkzL5QkLSypcw9LGHcFSx1ya0VL4An35nXkum
g6cCJ1NP8r2I4NcZWIrqiEwEEhECAAwFAkAqWToFgwd6S1IACgkQPKEfNJT6+GEm
XACcD+A53A5OGM7w750W11ukq4iZ9ckAnRMvndAqn3YTOxxlLPj2UPZiSgSqiEwE
EhECAAwFAkA9+roFgwdmqdIACgkQ8tdcY+OcZZyy3wCgtDcwlaq20w0cNuXFLLNe
EUaFFTwAni6RHN80moSVAdDTRkzZacJU3M5QiEwEEhECAAwFAkEOCoQFgwaWmggA
CgkQOcor9D1qil/83QCeITZ9wIo7XAMjC6y4ZWUL4m+edZsAoMOhRIRi42fmrNFu
vNZbnMGej81viEwEEhECAAwFAkKApTQFgwUj/1gACgkQBA3AhXyDn6jjJACcD1A4
UtXk84J13JQyoH9+dy24714Aniwlsso/9ndICJOkqs2j5dlHFq6oiEwEExECAAwF
Aj5NTYQFgwlXVwgACgkQLbt2v63UyTMFDACglT5G5NVKf5Mj65bFSlPzb92zk2QA
n1uc2h19/IwwrsbIyK/9POJ+JMP7iEwEExECAAwFAkHXgHYFgwXNJBYACgkQZu/b
yM2C/T4/vACfXe67xiSHB80wkmFZ2krb+oz/gBAAnjR2ucpbaonkQQgnC3GnBqmC
vNaJiEwEExECAAwFAkIYgQ4FgwWMI34ACgkQdsEDHKIxbqGg7gCfQi2HcrHn+yLF
uNlH1oSOh48ZM0oAn3hKV0uIRJphonHaUYiUP1ttWgdBiGUEExECAB0FCwcKAwQD
FQMCAxYCAQIXgAUCS3AvygUJEPPzpwASB2VHUEcAAQEJEIxxjTtQcuH1sNsAniYp
YBGqy/HhMnw3WE8kXahOOR5KAJ4xUmWPGYP4l3hKxyNK9OAUbpDVYIh7BDARAgA7
BQJCdzX1NB0AT29wcy4uLiBzaG91bGQgaGF2ZSBiZWVuIGxvY2FsISBJJ20gKnNv
KiBzdHVwaWQuLi4ACgkQOcor9D1qil/vRwCdFo08f66oKLiuEAqzlf9iDlPozEEA
n2EgvCYLCCHjfGosrkrU3WK5NFVgiI8EMBECAE8FAkVvAL9IHQBTaG91bGQgaGF2
ZSBiZWVuIGEgbG9jYWwgc2lnbmF0dXJlLCBvciBzb21ldGhpbmcgLSBXVEYgd2Fz
IEkgdGhpbmtpbmc/AAoJEDnKK/Q9aopfoPsAn3BVqKOalJeF0xPSvLR90PsRlnmG
AJ44oisY7Tl3NJbPgZal8W32fbqgbIkCIgQQAQIADAUCQYHLhQWDBiLZBwAKCRCq
4+bOZqFEaKgvEACCErnaHGyUYa0wETjj6DLEXsqeOiXad4i9aBQxnD35GUgcFofC
/nCY4XcnCMMEnmdQ9ofUuU3OBJ6BNJIbEusAabgLooebP/3KEaiCIiyhHYU5jarp
ZAh+Zopgs3Oc11mQ1tIaS69iJxrGTLodkAsAJAeEUwTPq9fHFFzC1eGBysoyFWg4
bIjz/zClI+qyTbFA5g6tRoiXTo8ko7QhY2AA5UGEg+83Hdb6akC04Z2QRErxKAqr
phHzj8XpjVOsQAdAi/qVKQeNKROlJ+iq6+YesmcWGfzeb87dGNweVFDJIGA0qY27
pTb2lExYjsRFN4Cb13NfodAbMTOxcAWZ7jAPCxAPlHUG++mHMrhQXEToZnBFE4nb
nC7vOBNgWdjUgXcpkUCkop4b17BFpR+k8ZtYLSS8p2LLz4uAeCcSm2/msJxT7rC/
FvoH8428oHincqs2ICo9zO/Ud4HmmO0O+SsZdVKIIjinGyOVWb4OOzkAlnnhEZ3o
6hAHcREIsBgPwEYVTj/9ZdC0AO44Nj9cU7awaqgtrnwwfr/o4V2gl8bLSkltZU27
/29HeuOeFGjlFe0YrDd/aRNsxbyb2O28H4sG1CVZmC5uK1iQBDiSyA7Q0bbdofCW
oQzm5twlpKWnY8Oe0ub9XP5p/sVfck4FceWFHwv+/PC9RzSl33lQ6vM2wIkCIgQT
AQIADAUCQp8KHAWDBQWacAAKCRDYwgoJWiRXzyE+D/9uc7z6fIsalfOYoLN60ajA
bQbI/uRKBFugyZ5RoaItusn9Z2rAtn61WrFhu4uCSJtFN1ny2RERg40f56pTghKr
D+YEt+Nze6+FKQ5AbGIdFsR/2bUk+ZZRSt83e14Lcb6ii/fJfzkoIox9ltkifQxq
Y7Tvk4noKu4oLSc8O1Wsfc/y0B9sYUUCmUfcnq58DEmGie9ovUslmyt5NPnveXxp
5UeaRc5Rqt9tK2B4A+7/cqENrdZJbAMSunt2+2fkYiRunAFPKPBdJBsY1sxeL/A9
aKe0viKEXQdAWqdNZKNCi8rd/oOP99/9lMbFudAbX6nL2DSb1OG2Z7NWEqgIAzjm
pwYYPCKeVz5Q8R+if9/fe5+STY/55OaI33fJ2H3v+U435VjYqbrerWe36xJItcJe
qUzW71fQtXi1CTEl3w2ch7VF5oj/QyjabLnAlHgSlkSi6p7By5C2MnbCHlCfPnIi
nPhFoRcRGPjJe9nFwGs+QblvS/Chzc2WX3s/2SWm4gEUKRX4zsAJ5ocyfa/vkxCk
SxK/erWlCPf/J1T70+i5waXDN/E3enSet/WL7h94pQKpjz8OdGL4JSBHuAVGA+a+
dknqnPF0KMKLhjrgV+L7O84FhbmAP7PXm3xmiMPriXf+el5fZZequQoIagf8rdRH
HhRJxQgI0HNknkaOqs8dtrkCDQQ+PqMdEAgA7+GJfxbMdY4wslPnjH9rF4N2qfWs
EN/lxaZoJYc3a6M02WCnHl6ahT2/tBK2w1QI4YFteR47gCvtgb6O1JHffOo2HfLm
RDRiRjd1DTCHqeyX7CHhcghj/dNRlW2Z0l5QFEcmV9U0Vhp3aFfWC4Ujfs3LU+hk
AWzE7zaD5cH9J7yv/6xuZVw411x0h4UqsTcWMu0iM1BzELqX1DY7LwoPEb/O9Rkb
f4fmLe11EzIaCa4PqARXQZc4dhSinMt6K3X4BrRsKTfozBu74F47D8Ilbf5vSYHb
uE5p/1oIDznkg/p8kW+3FxuWrycciqFTcNz215yyX39LXFnlLzKUb/F5GwADBQf+
Lwqqa8CGrRfsOAJxim63CHfty5mUc5rUSnTslGYEIOCR1BeQauyPZbPDsDD9MZ1Z
aSafanFvwFG6Llx9xkU7tzq+vKLoWkm4u5xf3vn55VjnSd1aQ9eQnUcXiL4cnBGo
TbOWI39EcyzgslzBdC++MPjcQTcA7p6JUVsP6oAB3FQWg54tuUo0Ec8bsM8b3Ev4
2LmuQT5NdKHGwHsXTPtl0klk4bQk4OajHsiy1BMahpT27jWjJlMiJc+IWJ0mghkK
Ht926s/ymfdf5HkdQ1cyvsz5tryVI3Fx78XeSYfQvuuwqp2H139pXGEkg0n6KdUO
etdZWhe70YGNPw1yjWJT1IhUBBgRAgAMBQJOdz3tBQkT+wG4ABIHZUdQRwABAQkQ
jHGNO1By4fUUmwCbBYr2+bBEn/L2BOcnw9Z/QFWuhRMAoKVgCFm5fadQ3Afi+UQl
AcOphrnJ
```

```
=443I
-----END PGP PUBLIC KEY BLOCK-----
```

To import the build key into your personal public GPG keyring, use `gpg --import`. For example, if you have saved the key in a file named `mysql_pubkey.asc`, the import command looks like this:

```
shell> gpg --import mysql_pubkey.asc
gpg: key 5072E1F5: public key "MySQL Release Engineering
<mysql-build@oss.oracle.com>" imported
gpg: Total number processed: 1
gpg:               imported: 1
gpg: no ultimately trusted keys found
```

You can also download the key from the public keyserver using the public key id, `5072E1F5`:

```
shell> gpg --recv-keys 5072E1F5
gpg: requesting key 5072E1F5 from hkp server keys.gnupg.net
gpg: key 5072E1F5: "MySQL Release Engineering <mysql-build@oss.oracle.com>"
1 new user ID
gpg: key 5072E1F5: "MySQL Release Engineering <mysql-build@oss.oracle.com>"
53 new signatures
gpg: no ultimately trusted keys found
gpg: Total number processed: 1
gpg:           new user IDs: 1
gpg:         new signatures: 53
```

If you want to import the key into your RPM configuration to validate RPM install packages, you should be able to import the key directly:

```
shell> rpm --import mysql_pubkey.asc
```

If you experience problems or require RPM specific information, see Section 2.1.4.4, “Signature Checking Using RPM”.

After you have downloaded and imported the public build key, download your desired MySQL package and the corresponding signature, which also is available from the download page. The signature file has the same name as the distribution file with an `.asc` extension, as shown by the examples in the following table.

Table 2.1 MySQL Package and Signature Files for Source files

File Type	File Name
Distribution file	`mysql-standard-5.7.5-linux-i686.tar.gz`
Signature file	`mysql-standard-5.7.5-linux-i686.tar.gz.asc`

Make sure that both files are stored in the same directory and then run the following command to verify the signature for the distribution file:

```
shell> gpg --verify package_name.asc
```

If the downloaded package is valid, you will see a "Good signature" similar to:

```
shell> gpg --verify mysql-standard-5.7.5-linux-i686.tar.gz.asc
gpg: Signature made Tue 01 Feb 2011 02:38:30 AM CST using DSA key ID 5072E1F5
```

```
gpg: Good signature from "MySQL Release Engineering <mysql-build@oss.oracle.com>"
```

The `Good signature` message indicates that the file signature is valid, when compared to the signature listed on our site. But you might also see warnings, like so:

```
shell> gpg --verify mysql-standard-5.7.5-linux-i686.tar.gz.asc
gpg: Signature made Wed 23 Jan 2013 02:25:45 AM PST using DSA key ID 5072E1F5
gpg: checking the trustdb
gpg: no ultimately trusted keys found
gpg: Good signature from "MySQL Release Engineering <mysql-build@oss.oracle.com>"
gpg: WARNING: This key is not certified with a trusted signature!
gpg:          There is no indication that the signature belongs to the owner.
Primary key fingerprint: A4A9 4068 76FC BD3C 4567  70C8 8C71 8D3B 5072 E1F5
```

That is normal, as they depend on your setup and configuration. Here are explanations for these warnings:

- *gpg: no ultimately trusted keys found*: This means that the specific key is not "ultimately trusted" by you or your web of trust, which is okay for the purposes of verifying file signatures.

- *WARNING: This key is not certified with a trusted signature! There is no indication that the signature belongs to the owner.*: This refers to your level of trust in your belief that you possess our real public key. This is a personal decision. Ideally, a MySQL developer would hand you the key in person, but more commonly, you downloaded it. Was the download tampered with? Probably not, but this decision is up to you. Setting up a web of trust is one method for trusting them.

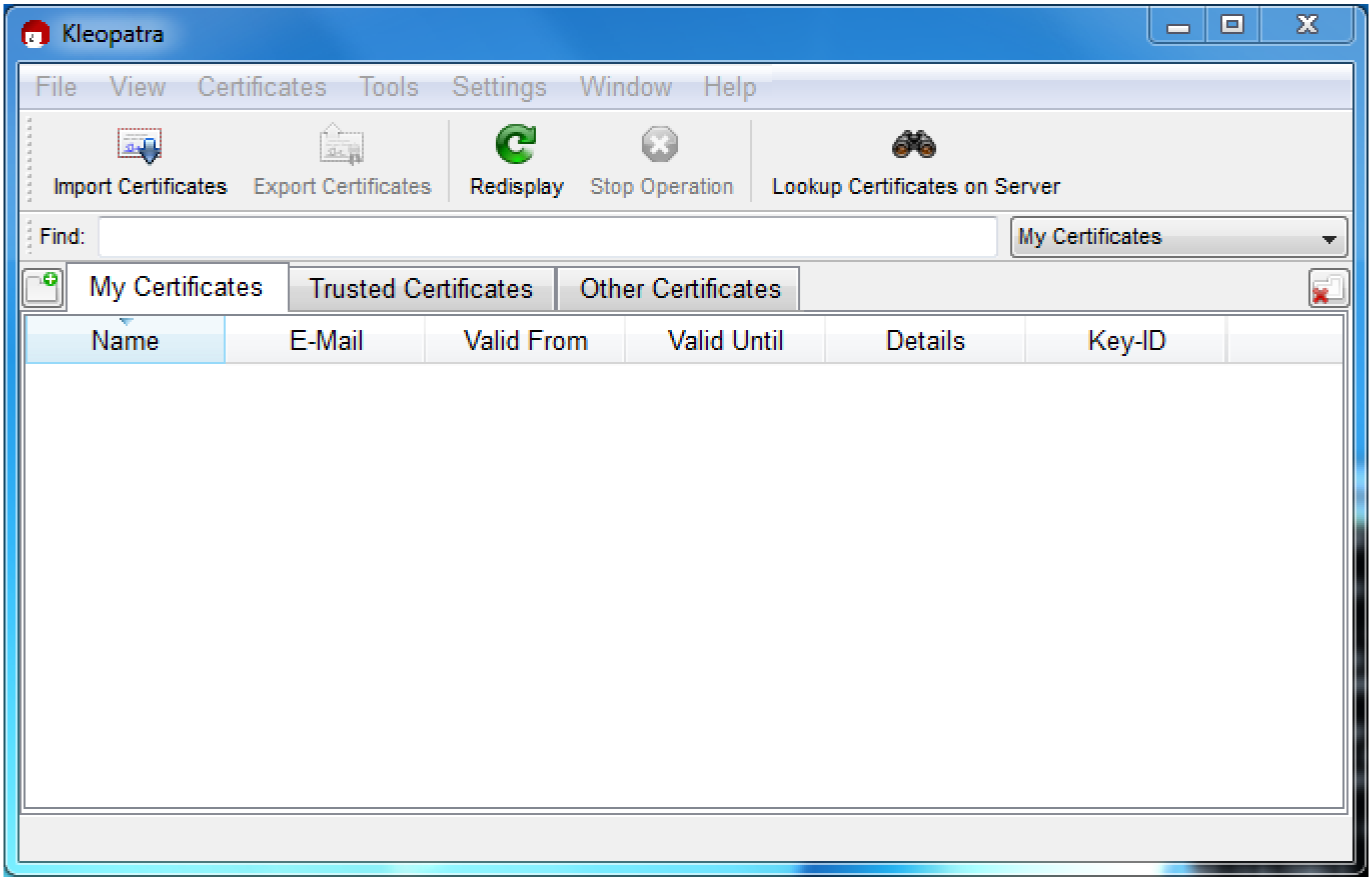
See the GPG documentation for more information on how to work with public keys.

2.1.4.3 Signature Checking Using Gpg4win for Windows

The Section 2.1.4.2, “Signature Checking Using GnuPG” section describes how to verify MySQL downloads using GPG. That guide also applies to Microsoft Windows, but another option is to use a GUI tool like Gpg4win. You may use a different tool but our examples are based on Gpg4win, and utilize its bundled `Kleopatra` GUI.

Download and install Gpg4win, and then load Kleopatra. The dialog should look similar to:

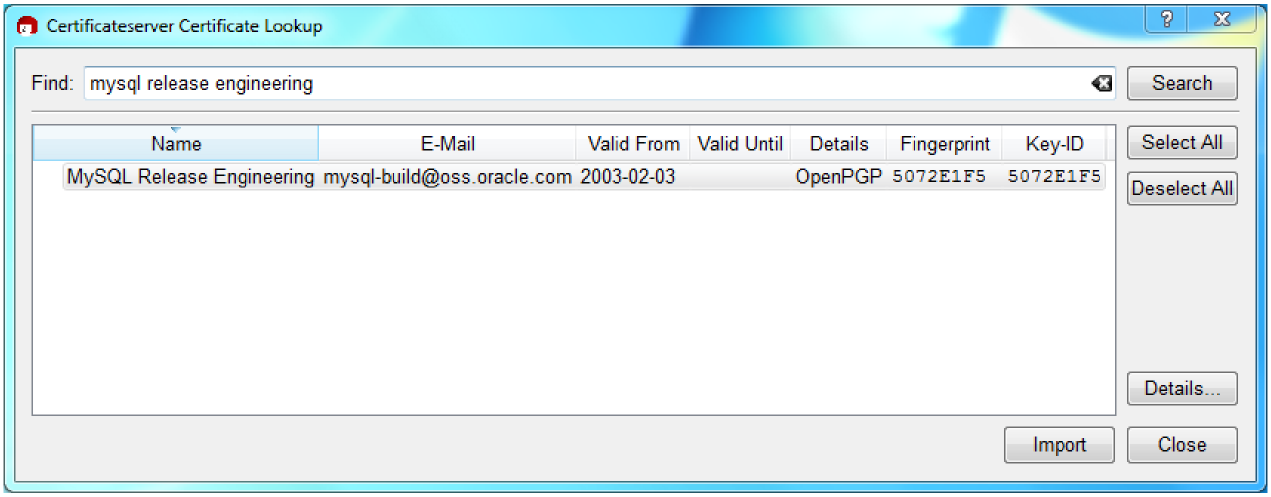
Figure 2.1 Initial screen after loading Kleopatra



Name	E-Mail	Valid From	Valid Until	Details	Key-ID

Next, add the MySQL Release Engineering certificate. Do this by clicking File, Lookup Certificates on Server. Type "Mysql Release Engineering" into the search box and press Search.

Figure 2.2 Finding the MySQL Release Engineering certificate



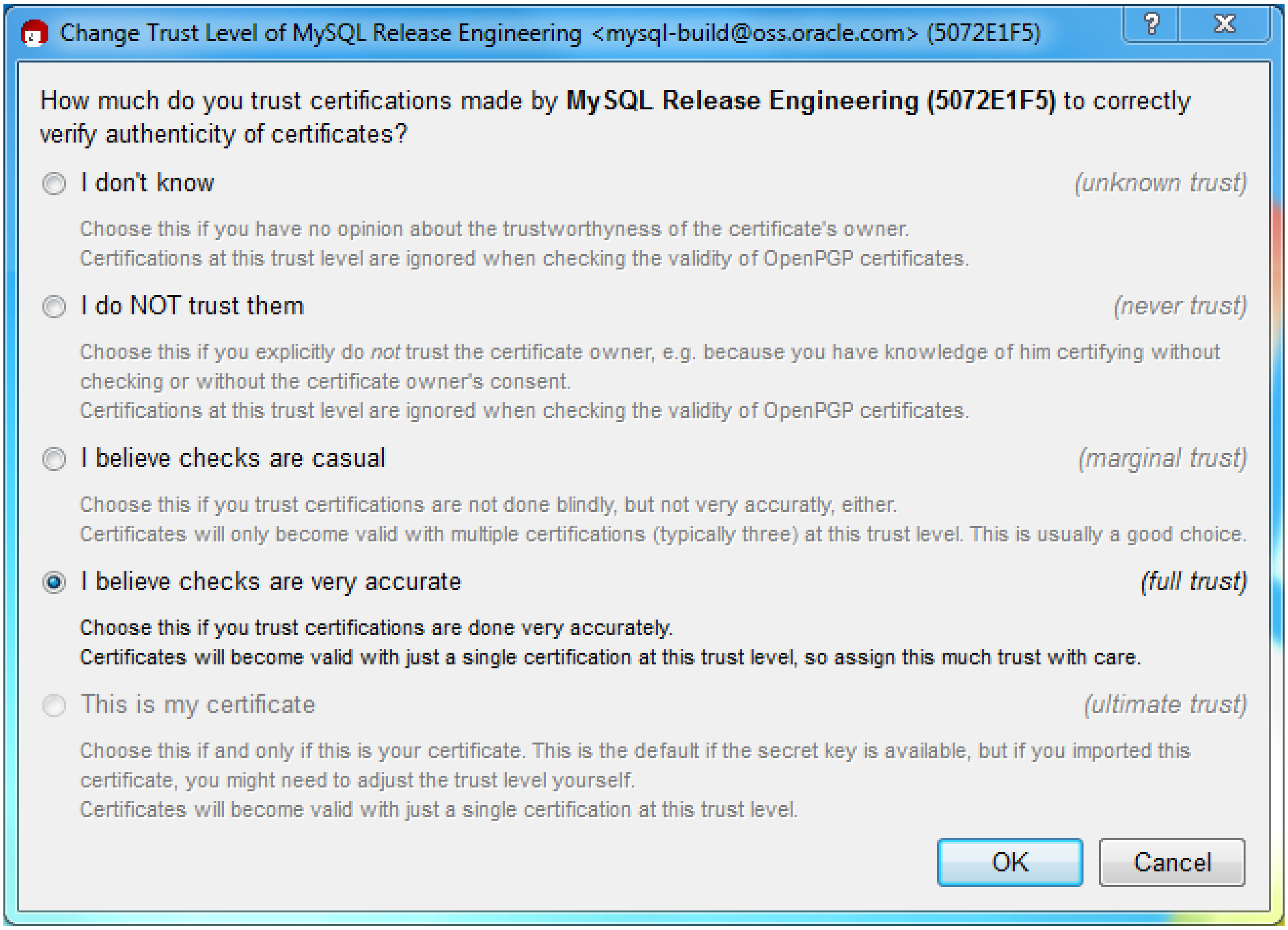
Find: mysql release engineering

Name	E-Mail	Valid From	Valid Until	Details	Fingerprint	Key-ID
MySQL Release Engineering	mysql-build@oss.oracle.com	2003-02-03		OpenPGP	5072E1F5	5072E1F5

Select the "MySQL Release Engineering" certificate. The Fingerprint and Key-ID must be "5072E1F5", or choose Details... to confirm the certificate is valid. Now, import it by clicking Import. An import dialog will be displayed, choose Okay, and this certificate will now be listed under the **Imported Certificates** tab.

Next, configure the trust level for our certificate. Select our certificate, then from the main menu select Certificates, Change Owner Trust.... We suggest choosing **I believe checks are very accurate** for our certificate, as otherwise you might not be able to verify our signature. Select **I believe checks are very accurate** and then press OK.

Figure 2.3 Changing the Trust level



Change Trust Level of MySQL Release Engineering <mysql-build@oss.oracle.com> (5072E1F5)

How much do you trust certifications made by **MySQL Release Engineering (5072E1F5)** to correctly verify authenticity of certificates?

○ I don't know *(unknown trust)*

Choose this if you have no opinion about the trustworthyness of the certificate's owner.
Certifications at this trust level are ignored when checking the validity of OpenPGP certificates.

○ I do NOT trust them *(never trust)*

Choose this if you explicitly do *not* trust the certificate owner, e.g. because you have knowledge of him certifying without checking or without the certificate owner's consent.
Certifications at this trust level are ignored when checking the validity of OpenPGP certificates.

○ I believe checks are casual *(marginal trust)*

Choose this if you trust certifications are not done blindly, but not very accuratly, either.
Certificates will only become valid with multiple certifications (typically three) at this trust level. This is usually a good choice.

● I believe checks are very accurate *(full trust)*

Choose this if you trust certifications are done very accurately.
Certificates will become valid with just a single certification at this trust level, so assign this much trust with care.

○ This is my certificate *(ultimate trust)*

Choose this if and only if this is your certificate. This is the default if the secret key is available, but if you imported this certificate, you might need to adjust the trust level yourself.
Certificates will become valid with just a single certification at this trust level.

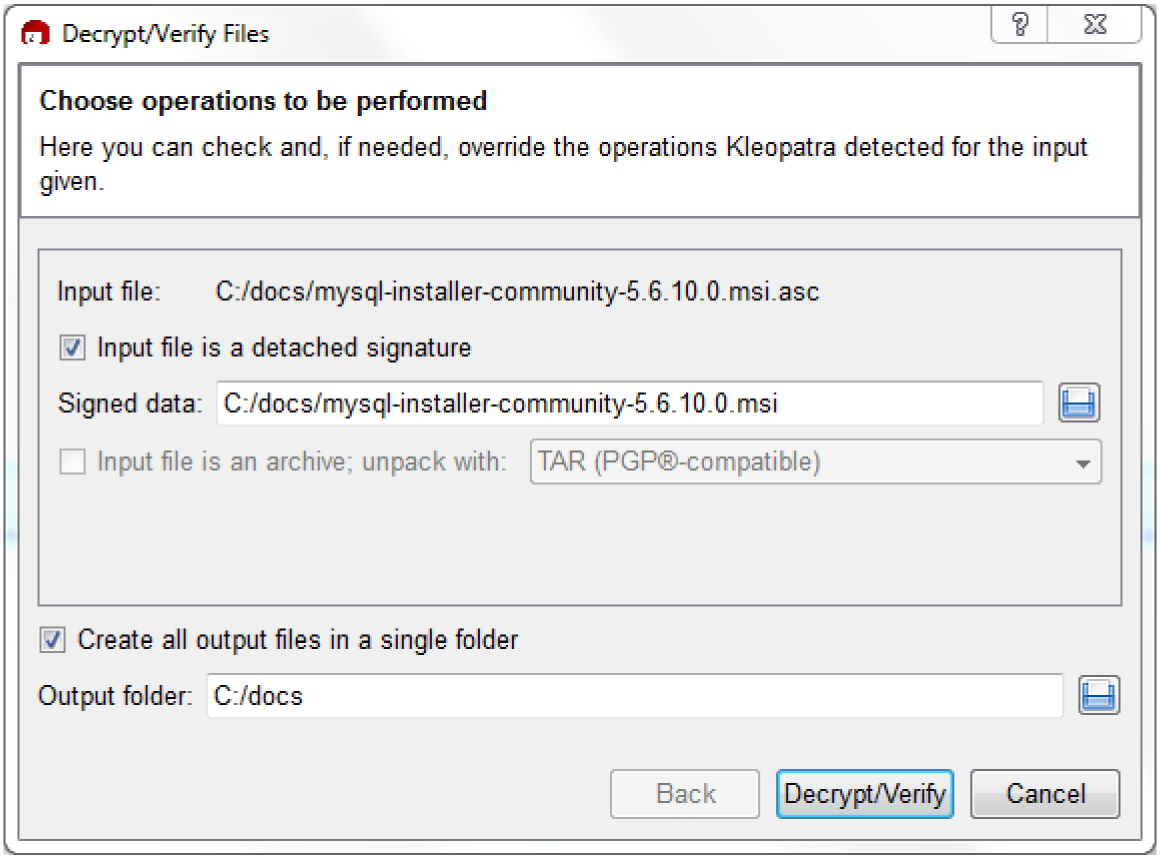
Next, verify the downloaded MySQL package file. This requires files for both the packaged file, and the signature. The signature file must have the same name as the packaged file but with an appended `.asc` extension, as shown by the example in the following table. The signature is linked to on the downloads page for each MySQL product. You must create the `.asc` file with this signature.

Table 2.2 MySQL Package and Signature Files for MySQL Installer for Microsoft Windows

File Type	File Name
Distribution file	`mysql-installer-community-5.7.5.msi`
Signature file	`mysql-installer-community-5.7.5.msi.asc`

Make sure that both files are stored in the same directory and then run the following command to verify the signature for the distribution file. Either drag and drop the signature (`.asc`) file into Kleopatra, or load the dialog from File, Decrypt/Verify Files..., and then choose either the `.msi` or `.asc` file.

Figure 2.4 The Decrypt/Verify Files dialog



Decrypt/Verify Files

Choose operations to be performed

Here you can check and, if needed, override the operations Kleopatra detected for the input given.

Input file: C:/docs/mysql-installer-community-5.6.10.0.msi.asc

☑ Input file is a detached signature

Signed data: C:/docs/mysql-installer-community-5.6.10.0.msi

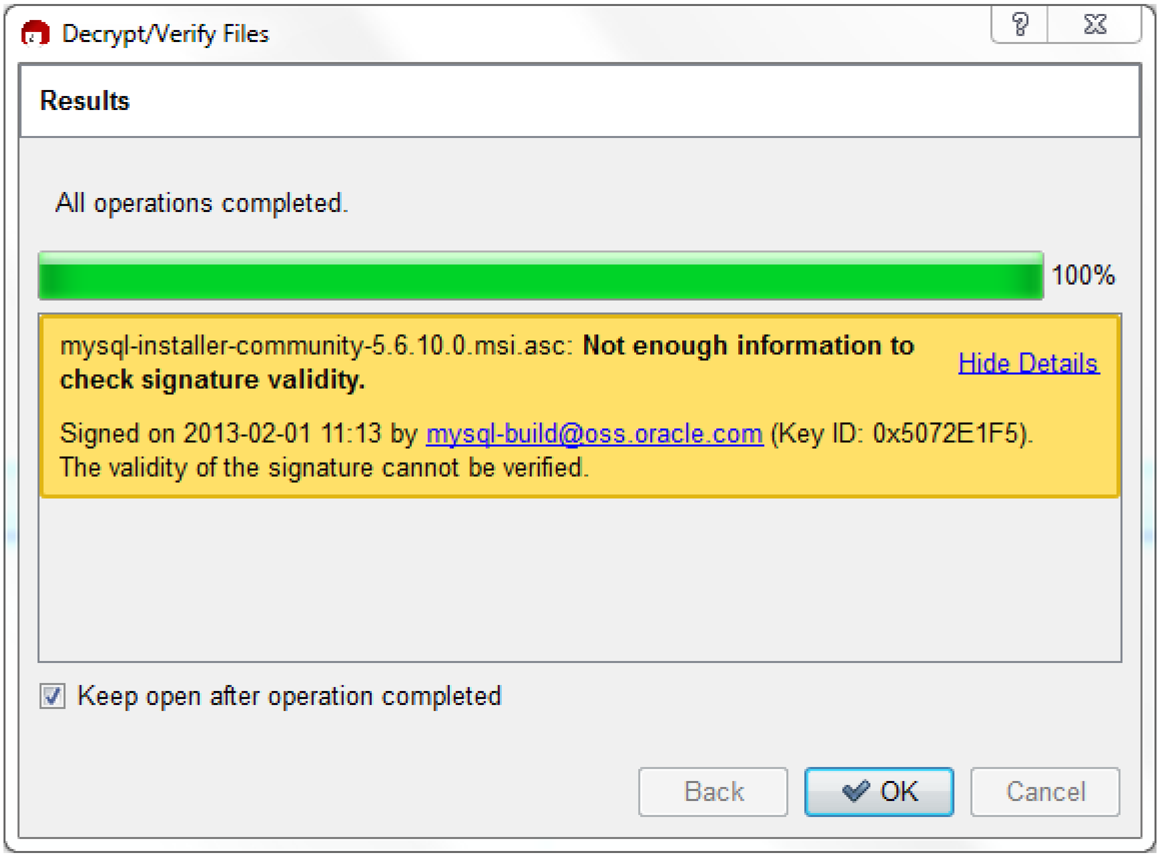
□ Input file is an archive; unpack with: TAR (PGP®-compatible)

☑ Create all output files in a single folder

Output folder: C:/docs

Click Decrypt/Verify to check the file. The two most common results will look like the following, and although the yellow warning looks problematic, the following means that the file check passed with success. You may now run this installer.

Figure 2.5 The Decrypt/Verify Results: Good



All operations completed.

100%

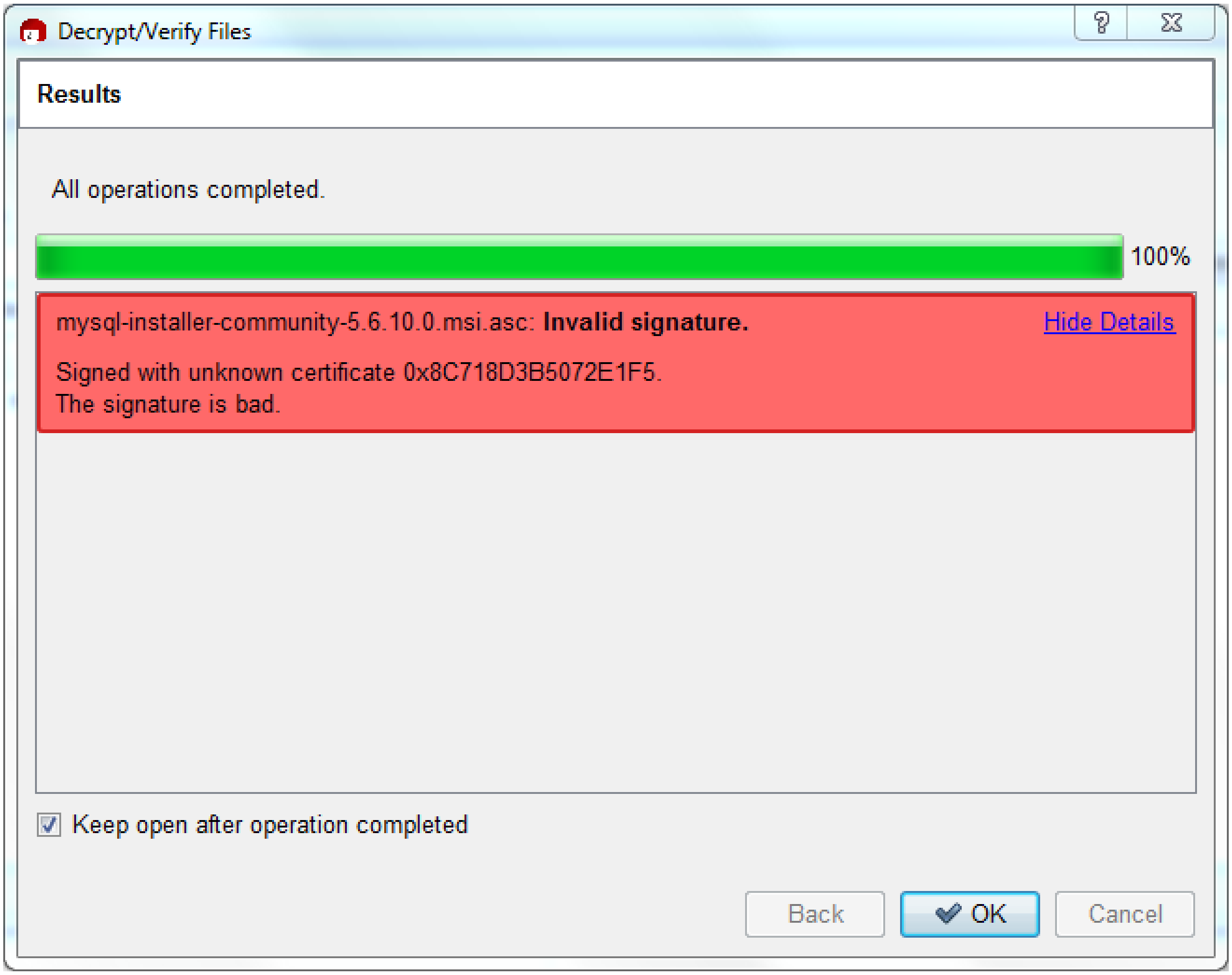
mysql-installer-community-5.6.10.0.msi.asc: **Not enough information to check signature validity.**

Signed on 2013-02-01 11:13 by mysql-build@oss.oracle.com (Key ID: 0x5072E1F5). The validity of the signature cannot be verified.

☑ Keep open after operation completed

Seeing a red "The signature is bad" error means the file is invalid. Do not execute the MSI file if you see this error.

Figure 2.6 The Decrypt/Verify Results: Bad



All operations completed.

mysql-installer-community-5.6.10.0.msi.asc: **Invalid signature.**

Signed with unknown certificate 0x8C718D3B5072E1F5.
The signature is bad.

☑ Keep open after operation completed

The Section 2.1.4.2, “Signature Checking Using GnuPG” section explains why you probably don't see a green `Good signature` result.

2.1.4.4 Signature Checking Using RPM

For RPM packages, there is no separate signature. RPM packages have a built-in GPG signature and MD5 checksum. You can verify a package by running the following command:

```
shell> rpm --checksig package_name.rpm
```

Example:

```
shell> rpm --checksig MySQL-server-5.7.5-0.linux_glibc2.5.i386.rpm
MySQL-server-5.7.5-0.linux_glibc2.5.i386.rpm: md5 gpg OK
```

Note

If you are using RPM 4.1 and it complains about `(GPG) NOT OK (MISSING KEYS: GPG#5072e1f5)`, even though you have imported the MySQL public build

> key into your own GPG keyring, you need to import the key into the RPM keyring first. RPM 4.1 no longer uses your personal GPG keyring (or GPG itself). Rather, RPM maintains a separate keyring because it is a system-wide application and a user's GPG public keyring is a user-specific file. To import the MySQL public key into the RPM keyring, first obtain the key, then use `rpm --import` to import the key. For example:

```
shell> gpg --export -a 5072e1f5 > 5072e1f5.asc
shell> rpm --import 5072e1f5.asc
```

Alternatively, `rpm` also supports loading the key directly from a URL, and you can use this manual page:

```
shell> rpm --import http://dev.mysql.com/doc/refman/5.7/en/checking-gpg-signature.html
```

If you need to obtain the MySQL public key, see Section 2.1.4.2, “Signature Checking Using `GnuPG`”.

2.1.5 Installation Layouts

The installation layout differs for different installation types (for example, native packages, binary tarballs, and source tarballs), which can lead to confusion when managing different systems or using different installation sources. The individual layouts are given in the corresponding installation type or platform chapter, as described following. Note that the layout of installations from vendors other than Oracle may differ from these layouts.

- Section 2.3.1, “MySQL Installation Layout on Microsoft Windows”
- Section 2.8.1, “MySQL Layout for Source Installation”
- Table 2.3, “MySQL Installation Layout for Generic Unix/Linux Binary Package”
- Table 2.9, “MySQL Installation Layout for Linux RPM Packages”
- Table 2.6, “MySQL Installation Layout on Mac OS X”

2.1.6 Compiler-Specific Build Characteristics

In some cases, the compiler used to build MySQL affects the features available for use. The notes in this section apply for binary distributions provided by Oracle Corporation or that you compile yourself from source.

`icc` (Intel C++ Compiler) Builds

A server built with `icc` has these characteristics:

- SSL support is not included.

2.2 Installing MySQL on Unix/Linux Using Generic Binaries

Oracle provides a set of binary distributions of MySQL. These include binary distributions in the form of compressed `tar` files (files with a `.tar.gz` extension) for a number of platforms, as well as binaries in platform-specific package formats for selected platforms.

This section covers the installation of MySQL from a compressed `tar` file binary distribution. For other platform-specific package formats, see the other platform-specific sections. For example, for Windows distributions, see Section 2.3, “Installing MySQL on Microsoft Windows”.

To obtain MySQL, see Section 2.1.3, “How to Get MySQL”.

MySQL compressed `tar` file binary distributions have names of the form `mysql-VERSION-OS.tar.gz`, where *`VERSION`* is a number (for example, `5.7.5`), and *`OS`* indicates the type of operating system for which the distribution is intended (for example, `pc-linux-i686` or `winx64`).

To install MySQL from a compressed `tar` file binary distribution, your system must have GNU `gunzip` to uncompress the distribution and a reasonable `tar` to unpack it. If your `tar` program supports the `z` option, it can both uncompress and unpack the file.

GNU `tar` is known to work. The standard `tar` provided with some operating systems is not able to unpack the long file names in the MySQL distribution. You should download and install GNU `tar`, or if available, use a preinstalled version of GNU tar. Usually this is available as `gnutar`, `gtar`, or as `tar` within a GNU or Free Software directory, such as `/usr/sfw/bin` or `/usr/local/bin`. GNU `tar` is available from http://www.gnu.org/software/tar/.

Warning

If you have previously installed MySQL using your operating system native package management system, such as `yum` or `apt-get`, you may experience problems installing using a native binary. Make sure your previous MySQL previous installation has been removed entirely (using your package management system), and that any additional files, such as old versions of your data files, have also been removed. You should also check the existence of configuration files such as `/etc/my.cnf` or the `/etc/mysql` directory have been deleted.

If you run into problems and need to file a bug report, please use the instructions in Section 1.7, “How to Report Bugs or Problems”.

On Unix, to install a compressed `tar` file binary distribution, unpack it at the installation location you choose (typically `/usr/local/mysql`). This creates the directories shown in the following table.

Table 2.3 MySQL Installation Layout for Generic Unix/Linux Binary Package

Directory	Contents of Directory
`bin`	Client programs and the `mysqld` server
`data`	Log files, databases
`docs`	Manual in Info format
`man`	Unix manual pages
`include`	Include (header) files
`lib`	Libraries
`scripts`	`mysql_install_db`
`share`	Miscellaneous support files, including error messages, sample configuration files, SQL for database installation
`sql-bench`	Benchmarks

Debug versions of the `mysqld` binary are available as `mysqld-debug`. To compile your own debug version of MySQL from a source distribution, use the appropriate configuration options to enable debugging support. For more information on compiling from source, see Section 2.8, “Installing MySQL from Source”.

To install and use a MySQL binary distribution, the basic command sequence looks like this:

```
shell> groupadd mysql
shell> useradd -r -g mysql mysql
shell> cd /usr/local
shell> tar zxvf /path/to/mysql-VERSION-OS.tar.gz
shell> ln -s full-path-to-mysql-VERSION-OS mysql
shell> cd mysql
shell> chown -R mysql .
shell> chgrp -R mysql .
shell> scripts/mysql_install_db --user=mysql
shell> chown -R root .
shell> chown -R mysql data
shell> bin/mysqld_safe --user=mysql &
# Next command is optional
shell> cp support-files/mysql.server /etc/init.d/mysql.server
```

mysql_install_db creates a default option file named my.cnf in the base installation directory. This file is created from a template included in the distribution package named my-default.cnf. For more information, see Using a Sample Default Server Configuration File.

A more detailed version of the preceding description for installing a binary distribution follows.

Note

This procedure assumes that you have root (administrator) access to your system. Alternatively, you can prefix each command using the sudo (Linux) or pfexec (OpenSolaris) command.

The procedure does not set up any passwords for MySQL accounts. After following the procedure, proceed to Section 2.9.2, “Securing the Initial MySQL Accounts”.

Create a mysql User and Group

If your system does not already have a user and group for mysqld to run as, you may need to create one. The following commands add the mysql group and the mysql user. You might want to call the user and group something else instead of mysql. If so, substitute the appropriate name in the following instructions. The syntax for useradd and groupadd may differ slightly on different versions of Unix, or they may have different names such as adduser and addgroup.

```
shell> groupadd mysql
shell> useradd -r -g mysql mysql
```

Note

Because the user is required only for ownership purposes, not login purposes, the useradd command uses the -r option to create a user that does not have login permissions to your server host. Omit this option to permit logins for the user (or if your useradd does not support the option).

Obtain and Unpack the Distribution

Pick the directory under which you want to unpack the distribution and change location into it. The example here unpacks the distribution under /usr/local. The instructions, therefore, assume that you have permission to create files and directories in /usr/local. If that directory is protected, you must perform the installation as root.

```
shell> cd /usr/local
```

Obtain a distribution file using the instructions in Section 2.1.3, “How to Get MySQL”. For a given release, binary distributions for all platforms are built from the same MySQL source distribution.

Unpack the distribution, which creates the installation directory. Then create a symbolic link to that directory. `tar` can uncompress and unpack the distribution if it has `z` option support:

```
shell> tar zxvf /path/to/mysql-VERSION-OS.tar.gz
shell> ln -s full-path-to-mysql-VERSION-OS mysql
```

The `tar` command creates a directory named `mysql-VERSION-OS`. The `ln` command makes a symbolic link to that directory. This enables you to refer more easily to the installation directory as `/usr/local/mysql`.

If your `tar` does not have `z` option support, use `gunzip` to unpack the distribution and `tar` to unpack it. Replace the preceding `tar` command with the following alternative command to uncompress and extract the distribution:

```
shell> gunzip < /path/to/mysql-VERSION-OS.tar.gz | tar xvf -
```

Perform Postinstallation Setup

The remainder of the installation process involves setting up the configuration file, creating the core databases, and starting the MySQL server. For next steps, see Section 2.9, “Postinstallation Setup and Testing”.

Note

The accounts that are listed in the MySQL grant tables initially have no passwords. After starting the server, you should set up passwords for them using the instructions in Section 2.9.2, “Securing the Initial MySQL Accounts”.

2.3 Installing MySQL on Microsoft Windows

MySQL is available for Microsoft Windows, for both 32-bit and 64-bit versions. For supported Windows platform information, see http://www.mysql.com/support/supportedplatforms/database.html.

It is possible to run MySQL as a standard application or as a Windows service. By using a service, you can monitor and control the operation of the server through the standard Windows service management tools. For more information, see Section 2.3.5.7, “Starting MySQL as a Windows Service”.

Generally, you should install MySQL on Windows using an account that has administrator rights. Otherwise, you may encounter problems with certain operations such as editing the `PATH` environment variable or accessing the `Service Control Manager`. Once installed, MySQL does not need to be executed using a user with Administrator privileges.

For a list of limitations on the use of MySQL on the Windows platform, see Section E.10.6, “Windows Platform Limitations”.

In addition to the MySQL Server package, you may need or want additional components to use MySQL with your application or development environment. These include, but are not limited to:

- To connect to the MySQL server using ODBC, you must have a Connector/ODBC driver. For more information, including installation and configuration instructions, see MySQL Connector/ODBC Developer Guide. But note that MySQL Installer will install and configure Connector/ODBC for you.

- To use MySQL server with .NET applications, you must have the Connector/Net driver. For more information, including installation and configuration instructions, see MySQL Connector/Net Developer Guide. But note that MySQL Installer will install and configure Connector/NET for you.

MySQL distributions for Windows can be downloaded from http://dev.mysql.com/downloads/. See Section 2.1.3, “How to Get MySQL”.

MySQL for Windows is available in several distribution formats, detailed following. Generally speaking, you should use MySQL Installer. It is simpler to use than the Zip file, and you need no additional tools to get MySQL up and running. MySQL Installer automatically installs MySQL Server and additional MySQL products, creates an options file, starts the server, and enables you to create default user accounts. For more information on choosing a package, see Section 2.3.2, “Choosing An Installation Package”.

- A MySQL Installer distribution includes MySQL Server and additional MySQL products including MySQL Workbench, MySQL Notifier, and MySQL for Excel. MySQL Installer can also be used to upgrade these products in the future.

  For instructions on installing MySQL using MySQL Installer, see Section 2.3.3, “Installing MySQL on Microsoft Windows Using MySQL Installer”.

- The standard binary distribution (packaged as a Zip file) contains all of the necessary files that you unpack into your chosen location. This package contains all of the files in the full Windows MSI Installer package, but does not include an installation program.

  For instructions on installing MySQL using the Zip file, see Section 2.3.5, “Installing MySQL on Microsoft Windows Using a `noinstall` Zip Archive”.

- The source distribution format contains all the code and support files for building the executables using the Visual Studio compiler system.

  For instructions on building MySQL from source on Windows, see Section 2.8, “Installing MySQL from Source”.

MySQL on Windows considerations:

- **Large Table Support**

  If you need tables with a size larger than 4GB, install MySQL on an NTFS or newer file system. Do not forget to use `MAX_ROWS` and `AVG_ROW_LENGTH` when you create tables. See Section 13.1.14, “`CREATE TABLE` Syntax”.

- **MySQL and Virus Checking Software**

  Virus-scanning software such as Norton/Symantec Anti-Virus on directories containing MySQL data and temporary tables can cause issues, both in terms of the performance of MySQL and the virus-scanning software misidentifying the contents of the files as containing spam. This is due to the fingerprinting mechanism used by the virus-scanning software, and the way in which MySQL rapidly updates different files, which may be identified as a potential security risk.

  After installing MySQL Server, it is recommended that you disable virus scanning on the main directory (`datadir`) used to store your MySQL table data. There is usually a system built into the virus-scanning software to enable specific directories to be ignored.

  In addition, by default, MySQL creates temporary files in the standard Windows temporary directory. To prevent the temporary files also being scanned, configure a separate temporary directory for MySQL temporary files and add this directory to the virus scanning exclusion list. To do this, add a configuration option for the tmpdir parameter to your my.ini configuration file. For more information, see Section 2.3.5.2, “Creating an Option File”.

2.3.1 MySQL Installation Layout on Microsoft Windows

For MySQL 5.7 on Windows, the default installation directory is C:\Program Files\MySQL\MySQL Server 5.7. Some Windows users prefer to install in C:\mysql, the directory that formerly was used as the default. However, the layout of the subdirectories remains the same.

All of the files are located within this parent directory, using the structure shown in the following table.

Table 2.4 Default MySQL Installation Layout for Microsoft Windows

Directory	Contents of Directory	Notes
bin	Client programs and the mysqld server	
%ALLUSERSPROFILE%\MySQL\MySQL Server 5.7\	Log files, databases (Windows XP, Windows Server 2003)	The Windows system variable %ALLUSERSPROFILE% defaults to C:\Documents and Settings\All Users\Application Data
%PROGRAMDATA%\MySQL\MySQL Server 5.7\	Log files, databases (Vista, Windows 7, Windows Server 2008, and newer)	The Windows system variable %PROGRAMDATA% defaults to C:\ProgramData
examples	Example programs and scripts	
include	Include (header) files	
lib	Libraries	
scripts	Utility scripts	
share	Miscellaneous support files, including error messages, character set files, sample configuration files, SQL for database installation	

If you install MySQL using the MySQL Installer, this package creates and sets up the data directory that the installed server will use, and also creates a pristine “template” data directory named data under the installation directory. After an installation has been performed using this package, the template data directory can be copied to set up additional MySQL instances. See Section 5.3, “Running Multiple MySQL Instances on One Machine”.

2.3.2 Choosing An Installation Package

For MySQL 5.7, there are installation package formats to choose from when installing MySQL on Windows:

- **MySQL Installer**: This package has a file name similar to mysql-installer-community-5.7.5.0.msi or mysql-installer-commercial-5.7.5.0.msi, and utilizes MSIs to automatically install MySQL server and other products. It will download and apply updates to itself, and for each of the installed products. It also configures the additional non-server products.

  The installed products are configurable, and this includes: documentation with samples and examples, connectors (such as C, C++, J, NET, and ODBC), MySQL Workbench, MySQL Notifier for Microsoft Windows, MySQL for Excel, and the MySQL Server with its components.

MySQL Installer will run on all Windows platforms that are supported by MySQL (see http://www.mysql.com/support/supportedplatforms/database.html).

> **Note**
>
> Because MySQL Installer is not a native component of Microsoft Windows and depends on .NET, it will not work on minimal installation options like the "Server Core" version of Windows Server 2008.

For instructions on installing MySQL using MySQL Installer, see Section 2.3.3, “Installing MySQL on Microsoft Windows Using MySQL Installer”.

- **The Noinstall Archive**: This package has a file name similar to `mysql-5.7.5-win32.zip` or `mysql-5.7.5-winx64.zip`, and contains all the files found in the Complete install package, with the exception of the GUI. This package does not include an automated installer, and must be manually installed and configured.

Your choice of install package affects the installation process you must follow. If you choose to use MySQL Installer, see Section 2.3.3, “Installing MySQL on Microsoft Windows Using MySQL Installer”. If you choose to install a Noinstall archive, see Section 2.3.5, “Installing MySQL on Microsoft Windows Using a `noinstall` Zip Archive”.

2.3.3 Installing MySQL on Microsoft Windows Using MySQL Installer

MySQL Installer is an application that simplifies the installation and updating process for a wide range of MySQL products, including MySQL Notifier for Microsoft Windows, MySQL Workbench, and MySQL for Excel. From this central application, you can see which MySQL products are already installed, configure them, and update or remove them if necessary. The installer can also install plugins, documentation, tutorials, and example databases. The MySQL Installer is only available for Microsoft Windows, and includes both a GUI and command-line interface.

Installer package types

- `Full:` Bundles all of the MySQL products (including MySQL Server). The file' size is over 160MB, and its name has the form `mysql-installer-community-`*`VERSION`*`.`*`N`*`.msi` where `VERSION` is the MySQL Server version number such as 5.6 and `N` is the package number, which begins at 0.

- `Web:` Only contains the Installer and configuration files, and it only downloads the MySQL products you choose to install. The size of this file is about 2MB; the name of the file has the form `mysql-installer-community-web-`*`VERSION`*`.`*`N`*`.msi` where `VERSION` is the MySQL Server version number such as 5.6 and `N` is the package number, which begins at 0.

Installer editions

- `Community edition:` Downloadable at http://dev.mysql.com/downloads/installer/. It installs the community edition of all MySQL products.

- `Commercial edition:` Downloadable at either My Oracle Support (MOS) or https://edelivery.oracle.com/. It installs the commercial version of all MySQL products, including Workbench SE. It also integrates with your MOS account, so enter in your MOS credentials to automatically receive updates for your commercial MySQL products.

For release notes detailing the changes in each release of MySQL Installer, see MySQL Installer Release Notes.

MySQL Installer is compatible with pre-existing installations, and adds them to its list of installed components. While the MySQL Installer is bundled with a specific version of MySQL Server, a single MySQL Installer instance can install and manage multiple MySQL Server versions. For example, a single MySQL Installer instance can install versions 5.1, 5.5, and 5.6. It can also manage either commercial or community editions of the MySQL Server.

> **Note**
>
> A single host can not have both community and commercial editions of MySQL Server installed. For example, if you want both MySQL Server 5.5 and 5.6 installed on a single host, then both must be the same commercial or community edition.

MySQL Installer handles the initial configuration and setup of the applications. For example:

1. It will create MySQL Server connections in MySQL Workbench.

2. It creates the configuration file (`my.ini`) that is used to configure the MySQL Server. The values written to this file are influenced by choices you make during the installation process.

3. It imports example databases.

4. It creates MySQL Server user accounts with configurable permissions based on general roles, such as DB Administrator, DB Designer, and Backup Admin. It optionally creates a Windows user named `MysqlSys` with limited privileges, which would then run the MySQL Server.

   This feature is only available during the initial installation of the MySQL Server, and not during future updates. User accounts may also be added with MySQL Workbench.

5. If the "Advanced Configuration" option is checked, then the **Logging Options** are also configured. This includes defining file paths for the error log, general log, slow query log (including the configuration of seconds it requires to execute a query), and the binary log.

MySQL Installer can optionally check for updated components and download them for you automatically.

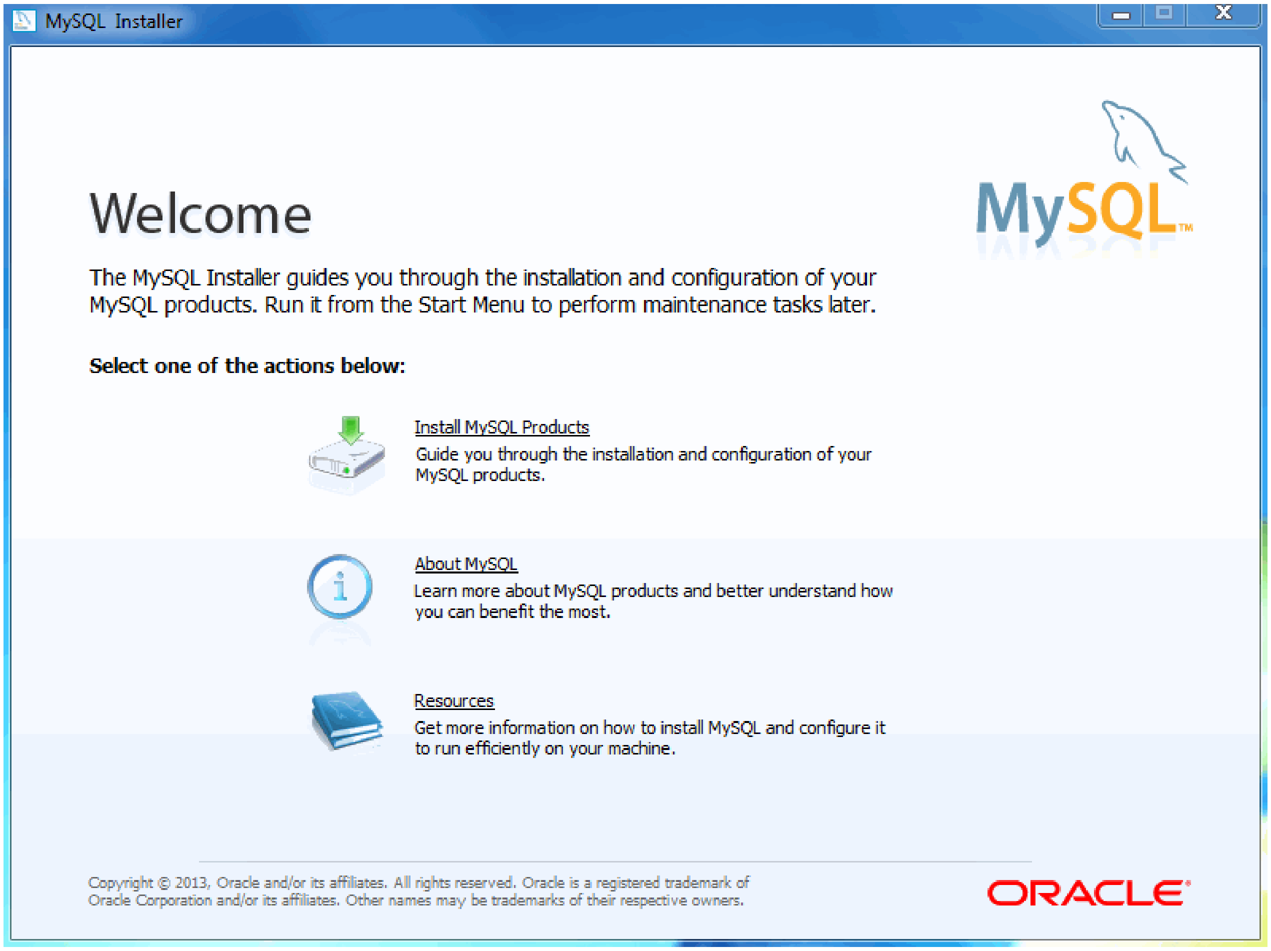
2.3.3.1 MySQL Installer GUI

After installation of the GUI version, the installer will have add its own Start Menu item under MySQL.

> **Note**
>
> Files that are generated by MySQL Installer grant full permissions to the user that executes MySQL Installer, including `my.ini`. This does not apply to files and directories for specific products such as the MySQL Server data directory in `ProgramData`, that is owned by `SYSTEM`.

After the installer itself has been installed and started, the following screen is displayed:

Figure 2.7 MySQL Installer - Welcome Screen



Welcome

The MySQL Installer guides you through the installation and configuration of your MySQL products. Run it from the Start Menu to perform maintenance tasks later.

Select one of the actions below:

Install MySQL Products
Guide you through the installation and configuration of your MySQL products.

About MySQL
Learn more about MySQL products and better understand how you can benefit the most.

Resources
Get more information on how to install MySQL and configure it to run efficiently on your machine.

Copyright © 2013, Oracle and/or its affiliates. All rights reserved. Oracle is a registered trademark of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

There are three main options:

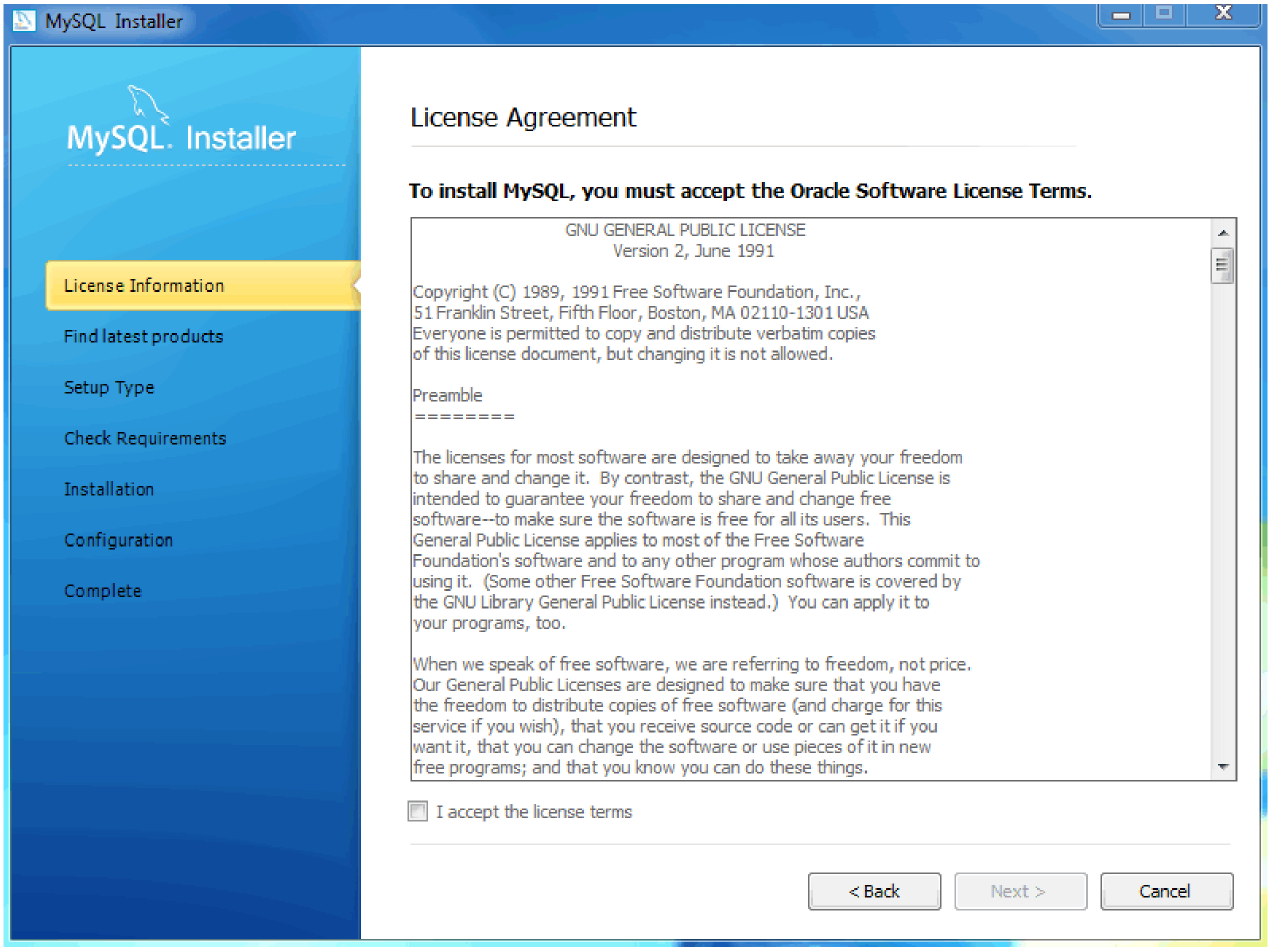
1. Install MySQL Products - The Installation Wizard.

2. About MySQL - Learn about MySQL products and features.

3. Resources - Information to help install and configure MySQL.

To **Install MySQL Products** after executing MySQL Installer for the first time, you must accept the license agreement before proceeding with the installation process.

Figure 2.8 MySQL Installer - License Agreement



License Agreement

To install MySQL, you must accept the Oracle Software License Terms.

GNU GENERAL PUBLIC LICENSE
Version 2, June 1991

Copyright (C) 1989, 1991 Free Software Foundation, Inc.,
51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA
Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.

Preamble
========

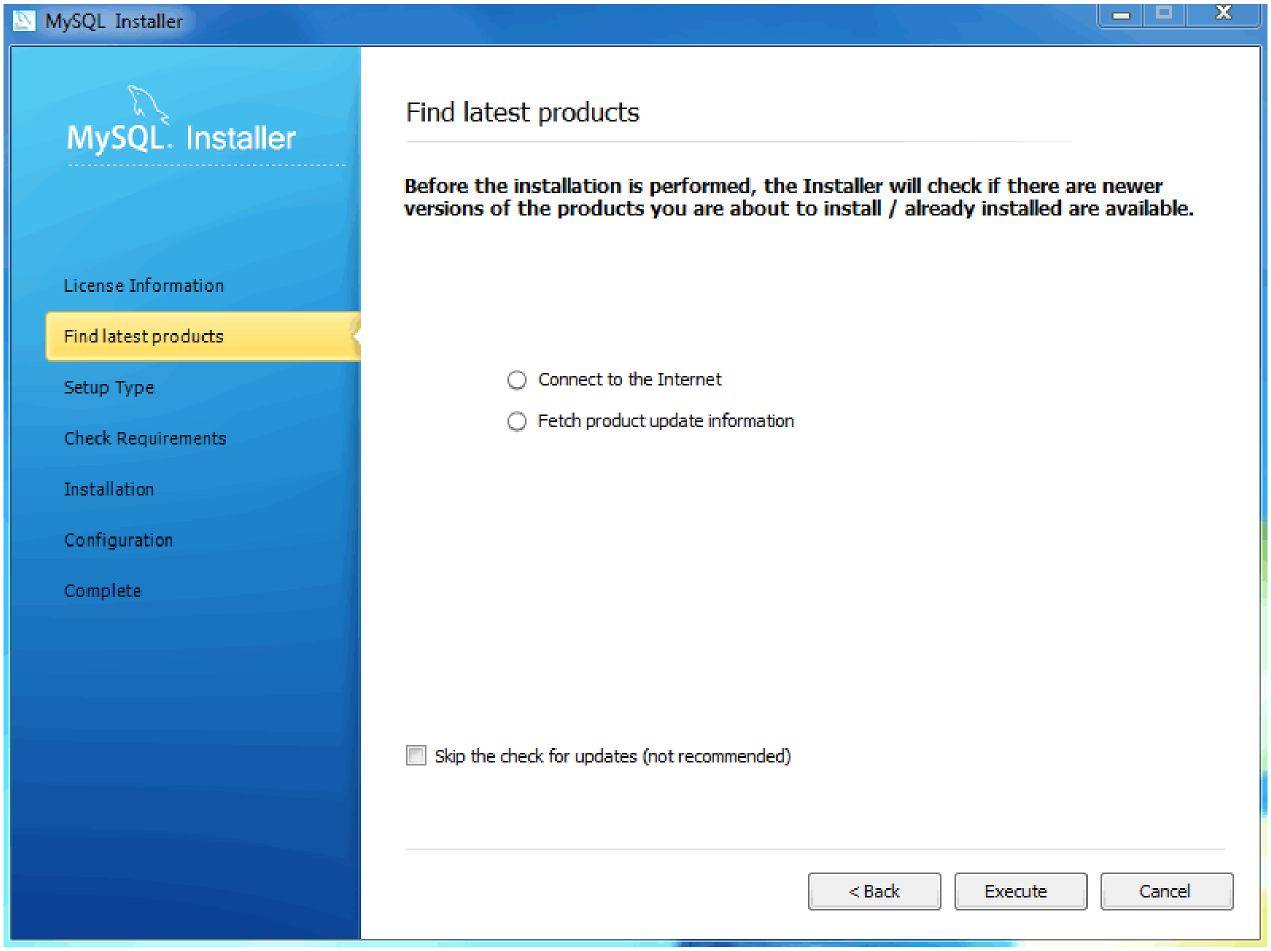
The licenses for most software are designed to take away your freedom
to share and change it. By contrast, the GNU General Public License is
intended to guarantee your freedom to share and change free
software--to make sure the software is free for all its users. This
General Public License applies to most of the Free Software
Foundation's software and to any other program whose authors commit to
using it. (Some other Free Software Foundation software is covered by
the GNU Library General Public License instead.) You can apply it to
your programs, too.

When we speak of free software, we are referring to freedom, not price.
Our General Public Licenses are designed to make sure that you have
the freedom to distribute copies of free software (and charge for this
service if you wish), that you receive source code or can get it if you
want it, that you can change the software or use pieces of it in new
free programs; and that you know you can do these things.

□ I accept the license terms

If you are connected to the Internet, then the Installer will search for the latest MySQL components and add them to the installation bundle. Click Connect to the Internet to complete this step, or otherwise check the `Skip` checkbox and then Continue.

Figure 2.9 MySQL Installer - Find latest products



Find latest products

Before the installation is performed, the Installer will check if there are newer versions of the products you are about to install / already installed are available.

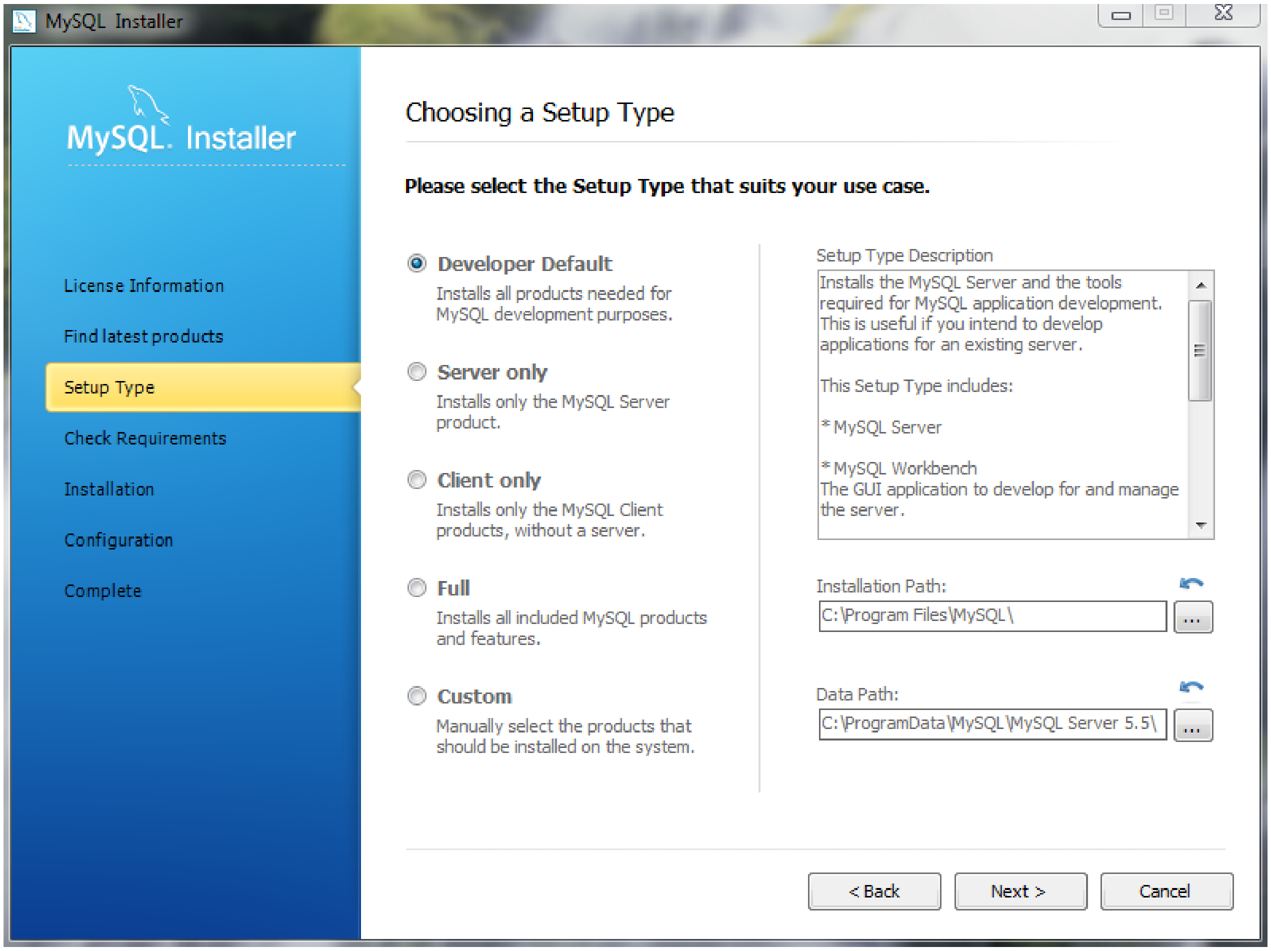
○ Connect to the Internet

○ Fetch product update information

□ Skip the check for updates (not recommended)

If you chose "Connect to the Internet," the next page will show the progress of MySQL Installer's search for available updates. When the search is complete (or if you opted to skip the search), you will be taken to the **Choose Setup Type** page:

Figure 2.10 MySQL Installer - Choosing a Setup Type



Choosing a Setup Type

Please select the Setup Type that suits your use case.

● **Developer Default**
Installs all products needed for MySQL development purposes.

○ **Server only**
Installs only the MySQL Server product.

○ **Client only**
Installs only the MySQL Client products, without a server.

○ **Full**
Installs all included MySQL products and features.

○ **Custom**
Manually select the products that should be installed on the system.

Setup Type Description

Installs the MySQL Server and the tools required for MySQL application development. This is useful if you intend to develop applications for an existing server.

This Setup Type includes:

* MySQL Server

* MySQL Workbench
The GUI application to develop for and manage the server.

Installation Path: C:\Program Files\MySQL\

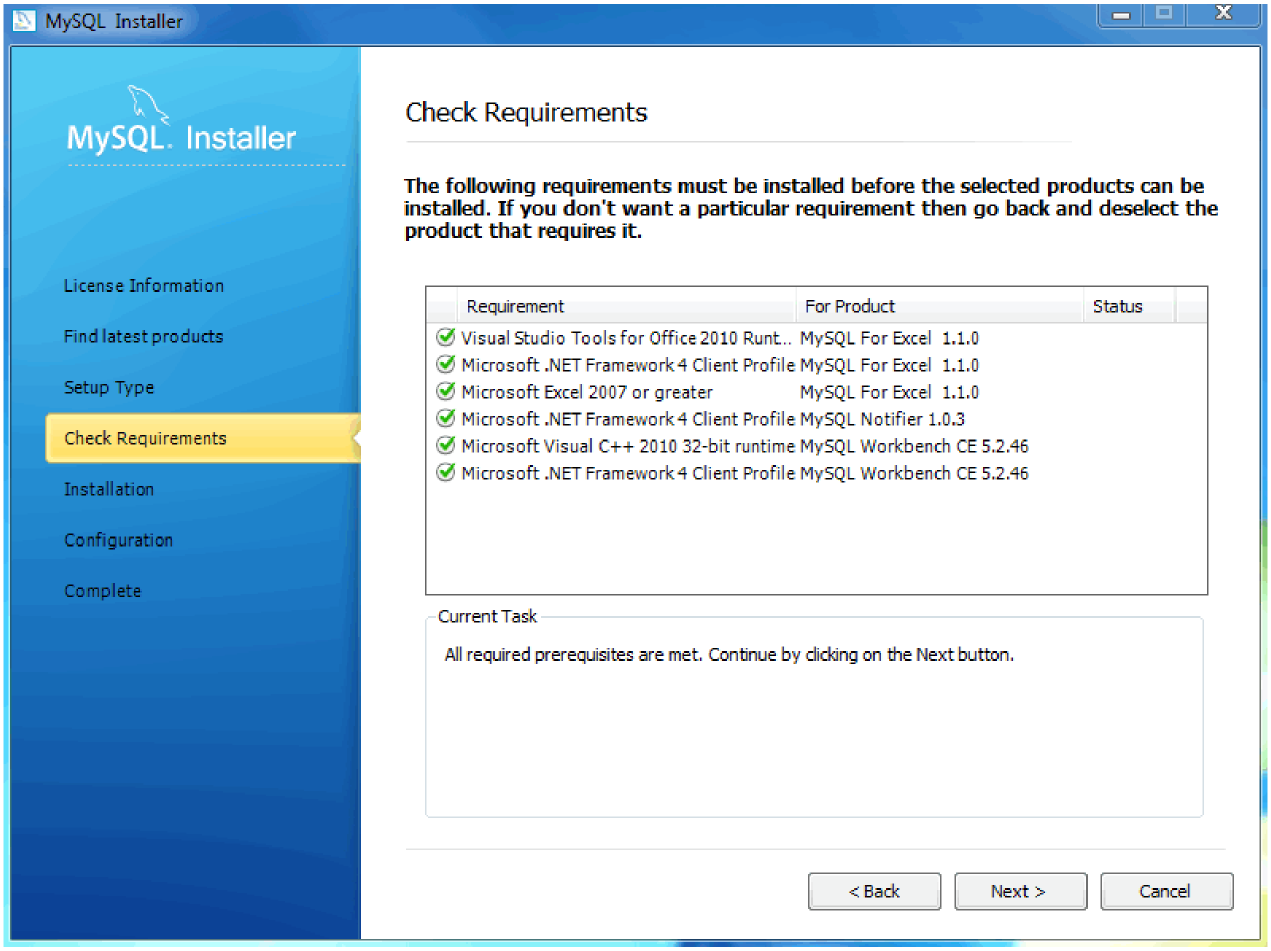
Data Path: C:\ProgramData\MySQL\MySQL Server 5.5\

Determine the option most compatible with your preferences by reading the **Setup Type Description** descriptions.

The Installation and Data paths are also defined here, and a caution flag will notify you if the data path you define already exists.

After you select a setup type, the MySQL Installer will check your system for the necessary external requirements and download then install missing components onto your system.

Figure 2.11 MySQL Installer - Check Requirements



Check Requirements

The following requirements must be installed before the selected products can be installed. If you don't want a particular requirement then go back and deselect the product that requires it.

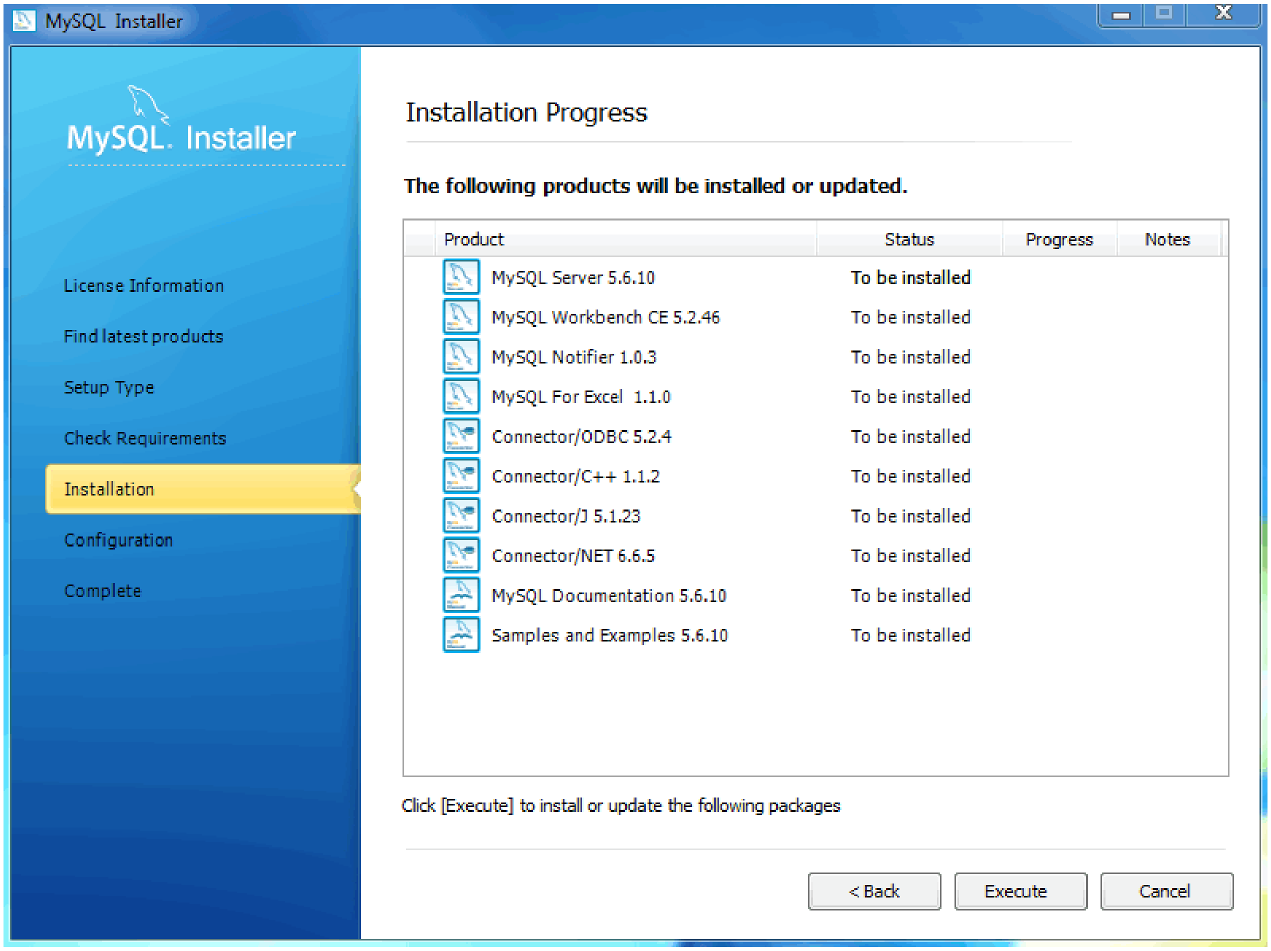
Requirement	For Product	Status
✓ Visual Studio Tools for Office 2010 Runt...	MySQL For Excel 1.1.0	
✓ Microsoft .NET Framework 4 Client Profile	MySQL For Excel 1.1.0	
✓ Microsoft Excel 2007 or greater	MySQL For Excel 1.1.0	
✓ Microsoft .NET Framework 4 Client Profile	MySQL Notifier 1.0.3	
✓ Microsoft Visual C++ 2010 32-bit runtime	MySQL Workbench CE 5.2.46	
✓ Microsoft .NET Framework 4 Client Profile	MySQL Workbench CE 5.2.46	

Current Task

All required prerequisites are met. Continue by clicking on the Next button.

The next window lists the MySQL products that are scheduled to be installed:

Figure 2.12 MySQL Installer - Installation Progress



Installation Progress

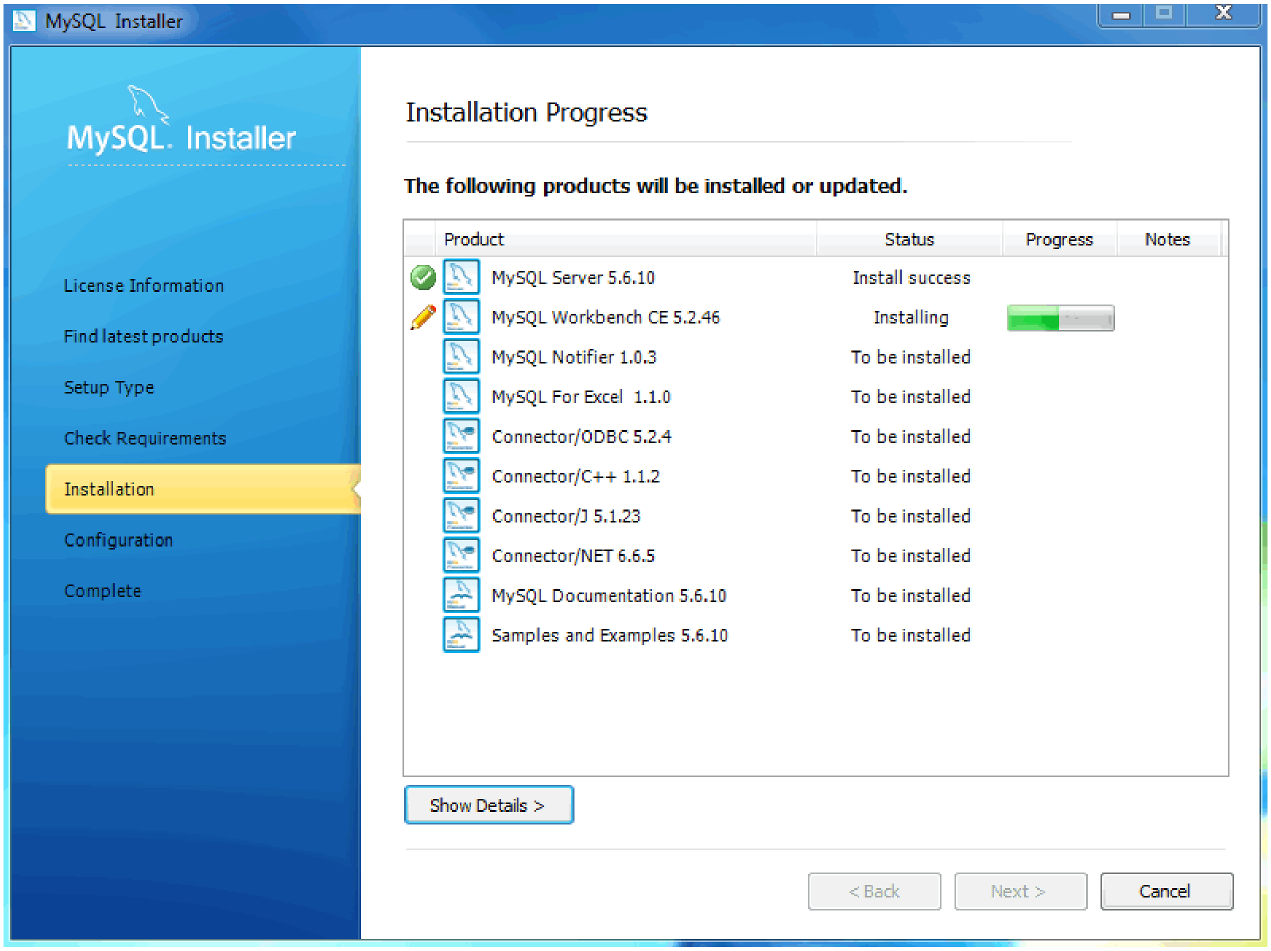
The following products will be installed or updated.

Product	Status	Progress	Notes
MySQL Server 5.6.10	To be installed		
MySQL Workbench CE 5.2.46	To be installed		
MySQL Notifier 1.0.3	To be installed		
MySQL For Excel 1.1.0	To be installed		
Connector/ODBC 5.2.4	To be installed		
Connector/C++ 1.1.2	To be installed		
Connector/J 5.1.23	To be installed		
Connector/NET 6.6.5	To be installed		
MySQL Documentation 5.6.10	To be installed		
Samples and Examples 5.6.10	To be installed		

Click [Execute] to install or update the following packages

As components are installed, you'll see their status change from "to be installed" to "install success."

Figure 2.13 MySQL Installer - Installation Progress status



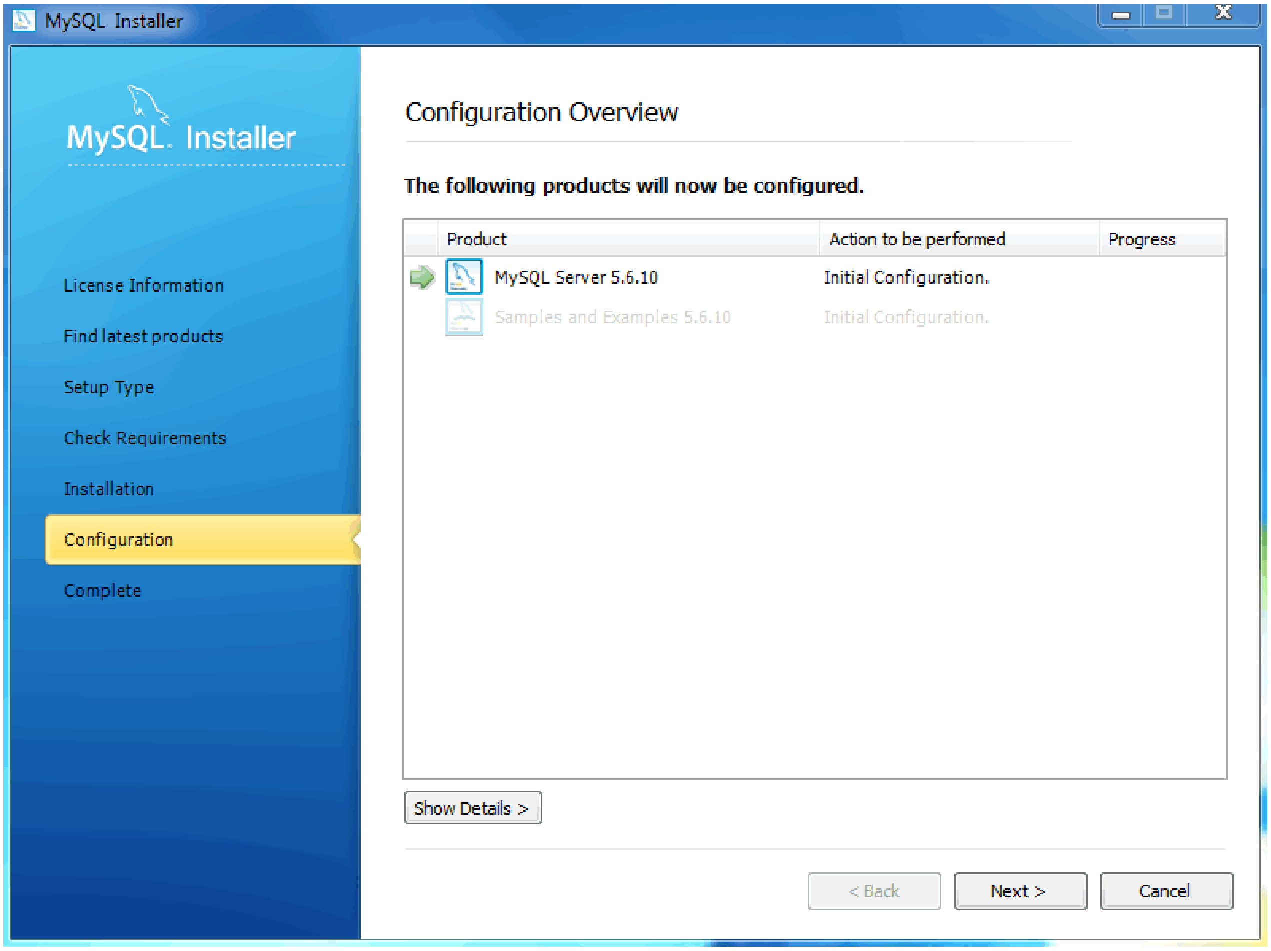
Installation Progress

The following products will be installed or updated.

	Product	Status	Progress	Notes
✓	MySQL Server 5.6.10	Install success		
✎	MySQL Workbench CE 5.2.46	Installing	🟩	
	MySQL Notifier 1.0.3	To be installed		
	MySQL For Excel 1.1.0	To be installed		
	Connector/ODBC 5.2.4	To be installed		
	Connector/C++ 1.1.2	To be installed		
	Connector/J 5.1.23	To be installed		
	Connector/NET 6.6.5	To be installed		
	MySQL Documentation 5.6.10	To be installed		
	Samples and Examples 5.6.10	To be installed		

After all components are installed, the next step involves configuring the products. The `Configuration Overview` window displays the progress and then loads a configuration window if it is required.

Figure 2.14 MySQL Installer - Configuration Overview



Configuration Overview

The following products will now be configured.

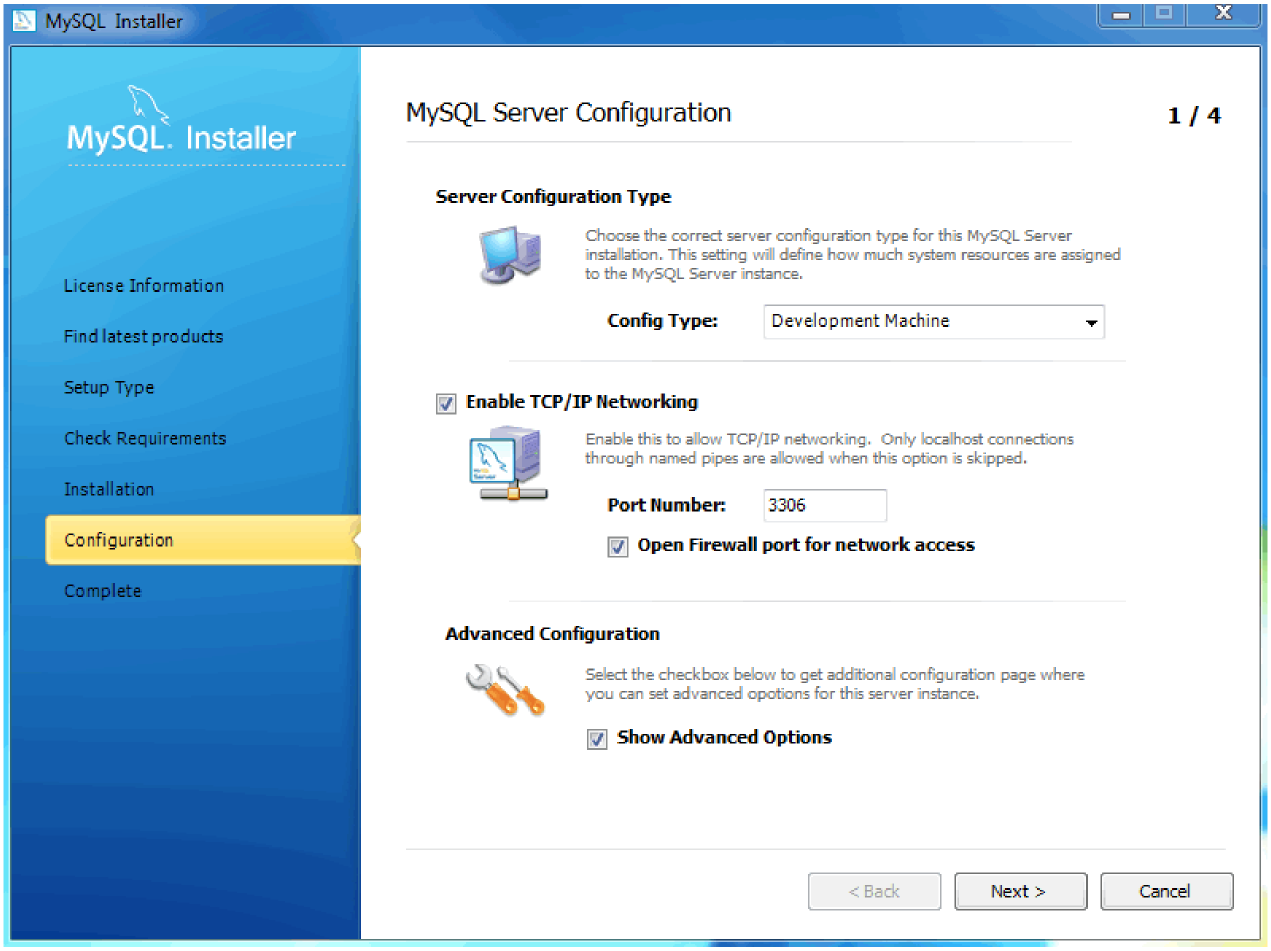
Product	Action to be performed	Progress
MySQL Server 5.6.10	Initial Configuration.	
Samples and Examples 5.6.10	Initial Configuration.	

The ideal MySQL Server configuration depends on your intended use, as explained in the next window. Choose the description that most closely applies to your machine.

You may enable TCP/IP Networking here as otherwise only localhost connections are allowed.

Checking the "Advanced Configuration" option provides additional **Logging Options** to configure. This includes defining file paths for the error log, general log, slow query log (including the configuration of seconds it requires to execute a query), and the binary log.

Figure 2.15 MySQL Installer - MySQL Server Configuration: Define platform, networking, and logging options



MySQL Server Configuration

1 / 4

Server Configuration Type

Choose the correct server configuration type for this MySQL Server installation. This setting will define how much system resources are assigned to the MySQL Server instance.

Config Type: Development Machine

☑ Enable TCP/IP Networking

Enable this to allow TCP/IP networking. Only localhost connections through named pipes are allowed when this option is skipped.

Port Number: 3306

☑ Open Firewall port for network access

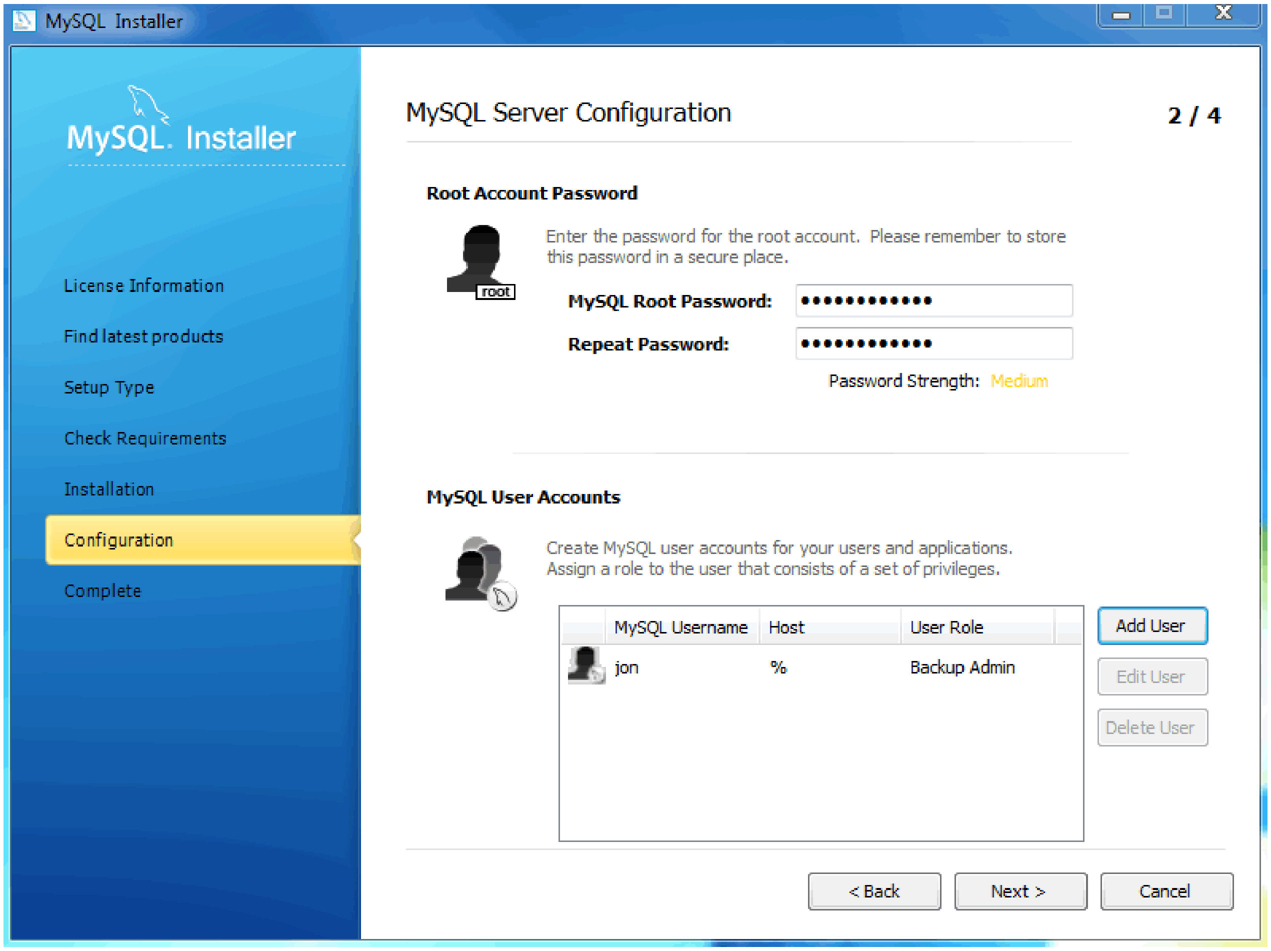
Advanced Configuration

Select the checkbox below to get additional configuration page where you can set advanced options for this server instance.

☑ Show Advanced Options

Next, choose your account information. Defining a root password is required, whereas it's optional to create additional users. There are several different predefined user roles that each have different permission levels. For example, a "DB Admin" will have more privileges than a "DB Designer.".

Figure 2.16 MySQL Installer - MySQL Server Configuration: User accounts



MySQL Server Configuration

2 / 4

Root Account Password

Enter the password for the root account. Please remember to store this password in a secure place.

MySQL Root Password: ••••••••••••

Repeat Password: ••••••••••••

Password Strength: Medium

MySQL User Accounts

Create MySQL user accounts for your users and applications. Assign a role to the user that consists of a set of privileges.

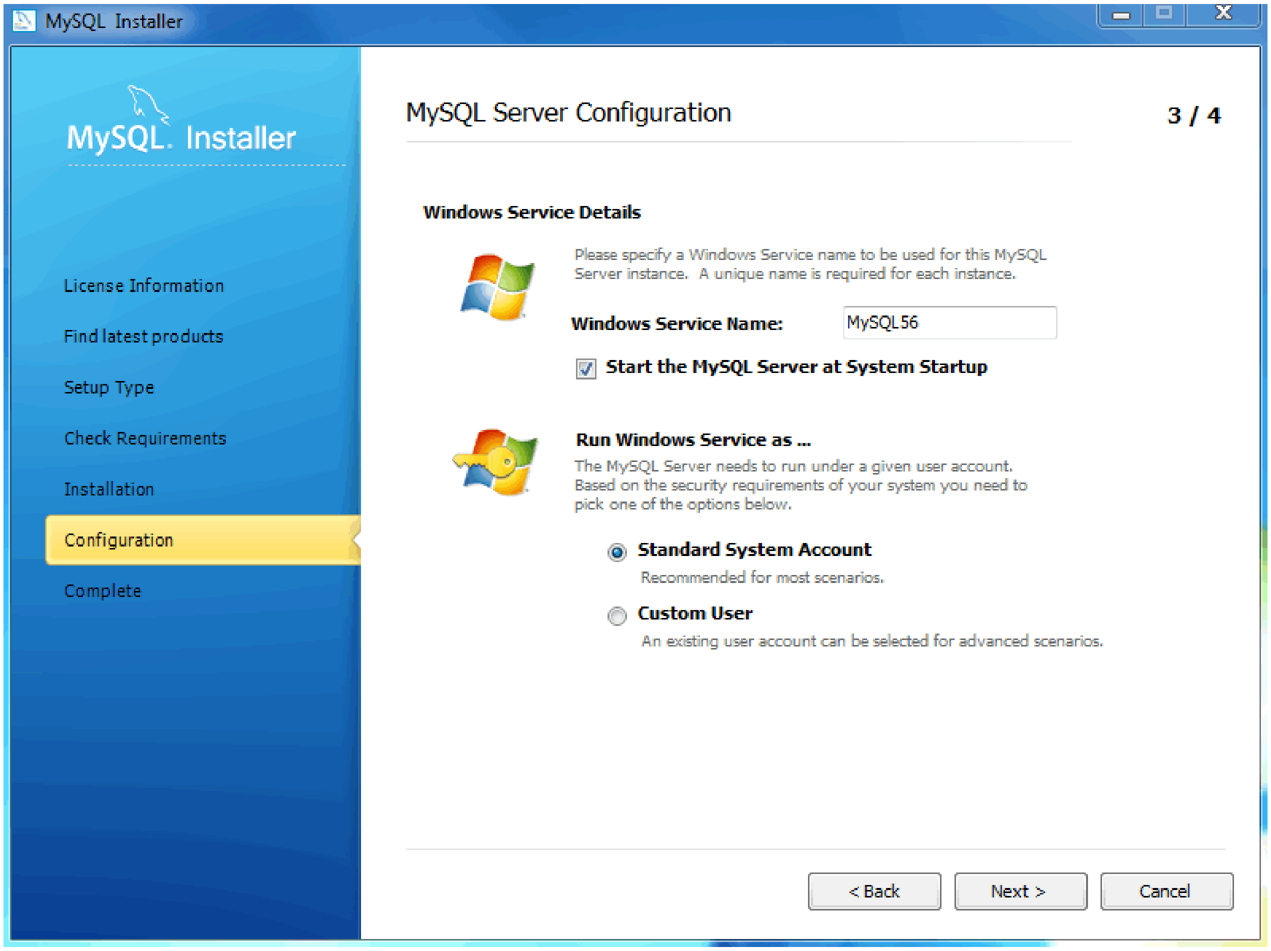
MySQL Username	Host	User Role
jon	%	Backup Admin

Note

If the MySQL Server is already installed, then the `Current Root Password` will also be needed.

Next, configure the **Windows Service Details**. This includes the service name, how the MySQL Server should be loaded at startup, and how the Windows Service for MySQL Server will be run.

Figure 2.17 MySQL Installer - MySQL Server Configuration: Windows service details



MySQL Server Configuration

Windows Service Details

Please specify a Windows Service name to be used for this MySQL Server instance. A unique name is required for each instance.

Windows Service Name: MySQL56

☑ Start the MySQL Server at System Startup

Run Windows Service as ...

The MySQL Server needs to run under a given user account. Based on the security requirements of your system you need to pick one of the options below.

● Standard System Account

Recommended for most scenarios.

○ Custom User

An existing user account can be selected for advanced scenarios.

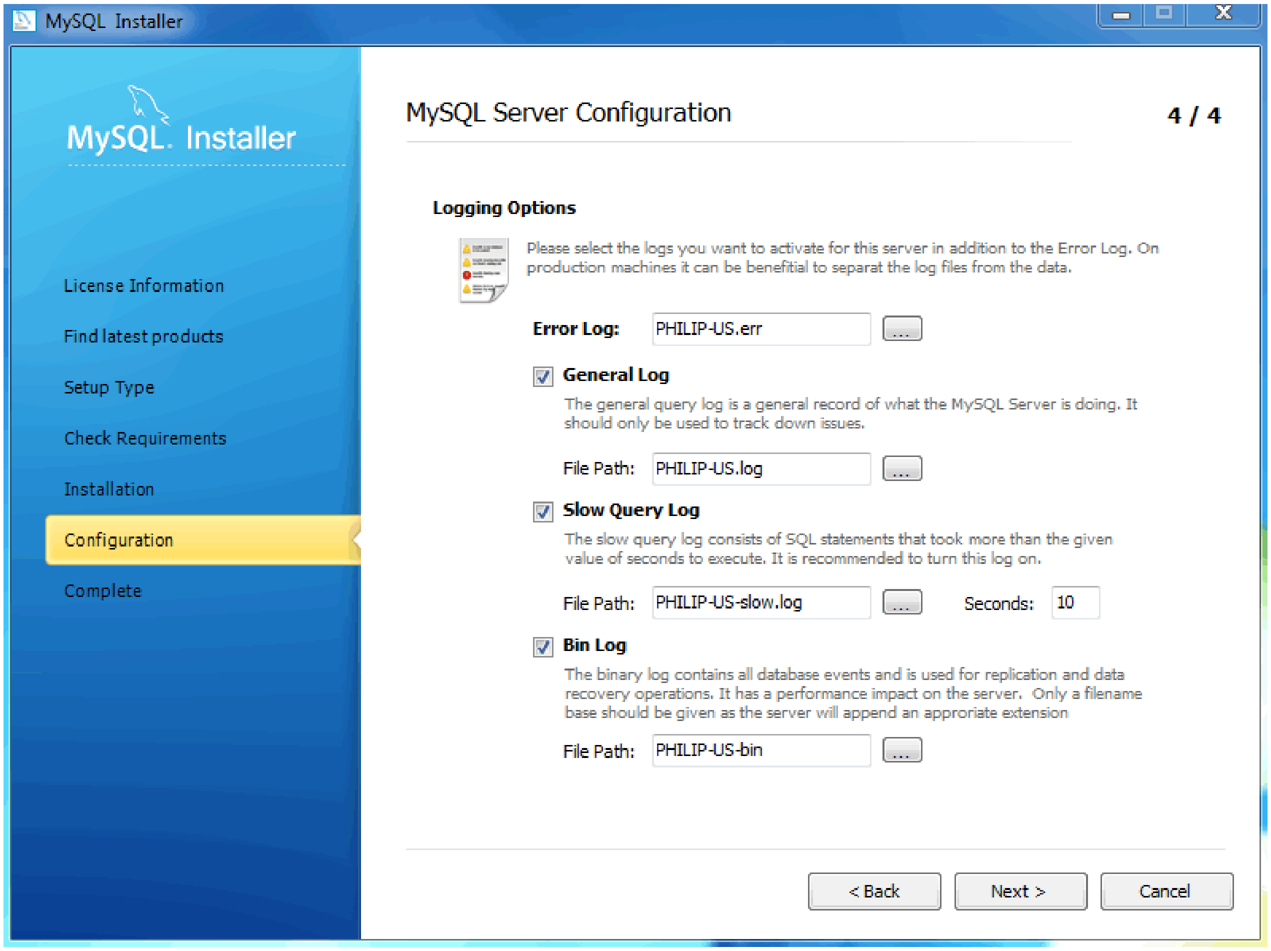
Note

When configuring **Run Windows Services as ...** using a **Custom User**, the custom user must have privileges to log on to Windows as a service. And the Next button will be disabled until this user is given these user rights.

On Microsoft Windows 7, this is configured by loading the `Start Menu`, `Control Panel`, `Administrative Tools`, `Local Security Policy`, `Local Policies`, `User Rights Assignment`, then `Log On As A Service`. Choose `Add User or Group` here to add the custom user, and then OK, OK to save.

The final configuration step is available if the **Advanced Configuration** option was checked, and it includes configuration options related to log file names:

Figure 2.18 MySQL Installer - MySQL Server Configuration: Logging options



Logging Options

Please select the logs you want to activate for this server in addition to the Error Log. On production machines it can be benefitial to separat the log files from the data.

Error Log: PHILIP-US.err

☑ General Log

The general query log is a general record of what the MySQL Server is doing. It should only be used to track down issues.

File Path: PHILIP-US.log

☑ Slow Query Log

The slow query log consists of SQL statements that took more than the given value of seconds to execute. It is recommended to turn this log on.

File Path: PHILIP-US-slow.log Seconds: 10

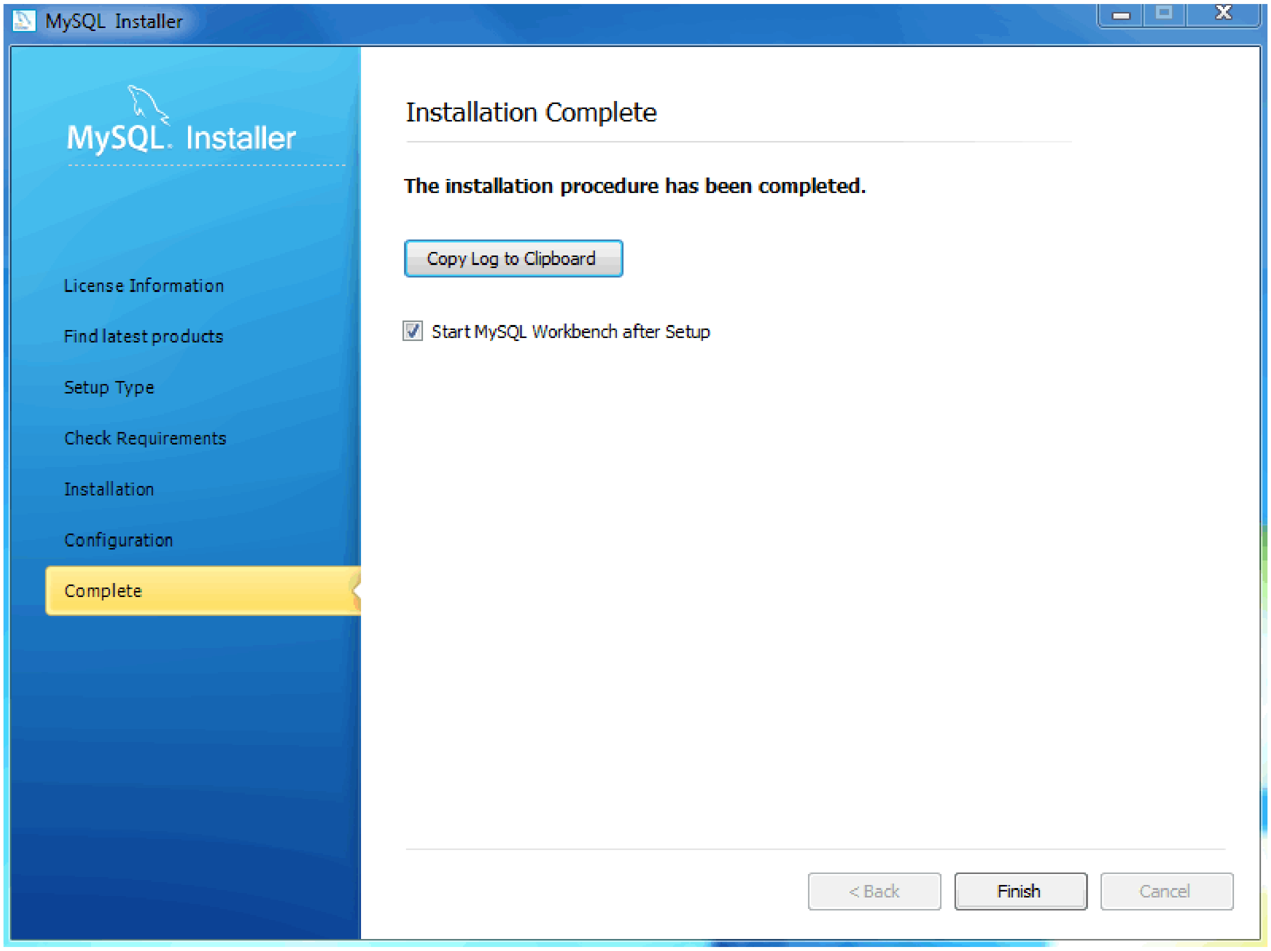
☑ Bin Log

The binary log contains all database events and is used for replication and data recovery operations. It has a performance impact on the server. Only a filename base should be given as the server will append an approriate extension

File Path: PHILIP-US-bin

After the MySQL Installer configuration process is completed, you may save the installation log, and then load MySQL Workbench if the **Start MySQL Workbench after Setup** option is checked:

Figure 2.19 MySQL Installer - Installation Complete



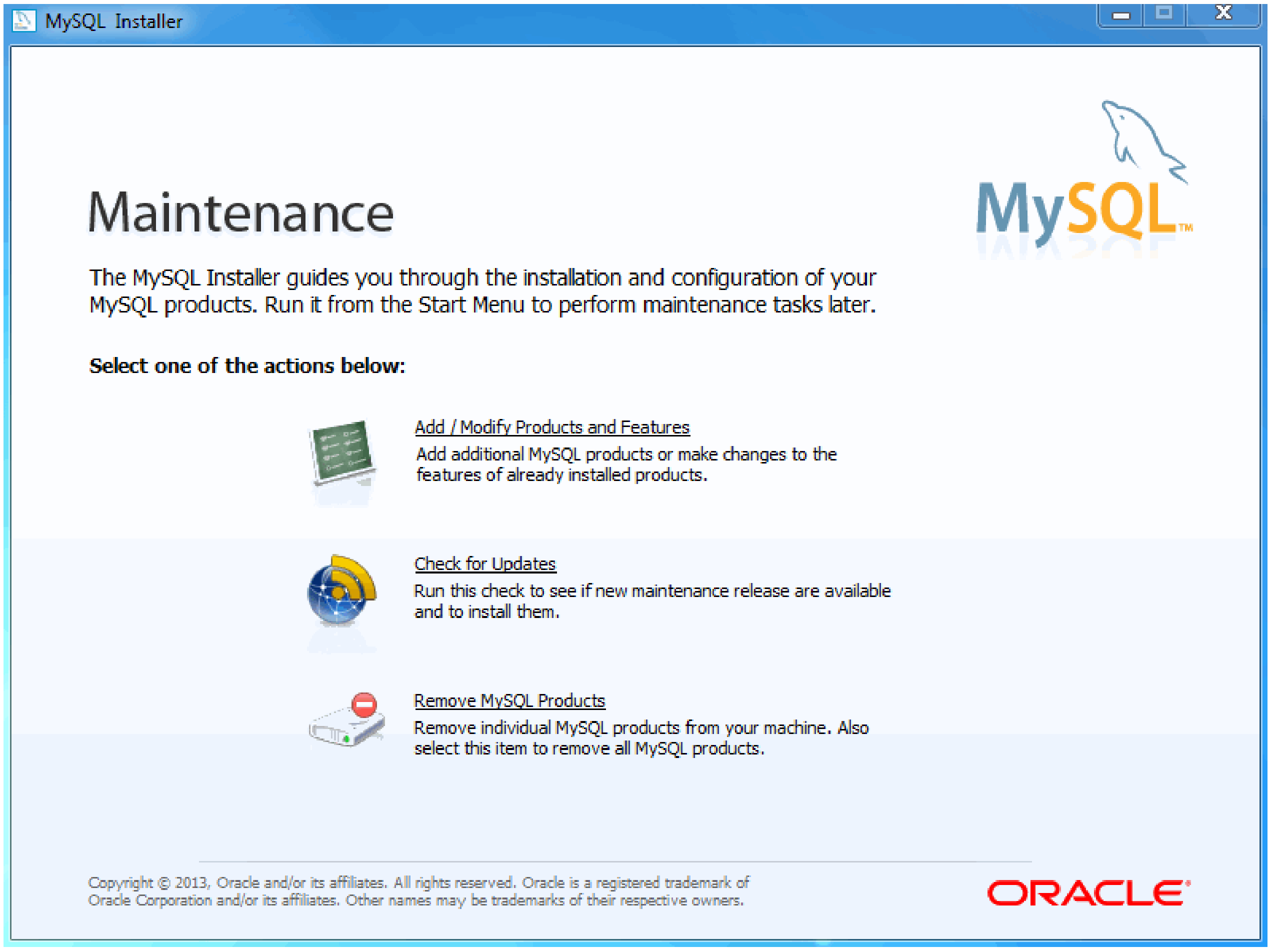
Installation Complete

The installation procedure has been completed.

☑ Start MySQL Workbench after Setup

You can now open MySQL Installer from the Microsoft Windows Start menu under the `MySQL` group, which will load the MySQL Installer `Maintenance Screen`. This is used to add, update, and remove features.

Figure 2.20 MySQL Installer - Maintenance Screen



Maintenance

The MySQL Installer guides you through the installation and configuration of your MySQL products. Run it from the Start Menu to perform maintenance tasks later.

Select one of the actions below:

Add / Modify Products and Features
Add additional MySQL products or make changes to the features of already installed products.

Check for Updates
Run this check to see if new maintenance release are available and to install them.

Remove MySQL Products
Remove individual MySQL products from your machine. Also select this item to remove all MySQL products.

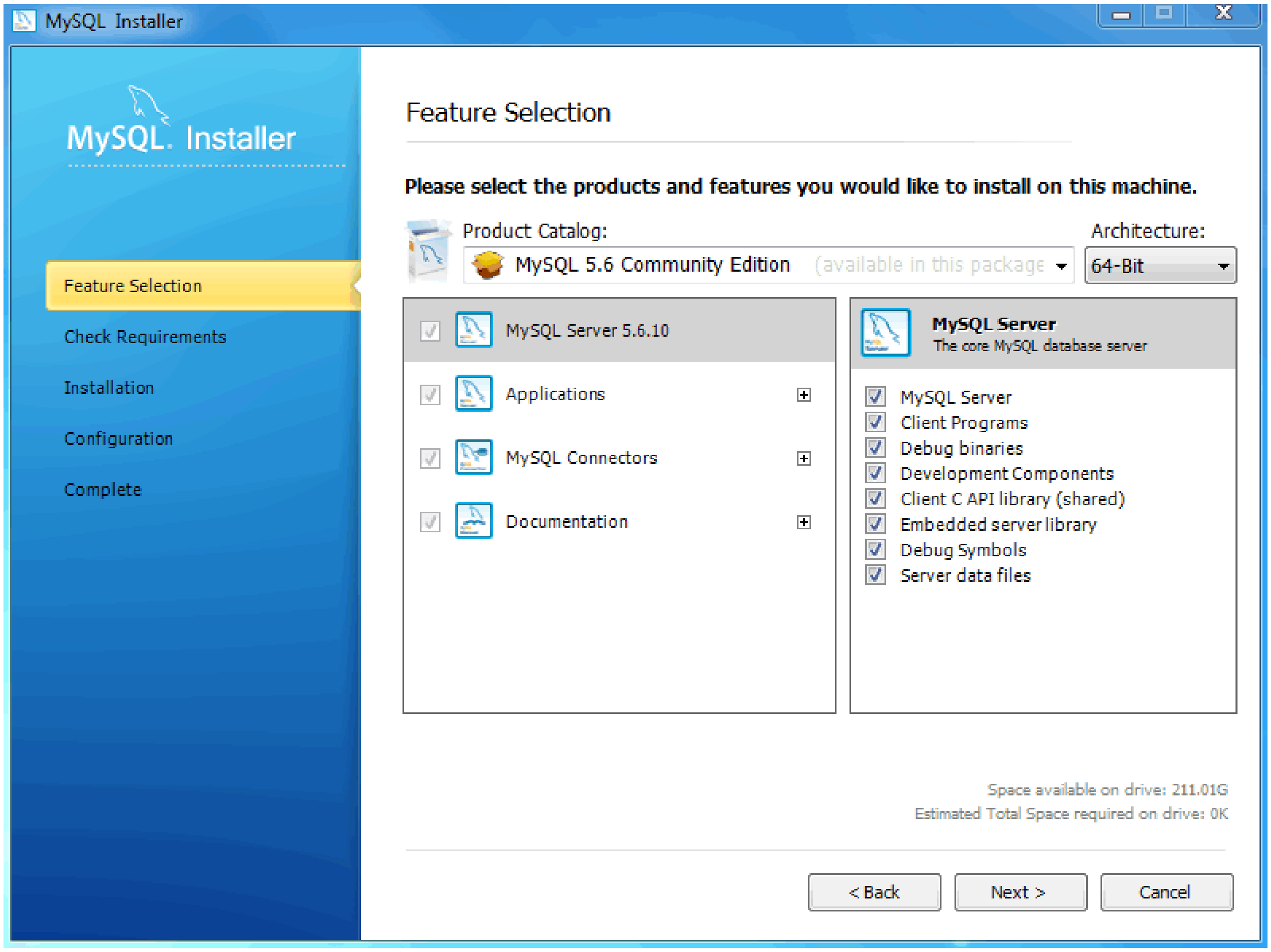
Copyright © 2013, Oracle and/or its affiliates. All rights reserved. Oracle is a registered trademark of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

Note

An `Update Screen` screen is shown if MySQL Installer is used on a machine with older products installed, as opposed to the `Maintenance Screen` shown above. However, the functionality remains the same.

Add/Modify Products and Features will list all installed and available MySQL products.

Figure 2.21 MySQL Installer - Add/Modify Products and Features



Feature Selection

Please select the products and features you would like to install on this machine.

Product Catalog: MySQL 5.6 Community Edition (available in this package)

Architecture: 64-Bit

☑ MySQL Server 5.6.10
☑ Applications
☑ MySQL Connectors
☑ Documentation

MySQL Server
The core MySQL database server

☑ MySQL Server
☑ Client Programs
☑ Debug binaries
☑ Development Components
☑ Client C API library (shared)
☑ Embedded server library
☑ Debug Symbols
☑ Server data files

Space available on drive: 211.01G
Estimated Total Space required on drive: 0K

The installation is now complete. MySQL Server should be running, and most MySQL products installed and available for use.

See also the MySQL Workbench documentation (http://dev.mysql.com/doc/workbench/en/).

2.3.3.2 MySQL Installer Console

`MySQLInstallerConsole` provides functionality similar to the GUI version of MySQL Installer, but from the command-line. It is installed when MySQL Installer is initially executed, and then available within the `MySQL Installer` directory. Typically that is in `C:\Program Files (x86)\MySQL\MySQL Installer\`, and the console must be executed with administrative privileges.

To use, invoke the Command Prompt with administrative privileges by choosing Start, Accessories, then right-click on Command Prompt and choose `Run as administrator`. And from the command-line, optionally change the directory to where `MySQLInstallerConsole` is located:

```
C:\> cd "C:\Program Files (x86)\MySQL\MySQL Installer"
```

`MySQLInstallerConsole` supports the following options, which are specified on the command line:

- `--help`, `-h`, or `-?`

Displays a help message with usage examples, and then exits.

```
C:\> MySQLInstallerConsole --help
```

- `--updates` (or `-u`)

  Checks for new products before any further action is taken. Disabled by default.

- `--nowait`

  Skips the final pause when the program finishes. Otherwise, a "`Press Enter to continue.`" dialogue is generated. It is used in conjunction with other options.

- `--catalog=`*`catalog_name`* (or `-c`)

  Sets the default catalog. Use `--list` to view a list of available catalogs.

- `--type=`*`installation_type`* (or `-t`)

  Sets the installation type.

  The possible values for `installation_type` are: developer, server, client, full, and custom.

- `--action=`*`action_name`*

  The action being performed.

  The possible values are: install, remove, upgrade, list, and status.

  - **install**: Installs a product or products, as defined by `--products`

  - **upgrade**: Upgrades a product or products, as defined by `--products`.

  - **remove**: Removes a product or products, as defined by `--products`.

  - **list**: Lists the product manifest, both installed and available products.

  - **status**: Shows the status after another action is performed.

- `--product=`*`product_name`*[*`:feature1`*],[*`feature2`*], [...] (or `-p`)

  Set the feature list of a product. Use `--list` to view available products, or pass in `--product=*` (an asterisk) to install all available products.

- `--config=`*`product_name`*:passwd=*`root_password`*[*`;parameter1=value`*], [;*`parameter2=value`*], ...

  The configuration parameters for the most recently listed products.

- `--user=`*`product_name`*:name=*`username`*,host:*`hostname`*,role=*`rolename`*,password=*`password`* or `--user=`*`product_name`*:name=*`username`*,host:*`hostname`*,role=*`rolename`*,tokens=*`tokens`*

  Creates a new user.

Requires: name, host, role, and the password or tokens. Tokens are separated by pipe ("|") characters.

2.3.4 MySQL Notifier for Microsoft Windows

The MySQL Notifier for Microsoft Windows is a tool that enables you to monitor and adjust the status of your local and remote MySQL Server instances through an indicator that resides in the system tray. The MySQL Notifier for Microsoft Windows also gives quick access to several MySQL GUI tools (such as MySQL Workbench) through its context menu.

The MySQL Notifier for Microsoft Windows is installed by MySQL Installer, and (by default) will start-up when Microsoft Windows is started.

Note

To install, download and execute the MySQL Installer, be sure the MySQL Notifier for Microsoft Windows product is selected, then proceed with the installation. See the MySQL Installer manual for additional details.

For release notes detailing the changes in each release of MySQL Notifier for Microsoft Windows, see the MySQL Notifier Release Notes.

Visit the MySQL Notifier forum for additional MySQL Notifier for Microsoft Windows help and support.

Features include:

- Start, Stop, and Restart instances of the MySQL Server.
- Automatically detects (and adds) new MySQL Server services. These are listed under Manage Monitored Items, and may also be configured.
- The Tray icon changes, depending on the status. It's green if all monitored MySQL Server instances are running, or red if at least one service is stopped. The **Update MySQL Notifier tray icon based on service status** option, which dictates this behavior, is enabled by default for each service.
- Links to other applications like MySQL Workbench, MySQL Installer, and the MySQL Utilities. For example, choosing Configure Instance will load the MySQL Workbench Server Administration window for that particular instance.
- If MySQL Workbench is also installed, then the Configure Instance and SQL Editor options are available for local (but not remote) MySQL instances.
- Monitoring of both local and remote MySQL instances.

Note

Remote monitoring is available since MySQL Notifier for Microsoft Windows 1.1.0.

The MySQL Notifier for Microsoft Windows resides in the system tray and provides visual status information for your MySQL Server instances. A green icon is displayed at the top left corner of the tray icon if the current MySQL Server is running, or a red icon if the service is stopped.

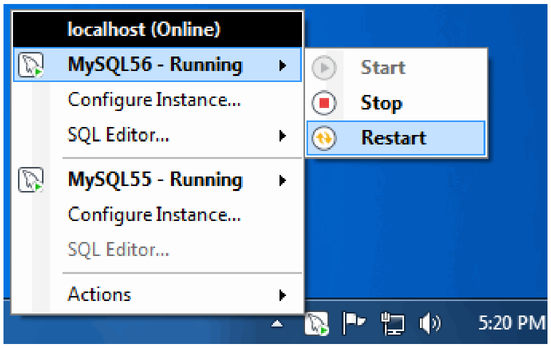
The MySQL Notifier for Microsoft Windows automatically adds discovered MySQL Services on the local machine, and each service is saved and configurable. By default, the **Automatically add new services whose name contains** option is enabled and set to `mysql`. Related **Notifications Options** include being notified when new services are either discovered or experience status changes, and are also enabled by default. And uninstalling a service will also remove the service from the MySQL Notifier for Microsoft Windows.

> **Note**
>
> The **Automatically add new services whose name contains** option default changed from ".*mysqld.*" to "mysql" in Notifier 1.1.0.

Clicking the system tray icon will reveal several options, as seen in the screenshots below:

The Service Instance menu is the main MySQL Notifier for Microsoft Windows window, and enables you to Stop, Start, and Restart the MySQL Server.

Figure 2.22 MySQL Notifier for Microsoft Windows Service Instance menu

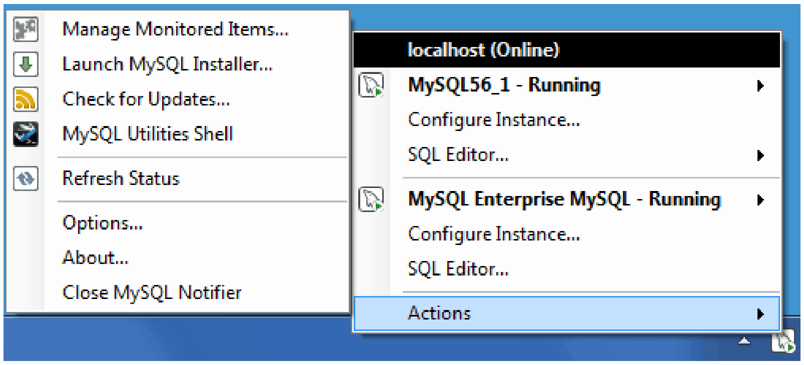


The Actions menu includes several links to external applications (if they are installed), and a a **Refresh Status** option to manually refresh the status of all monitored services (in both local and remote computers) and MySQL instances.

> **Note**
>
> The main menu will not show the Actions menu when there are no services being monitored by MySQL Notifier for Microsoft Windows.

> **Note**
>
> The **Refresh Status** feature is available since MySQL Notifier for Microsoft Windows 1.1.0.

Figure 2.23 MySQL Notifier for Microsoft Windows Actions menu

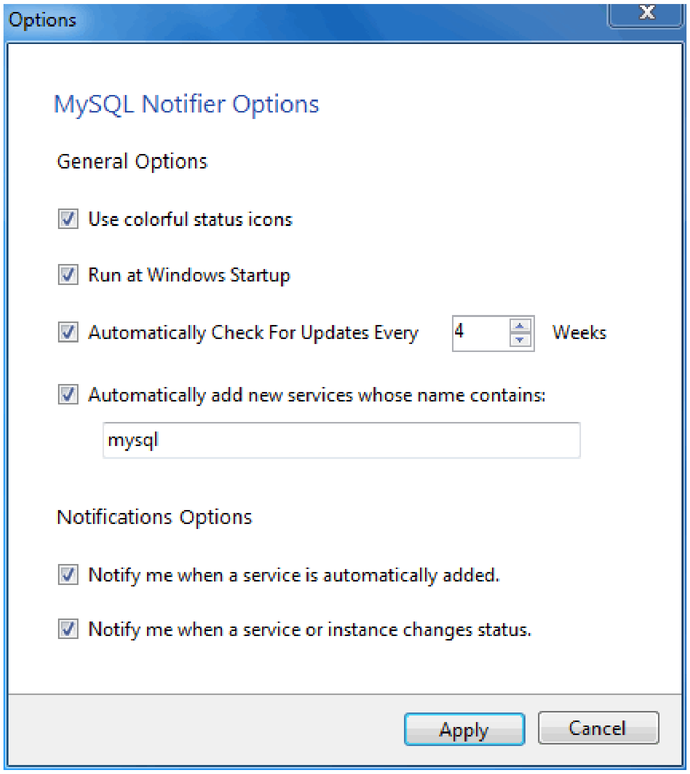


The Actions, Options menu configures MySQL Notifier for Microsoft Windows and includes options to:

- **Use colorful status icons**: Enables a colorful style of icons for the tray of the MySQL Notifier for Microsoft Windows.
- **Run at Windows Startup**: Allows the application to be loaded when Microsoft Windows starts.
- **Automatically Check For Updates Every** # Weeks: Checks for a new version of MySQL Notifier for Microsoft Windows, and runs this check every # weeks.
- **Automatically add new services whose name contains**: The text used to filter services and add them automatically to the monitored list of the local computer running MySQL Notifier, and on remote computers already monitoring Windows services. monitored services, and also filters the list of the Microsoft Windows services for the **Add New Service** dialog.

  Prior to version 1.1.0, this option was named "Automatically add new services that match this pattern."
- **Notify me when a service is automatically added**: Will display a balloon notification from the taskbar when a newly discovered service is added to the monitored services list.
- **Notify me when a service changes status**: Will display a balloon notification from the taskbar when a monitored service changes its status.

Figure 2.24 MySQL Notifier for Microsoft Windows Options menu



Options

MySQL Notifier Options

General Options

☑ Use colorful status icons

☑ Run at Windows Startup

☑ Automatically Check For Updates Every 4 Weeks

☑ Automatically add new services whose name contains:

mysql

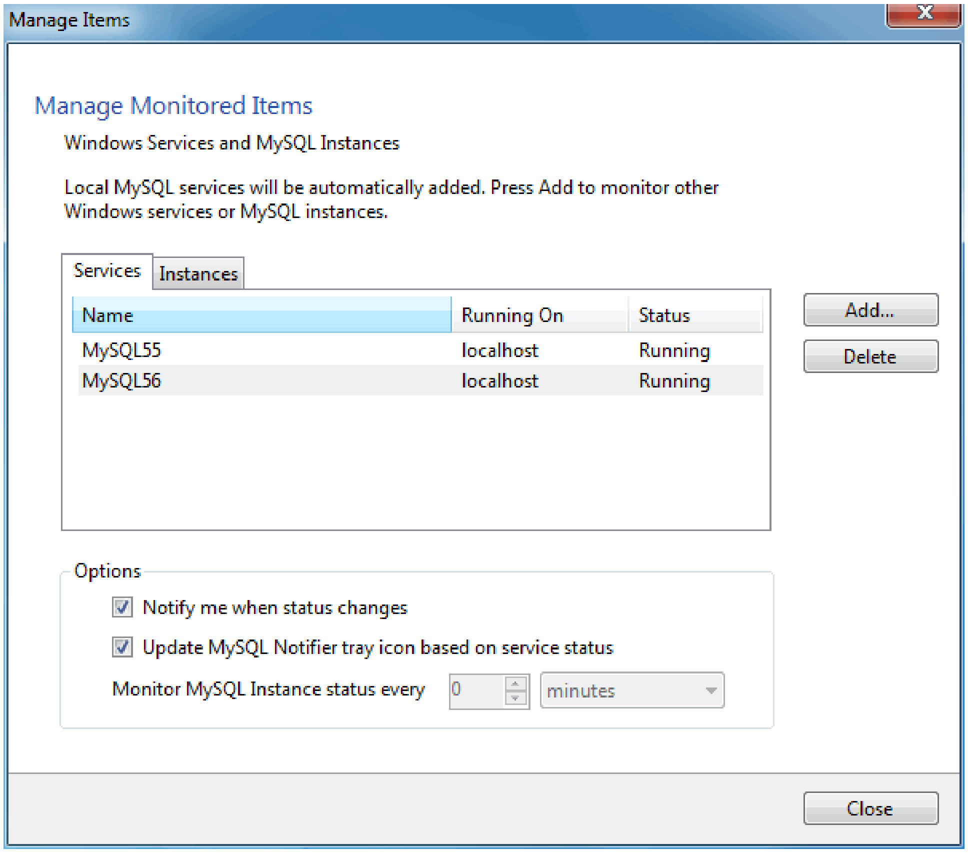
Notifications Options

☑ Notify me when a service is automatically added.

☑ Notify me when a service or instance changes status.

The Actions, Manage Monitored Items menu enables you to configure the monitored services and MySQL instances. First, with the **Services** tab open:

Figure 2.25 MySQL Notifier for Microsoft Windows Manage Services menu



Manage Monitored Items

Windows Services and MySQL Instances

Local MySQL services will be automatically added. Press Add to monitor other Windows services or MySQL instances.

Name	Running On	Status
MySQL55	localhost	Running
MySQL56	localhost	Running

Options

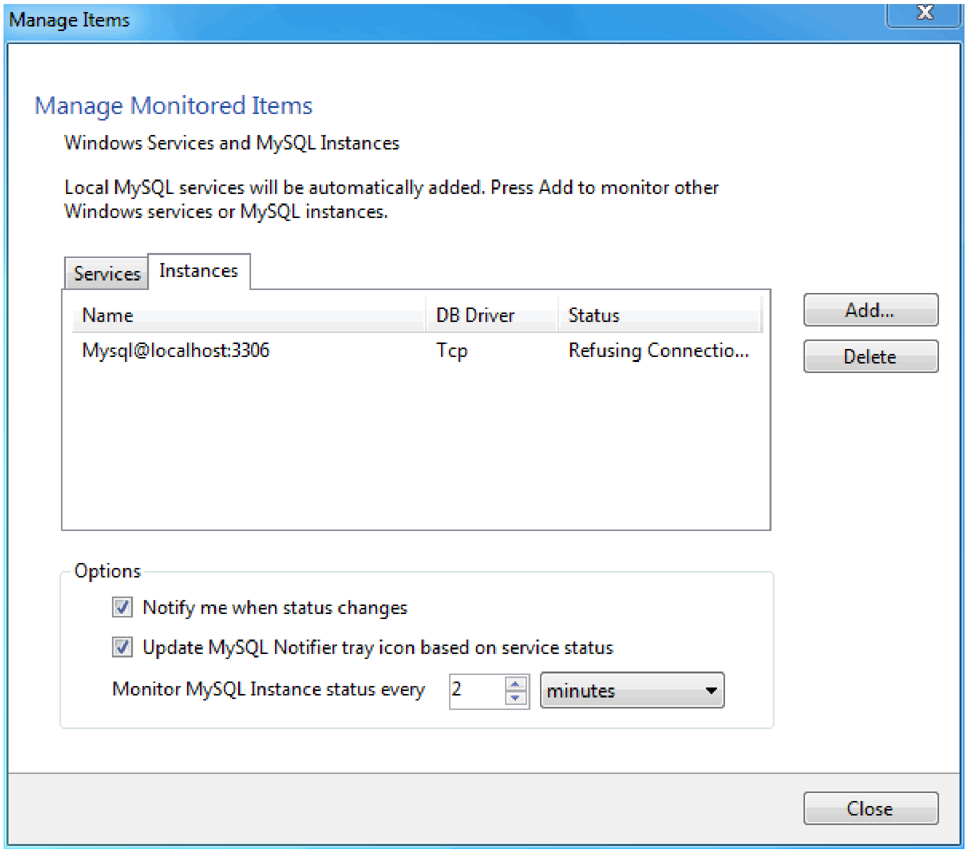
☑ Notify me when status changes

☑ Update MySQL Notifier tray icon based on service status

Monitor MySQL Instance status every 0 minutes

The **Instances** tab is similar:

Figure 2.26 MySQL Notifier for Microsoft Windows Manage Instances menu



Manage Monitored Items

Windows Services and MySQL Instances

Local MySQL services will be automatically added. Press Add to monitor other Windows services or MySQL instances.

Name	DB Driver	Status
Mysql@localhost:3306	Tcp	Refusing Connectio...

Options

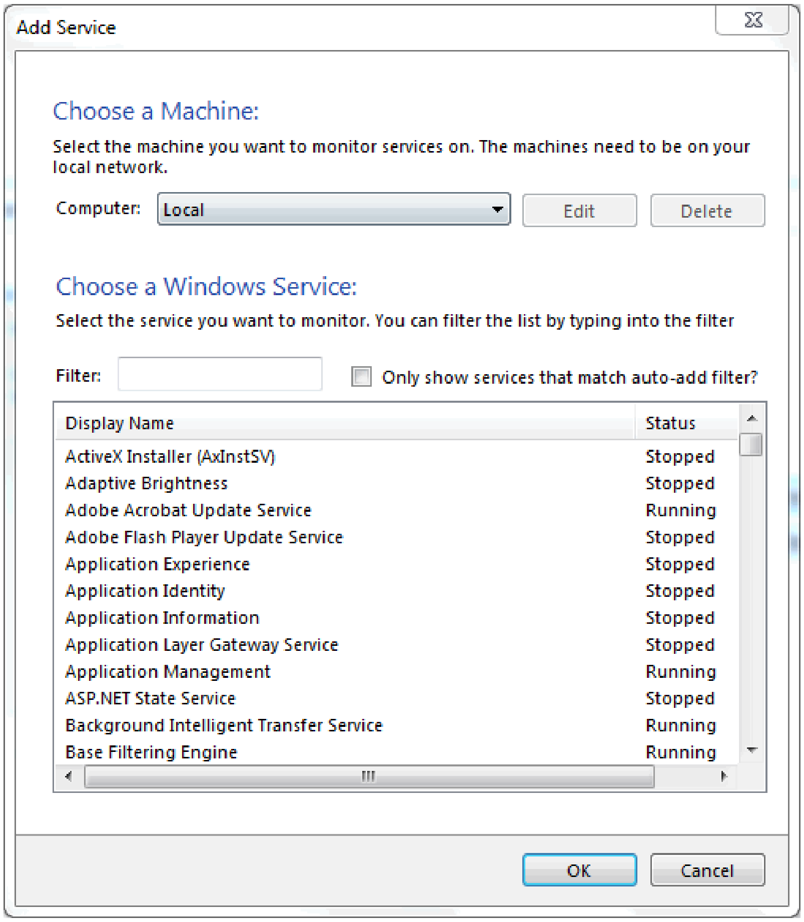
☑ Notify me when status changes

☑ Update MySQL Notifier tray icon based on service status

Monitor MySQL Instance status every 2 minutes

Adding a service or instance (after clicking Add in the Manage Monitored Items window) enables you to select a running Microsoft Windows service or instance connection, and configure MySQL Notifier for Microsoft Windows to monitor it. Add a new service or instance by clicking service name from the list, then OK to accept. Multiple services and instances may be selected.

Figure 2.27 MySQL Notifier for Microsoft Windows Adding new services



Select the machine you want to monitor services on. The machines need to be on your local network.

Computer: Local

Select the service you want to monitor. You can filter the list by typing into the filter

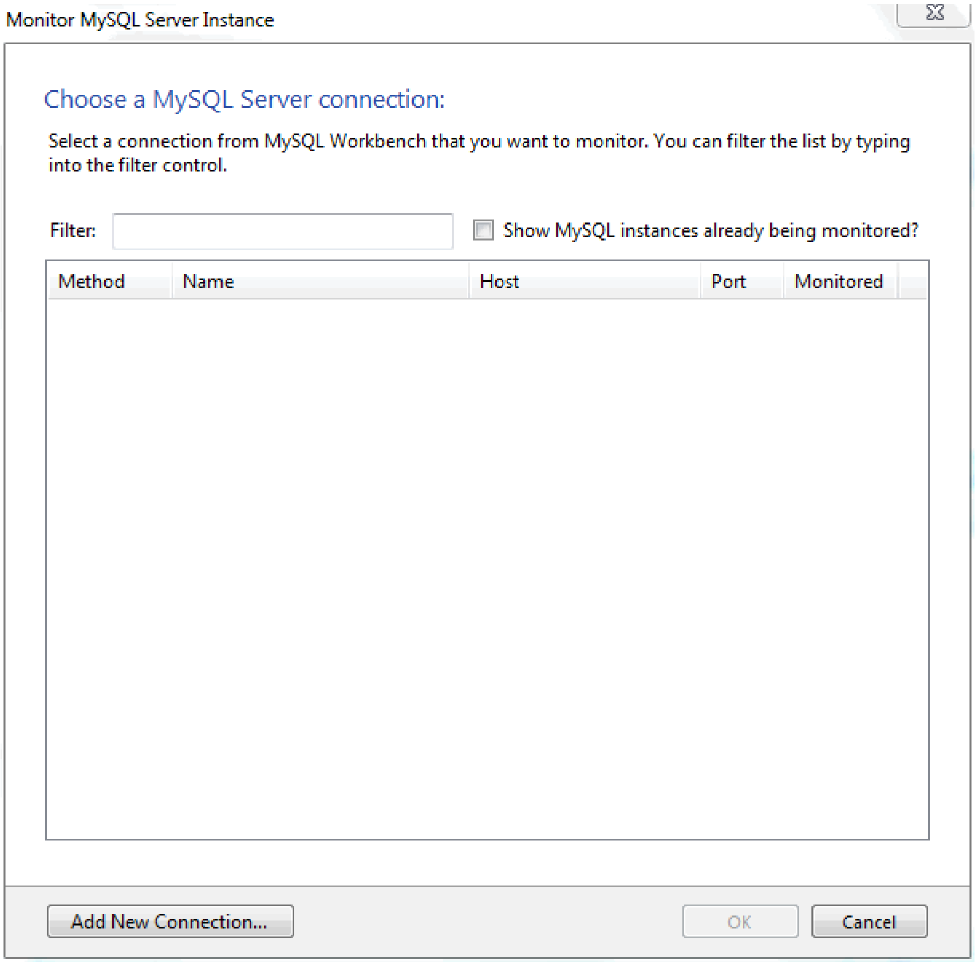
Filter:

□ Only show services that match auto-add filter?

Display Name	Status
ActiveX Installer (AxInstSV)	Stopped
Adaptive Brightness	Stopped
Adobe Acrobat Update Service	Running
Adobe Flash Player Update Service	Stopped
Application Experience	Stopped
Application Identity	Stopped
Application Information	Stopped
Application Layer Gateway Service	Stopped
Application Management	Running
ASP.NET State Service	Stopped
Background Intelligent Transfer Service	Running
Base Filtering Engine	Running

And instances:

Figure 2.28 MySQL Notifier for Microsoft Windows Adding new instances



Monitor MySQL Server Instance

Choose a MySQL Server connection:

Select a connection from MySQL Workbench that you want to monitor. You can filter the list by typing into the filter control.

Filter:

□ Show MySQL instances already being monitored?

Method	Name	Host	Port	Monitored

Note

The **Instances** tab available since MySQL Notifier for Microsoft Windows 1.1.0.

2.3.4.1 Remote monitoring set up and installation instructions

The MySQL Notifier for Microsoft Windows uses Windows Management Instrumentation (WMI) to manage and monitor services in remote computers running Windows XP or later. This guide explains how it works, and how to set up your system to monitor remote MySQL instances.

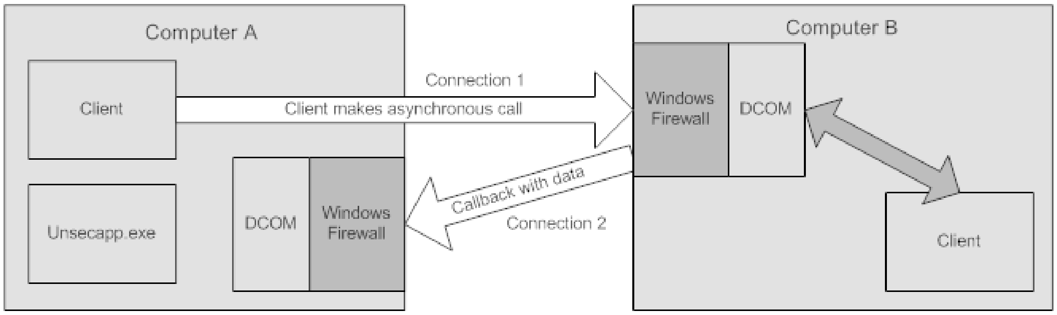
Note

Remote monitoring is available since MySQL Notifier for Microsoft Windows 1.1.0.

In order to configure WMI, it is important to understand that the underlying Distributed Component Object Model (DCOM) architecture is doing the WMI work. Specifically, MySQL Notifier for Microsoft Windows is using asynchronous notification queries on remote Microsoft Windows hosts as .NET events. These events send an asynchronous callback to the computer running the MySQL Notifier for Microsoft Windows so it knows when a service status has changed on the remote computer. Asynchronous notifications offer the best performance compared to semi-synchronous notifications or synchronous notifications that use timers.

Asynchronous notifications requires the remote computer to send a callback to the client computer (thus opening a reverse connection), so the Windows Firewall and DCOM settings must be properly configured for the communication to function properly.

Figure 2.29 MySQL Notifier for Microsoft Windows Distributed Component Object Model (DCOM)



Most of the common errors thrown by asynchronous WMI notifications are related to Windows Firewall blocking the communication, or to DCOM / WMI settings not being set up properly. For a list of common errors with solutions, see Common Errors.

The following steps are required to make WMI function. These steps are divided between two machines. A single host computer that runs MySQL Notifier for Microsoft Windows (Computer A), and multiple remote machines that are being monitored (Computer B).

Computer running MySQL Notifier for Microsoft Windows (Computer A)

1. Allow for remote administration by either editing the **Group Policy Editor**, or using `NETSH`:

   Using the **Group Policy Editor**:

   a. Click Start, click Run, type `GPEDIT.MSC`, and then click OK.

   b. Under the **Local Computer Policy** heading, double-click **Computer Configuration**.

   c. Double-click **Administrative Templates**, then **Network**, **Network Connections**, and then **Windows Firewall**.

   d. If the computer is in the domain, then double-click **Domain Profile**; otherwise, double-click **Standard Profile**.

   e. Click **Windows Firewall: Allow inbound remote administration exception**.

   f. On the Action menu either select Edit, or double-click the selection from the previous step.

   g. Check the Enabled radio button, and then click OK.

   Using the `NETSH` command:

   a. Open a command prompt window with Administrative rights (you can right-click the Command Prompt icon and click **Run as Administrator**).

   b. Execute the following command:

   ```
   NETSH firewall set service RemoteAdmin enable
   ```

2. Open the DCOM port TCP 135:

a. Open a command prompt window with Administrative rights (you can right-click the Command Prompt icon and click **Run as Administrator**) .

b. Execute the following command:

```
NETSH firewall add portopening protocol=tcp port=135 name=DCOM_TCP135
```

3. Add the client application which contains the sink for the callback (`MySqlNotifier.exe`) to the Windows Firewall Exceptions List (use either the Windows Firewall configuration or `NETSH`):

Using the Windows Firewall configuration:

a. In the Control Panel, double-click **Windows Firewall**.

b. In the Windows Firewall window's left panel, click **Allow a program or feature through Windows Firewall**.

c. In the Allowed Programs window, click Change Settings.

d. If `MySqlNotifier.exe` is in the Allowed programs and features list, make sure it is checked for the type of networks the computer connects to (Private, Public or both).

e. If `MySqlNotifier.exe` is not in the list, click Allow another program....

f. In the **Add a Program** window, select the `MySqlNotifier.exe` if it exists in the Programs list, otherwise click Browse... and go to the directory where `MySqlNotifier.exe` was installed to select it, then click Add.

g. Make sure `MySqlNotifier.exe` is checked for the type of networks the computer connects to (Private, Public or both).

Using the `NETSH` command:

a. Open a command prompt window with Administrative rights (you can right-click the Command Prompt icon and click Run as Administrator).

b. Execute the following command, where you change "*`[YOUR_INSTALL_DIRECTORY]`*":

```
NETSH firewall add allowedprogram program=[YOUR_INSTALL_DIRECTORY]\MySqlNotifier.exe name=MySqlNotif
```

4. If Computer B is either a member of `WORKGROUP` or is in a different domain that is untrusted by Computer A, then the callback connection (Connection 2) is created as an Anonymous connection. To grant Anonymous connections DCOM Remote Access permissions:

a. Click Start, click Run, type `DCOMCNFG`, and then click OK.

b. In the Component Services dialog box, expand Component Services, expand Computers, and then right-click **My Computer** and click Properties.

c. In the My Computer Properties dialog box, click the **COM Security** tab.

d. Under Access Permissions, click Edit Limits.

e. In the Access Permission dialog box, select **ANONYMOUS LOGON name** in the Group or user names box. In the Allow column under Permissions for User, select **Remote Access**, and then click OK.

Monitored Remote Computer (Computer B)

If the user account that is logged into the computer running the MySQL Notifier for Microsoft Windows (Computer A) is a local administrator on the remote computer (Computer B), such that the same account is an administrator on Computer B, you can skip to the "Allow for remote administration" step.

Setting DCOM security to allow a non-administrator user to access a computer remotely:

1. Grant "DCOM remote launch" and activation permissions for a user or group:

   a. Click Start, click Run, type `DCOMCNFG`, and then click OK.

   b. In the Component Services dialog box, expand Component Services, expand Computers, and then right-click **My Computer** and click Properties.

   c. In the My Computer Properties dialog box, click the **COM Security** tab.

   d. Under Access Permissions, click Edit Limits.

   e. In the **Launch Permission** dialog box, follow these steps if your name or your group does not appear in the Groups or user names list:

      i. In the **Launch Permission** dialog box, click Add.

      ii. In the Select Users, Computers, or Groups dialog box, add your name and the group in the "Enter the object names to select" box, and then click OK.

   f. In the **Launch Permission** dialog box, select your user and group in the Group or user names box. In the Allow column under Permissions for User, select **Remote Launch**, select **Remote Activation**, and then click OK.

   Grant DCOM remote access permissions:

   a. Click Start, click Run, type `DCOMCNFG`, and then click OK.

   b. In the Component Services dialog box, expand Component Services, expand Computers, and then right-click **My Computer** and click Properties.

   c. In the My Computer Properties dialog box, click the **COM Security** tab.

   d. Under Access Permissions, click Edit Limits.

   e. In the Access Permission dialog box, select **ANONYMOUS LOGON name** in the Group or user names box. In the Allow column under Permissions for User, select **Remote Access**, and then click OK.

2. Allowing non-administrator users access to a specific WMI namespace:

   a. In the Control Panel, double-click **Administrative Tools**.

   b. In the Administrative Tools window, double-click **Computer Management**.

c. In the Computer Management window, expand the **Services and Applications** tree and double-click the **WMI Control**.

d. Right-click the WMI Control icon and select **Properties**.

e. In the WMI Control Properties window, click the **Security** tab.

f. In the Security tab, select the namespace and click **Security**.

g. Locate the appropriate account and check **Remote Enable in the Permissions list**.

3. Allow for remote administration by either editing the **Group Policy Editor** or using `NETSH`:

   Using the **Group Policy Editor**:

   a. Click Start, click Run, type `GPEDIT.MSC`, and then click OK.

   b. Under the Local Computer Policy heading, double-click **Computer Configuration**.

   c. Double-click **Administrative Templates**, then **Network**, **Network Connections**, and then **Windows Firewall**.

   d. If the computer is in the domain, then double-click **Domain Profile**; otherwise, double-click **Standard Profile**.

   e. Click **Windows Firewall: Allow inbound remote administration exception**.

   f. On the Action menu either select Edit, or double-click the selection from the previous step.

   g. Check the Enabled radio button, and then click OK.

   Using the `NETSH` command:

   a. Open a command prompt window with Administrative rights (you can right-click the Command Prompt icon and click Run as Administrator).

   b. Execute the following command:

```
NETSH firewall set service RemoteAdmin enable
```

4. Now, be sure the user you are logging in with uses the `Name` value and not the `Full Name` value:

   a. In the **Control Panel**, double-click **Administrative Tools**.

   b. In the **Administrative Tools** window, double-click **Computer Management**.

   c. In the **Computer Management** window, expand the **System Tools then Local Users and Groups**.

   d. Click the **Users** node, and on the right side panel locate your user and make sure it uses the **Name** value to connect, and not the **Full Name** value.

5. If the remote computer is running on `Windows XP Professional`, make sure that remote logins are not being forcefully changed to the guest account user (also known as `ForceGuest`), which is enabled by default on computers that are not attached to a domain.

   a. Click Start, click Run, type `SECPOL.MSC`, and then click OK.

b. Under the **Local Policies** node, double-click **Security Options**.

c. Select **Network Access: Sharing and security model for local accounts** and save.

Common Errors

- `0x80070005`
  - DCOM Security was not configured properly (see Computer B, the `Setting DCOM security...` step).
  - The remote computer (Computer B) is a member of WORKGROUP or is in a domain that is untrusted by the client computer (Computer A) (see Computer A, the `Grant Anonymous connections DCOM Remote Access permissions` step).
- `0x8007000E`
  - The remote computer (Computer B) is a member of WORKGROUP or is in a domain that is untrusted by the client computer (Computer A) (see Computer A, the `Grant Anonymous connections DCOM Remote Access permissions` step).
- `0x80041003`
  - Access to the remote WMI namespace was not configured properly (see Computer B, the `Allowing non-administrator users access to a specific WMI namespace` step).
- `0x800706BA`
  - The DCOM port is not open on the client computers (Computer A) firewall. See the `Open the DCOM port TCP 135` step for Computer A.
  - The remote computer (Computer B) is inaccessible because its network location is set to Public. Make sure you can access it through the Windows Explorer.

2.3.5 Installing MySQL on Microsoft Windows Using a `noinstall` Zip Archive

Users who are installing from the `noinstall` package can use the instructions in this section to manually install MySQL. The process for installing MySQL from a Zip archive is as follows:

1. Extract the archive to the desired install directory
2. Create an option file
3. Choose a MySQL server type
4. Start the MySQL server
5. Secure the default user accounts

This process is described in the sections that follow.

2.3.5.1 Extracting the Install Archive

To install MySQL manually, do the following:

1. If you are upgrading from a previous version please refer to Section 2.3.7, “Upgrading MySQL on Windows”, before beginning the upgrade process.

2. Make sure that you are logged in as a user with administrator privileges.

3. Choose an installation location. Traditionally, the MySQL server is installed in `C:\mysql`. The MySQL Installer installs MySQL under `C:\Program Files\MySQL`. If you do not install MySQL at `C:\mysql`, you must specify the path to the install directory during startup or in an option file. See Section 2.3.5.2, "Creating an Option File".

4. Extract the install archive to the chosen installation location using your preferred Zip archive tool. Some tools may extract the archive to a folder within your chosen installation location. If this occurs, you can move the contents of the subfolder into the chosen installation location.

2.3.5.2 Creating an Option File

If you need to specify startup options when you run the server, you can indicate them on the command line or place them in an option file. For options that are used every time the server starts, you may find it most convenient to use an option file to specify your MySQL configuration. This is particularly true under the following circumstances:

- The installation or data directory locations are different from the default locations (`C:\Program Files\MySQL\MySQL Server 5.7` and `C:\Program Files\MySQL\MySQL Server 5.7\data`).

- You need to tune the server settings, such as memory, cache, or InnoDB configuration information.

When the MySQL server starts on Windows, it looks for option files in several locations, such as the Windows directory, `C:\`, and the MySQL installation directory (for the full list of locations, see Section 4.2.3.3, "Using Option Files"). The Windows directory typically is named something like `C:\WINDOWS`. You can determine its exact location from the value of the `WINDIR` environment variable using the following command:

```
C:\> echo %WINDIR%
```

MySQL looks for options in each location first in the `my.ini` file, and then in the `my.cnf` file. However, to avoid confusion, it is best if you use only one file. If your PC uses a boot loader where `C:` is not the boot drive, your only option is to use the `my.ini` file. Whichever option file you use, it must be a plain text file.

> **Note**
>
> When using the MySQL Installer to install MySQL Server, it will create the `my.ini` at the default location. And as of MySQL Server 5.5.27, the user running MySQL Installer is granted full permissions to this new `my.ini`.
>
> In other words, be sure that the MySQL Server user has permission to read the `my.ini` file.

You can also make use of the example option files included with your MySQL distribution; see Section 5.1.2, "Server Configuration Defaults".

An option file can be created and modified with any text editor, such as Notepad. For example, if MySQL is installed in `E:\mysql` and the data directory is in `E:\mydata\data`, you can create an option file containing a `[mysqld]` section to specify values for the `basedir` and `datadir` options:

```
[mysqld]
# set basedir to your installation path
basedir=E:/mysql
# set datadir to the location of your data directory
```

```
datadir=E:/mydata/data
```

Note that Windows path names are specified in option files using (forward) slashes rather than backslashes. If you do use backslashes, double them:

```
[mysqld]
# set basedir to your installation path
basedir=E:\\mysql
# set datadir to the location of your data directory
datadir=E:\\mydata\\data
```

The rules for use of backslash in option file values are given in Section 4.2.3.3, “Using Option Files”.

The data directory is located within the AppData directory for the user running MySQL.

If you would like to use a data directory in a different location, you should copy the entire contents of the data directory to the new location. For example, if you want to use E:\mydata as the data directory instead, you must do two things:

1. Move the entire data directory and all of its contents from the default location (for example C:\Program Files\MySQL\MySQL Server 5.7\data) to E:\mydata.

2. Use a --datadir option to specify the new data directory location each time you start the server.

2.3.5.3 Selecting a MySQL Server Type

The following table shows the available servers for Windows in MySQL 5.7.

Binary	Description
mysqld	Optimized binary with named-pipe support
mysqld-debug	Like mysqld, but compiled with full debugging and automatic memory allocation checking

All of the preceding binaries are optimized for modern Intel processors, but should work on any Intel i386-class or higher processor.

Each of the servers in a distribution support the same set of storage engines. The SHOW ENGINES statement displays which engines a given server supports.

All Windows MySQL 5.7 servers have support for symbolic linking of database directories.

MySQL supports TCP/IP on all Windows platforms. MySQL servers on Windows also support named pipes, if you start the server with the --enable-named-pipe option. It is necessary to use this option explicitly because some users have experienced problems with shutting down the MySQL server when named pipes were used. The default is to use TCP/IP regardless of platform because named pipes are slower than TCP/IP in many Windows configurations.

2.3.5.4 Starting the Server for the First Time

This section gives a general overview of starting the MySQL server. The following sections provide more specific information for starting the MySQL server from the command line or as a Windows service.

The information here applies primarily if you installed MySQL using the Noinstall version, or if you wish to configure and test MySQL manually rather than with the GUI tools.

Note

The MySQL server will automatically start after using the MySQL Installer, and the MySQL Notifier for Microsoft Windows GUI can be used to start/stop/restart at any time.

The examples in these sections assume that MySQL is installed under the default location of `C:\Program Files\MySQL\MySQL Server 5.7`. Adjust the path names shown in the examples if you have MySQL installed in a different location.

Clients have two options. They can use TCP/IP, or they can use a named pipe if the server supports named-pipe connections.

MySQL for Windows also supports shared-memory connections if the server is started with the `--shared-memory` option. Clients can connect through shared memory by using the `--protocol=MEMORY` option.

For information about which server binary to run, see Section 2.3.5.3, “Selecting a MySQL Server Type”.

Testing is best done from a command prompt in a console window (or “DOS window”). In this way you can have the server display status messages in the window where they are easy to see. If something is wrong with your configuration, these messages make it easier for you to identify and fix any problems.

To start the server, enter this command:

```
C:\> "C:\Program Files\MySQL\MySQL Server 5.7\bin\mysqld" --console
```

For a server that includes `InnoDB` support, you should see the messages similar to those following as it starts (the path names and sizes may differ):

```
2013-09-24T12:55:18.897250Z 0 [Note] InnoDB: The first specified data file "ibdata1" did not exist : a new
2013-09-24T12:55:18.897299Z 0 [Note] InnoDB: Need to create new data file "ibdata2"
2013-09-24T12:55:18.897492Z 0 [Note] InnoDB: Setting file "./ibdata1" size to 128 MB
2013-09-24T12:55:18.897509Z 0 [Note] InnoDB: Database physically writes the file full: wait ...
2013-09-24T12:55:19.013723Z 0 [Note] InnoDB: Setting file "./ibdata2" size to 250 MB
2013-09-24T12:55:19.013766Z 0 [Note] InnoDB: Database physically writes the file full: wait ...
2013-09-24T12:55:19.131808Z 0 [Note] InnoDB: Setting log file ./ib_logfile101 size to 48 MB
2013-09-24T12:55:19.571493Z 0 [Note] InnoDB: Setting log file ./ib_logfile1 size to 48 MB
2013-09-24T12:55:20.226902Z 0 [Note] InnoDB: Renaming log file ./ib_logfile101 to ./ib_logfile0
2013-09-24T12:55:20.227251Z 0 [Warning] InnoDB: New log files created, LSN=45781
2013-09-24T12:55:21.227716Z 0 [Note] InnoDB: Creating shared tablespace for temporary tables
2013-09-24T12:55:21.228286Z 0 [Note] InnoDB: Setting file "./ibtmp1" size to 12 MB
2013-09-24T12:55:21.228334Z 0 [Note] InnoDB: Database physically writes the file full: wait ...
2013-09-24T12:55:21.329536Z 0 [Note] InnoDB: Doublewrite buffer not found: creating new
2013-09-24T12:55:21.476956Z 0 [Note] InnoDB: Doublewrite buffer created
2013-09-24T12:55:22.077524Z 0 [Note] InnoDB: 96 redo rollback segment(s) found. 96 redo rollback segment(s
2013-09-24T12:55:22.077564Z 0 [Note] InnoDB: 32 non-redo rollback segment(s) are active.
2013-09-24T12:55:22.182853Z 0 [Warning] InnoDB: Creating foreign key constraint system tables.
2013-09-24T12:55:22.195621Z 0 [Note] InnoDB: Foreign key constraint system tables created
2013-09-24T12:55:22.195791Z 0 [Note] InnoDB: Creating tablespace and datafile system tables.
2013-09-24T12:55:22.202725Z 0 [Note] InnoDB: Tablespace and datafile system tables created.
2013-09-24T12:55:22.202844Z 0 [Note] InnoDB: Waiting for purge to start
2013-09-24T12:55:22.253342Z 0 [Note] InnoDB: 5.7.5 started; log sequence number 0
```

When the server finishes its startup sequence, you should see something like this, which indicates that the server is ready to service client connections:

```
mysqld: ready for connections
```

```
Version: '5.7.5'  socket: ''  port: 3306
```

The server continues to write to the console any further diagnostic output it produces. You can open a new console window in which to run client programs.

If you omit the --console option, the server writes diagnostic output to the error log in the data directory (C:\Program Files\MySQL\MySQL Server 5.7\data by default). The error log is the file with the .err extension, and may be set using the --log-error option.

Note

The accounts that are listed in the MySQL grant tables initially have no passwords. After starting the server, you should set up passwords for them using the instructions in Section 2.9.2, “Securing the Initial MySQL Accounts”.

2.3.5.5 Starting MySQL from the Windows Command Line

The MySQL server can be started manually from the command line. This can be done on any version of Windows.

Note

The MySQL Notifier for Microsoft Windows GUI can also be used to start/stop/restart the MySQL server.

To start the mysqld server from the command line, you should start a console window (or “DOS window”) and enter this command:

```
C:\> "C:\Program Files\MySQL\MySQL Server 5.7\bin\mysqld"
```

The path to mysqld may vary depending on the install location of MySQL on your system.

You can stop the MySQL server by executing this command:

```
C:\> "C:\Program Files\MySQL\MySQL Server 5.7\bin\mysqladmin" -u root shutdown
```

Note

If the MySQL root user account has a password, you need to invoke mysqladmin with the -p option and supply the password when prompted.

This command invokes the MySQL administrative utility mysqladmin to connect to the server and tell it to shut down. The command connects as the MySQL root user, which is the default administrative account in the MySQL grant system. Note that users in the MySQL grant system are wholly independent from any login users under Windows.

If mysqld doesn't start, check the error log to see whether the server wrote any messages there to indicate the cause of the problem. By default, the error log is located in the C:\Program Files\MySQL\MySQL Server 5.7\data directory. It is the file with a suffix of .err, or may be specified by passing in the --log-error option. Alternatively, you can try to start the server as mysqld --console; in this case, you may get some useful information on the screen that may help solve the problem.

The last option is to start mysqld with the --standalone and --debug options. In this case, mysqld writes a log file C:\mysqld.trace that should contain the reason why mysqld doesn't start. See Section 22.4.3, “The DBUG Package”.

Use mysqld --verbose --help to display all the options that mysqld supports.

2.3.5.6 Customizing the PATH for MySQL Tools

To make it easier to invoke MySQL programs, you can add the path name of the MySQL bin directory to your Windows system PATH environment variable:

- On the Windows desktop, right-click the My Computer icon, and select Properties.
- Next select the Advanced tab from the System Properties menu that appears, and click the Environment Variables button.
- Under **System Variables**, select Path, and then click the Edit button. The Edit System Variable dialogue should appear.
- Place your cursor at the end of the text appearing in the space marked **Variable Value**. (Use the **End** key to ensure that your cursor is positioned at the very end of the text in this space.) Then enter the complete path name of your MySQL bin directory (for example, C:\Program Files\MySQL\MySQL Server 5.7\bin)

  Note

  There must be a semicolon separating this path from any values present in this field.

  Dismiss this dialogue, and each dialogue in turn, by clicking OK until all of the dialogues that were opened have been dismissed. You should now be able to invoke any MySQL executable program by typing its name at the DOS prompt from any directory on the system, without having to supply the path. This includes the servers, the mysql client, and all MySQL command-line utilities such as mysqladmin and mysqldump.

  You should not add the MySQL bin directory to your Windows PATH if you are running multiple MySQL servers on the same machine.

Warning

You must exercise great care when editing your system PATH by hand; accidental deletion or modification of any portion of the existing PATH value can leave you with a malfunctioning or even unusable system.

2.3.5.7 Starting MySQL as a Windows Service

On Windows, the recommended way to run MySQL is to install it as a Windows service, so that MySQL starts and stops automatically when Windows starts and stops. A MySQL server installed as a service can also be controlled from the command line using NET commands, or with the graphical Services utility. Generally, to install MySQL as a Windows service you should be logged in using an account that has administrator rights.

Note

The MySQL Notifier for Microsoft Windows GUI can also be used to monitor the status of the MySQL service.

The Services utility (the Windows Service Control Manager) can be found in the Windows Control Panel (under Administrative Tools on Windows 2000, XP, Vista, and Server 2003). To avoid conflicts, it is advisable to close the Services utility while performing server installation or removal operations from the command line.

Installing the service

Before installing MySQL as a Windows service, you should first stop the current server if it is running by using the following command:

```
C:\> "C:\Program Files\MySQL\MySQL Server 5.7\bin\mysqladmin"
          -u root shutdown
```

Note

If the MySQL root user account has a password, you need to invoke mysqladmin with the -p option and supply the password when prompted.

This command invokes the MySQL administrative utility mysqladmin to connect to the server and tell it to shut down. The command connects as the MySQL root user, which is the default administrative account in the MySQL grant system. Note that users in the MySQL grant system are wholly independent from any login users under Windows.

Install the server as a service using this command:

```
C:\> "C:\Program Files\MySQL\MySQL Server 5.7\bin\mysqld" --install
```

The service-installation command does not start the server. Instructions for that are given later in this section.

To make it easier to invoke MySQL programs, you can add the path name of the MySQL bin directory to your Windows system PATH environment variable:

- On the Windows desktop, right-click the My Computer icon, and select Properties.
- Next select the Advanced tab from the System Properties menu that appears, and click the Environment Variables button.
- Under **System Variables**, select Path, and then click the Edit button. The Edit System Variable dialogue should appear.
- Place your cursor at the end of the text appearing in the space marked **Variable Value**. (Use the **End** key to ensure that your cursor is positioned at the very end of the text in this space.) Then enter the complete path name of your MySQL bin directory (for example, C:\Program Files\MySQL\MySQL Server 5.7\bin), Note that there should be a semicolon separating this path from any values present in this field. Dismiss this dialogue, and each dialogue in turn, by clicking OK until all of the dialogues that were opened have been dismissed. You should now be able to invoke any MySQL executable program by typing its name at the DOS prompt from any directory on the system, without having to supply the path. This includes the servers, the mysql client, and all MySQL command-line utilities such as mysqladmin and mysqldump.

  You should not add the MySQL bin directory to your Windows PATH if you are running multiple MySQL servers on the same machine.

Warning

You must exercise great care when editing your system PATH by hand; accidental deletion or modification of any portion of the existing PATH value can leave you with a malfunctioning or even unusable system.

The following additional arguments can be used when installing the service:

- You can specify a service name immediately following the `--install` option. The default service name is `MySQL`.
- If a service name is given, it can be followed by a single option. By convention, this should be `--defaults-file=file_name` to specify the name of an option file from which the server should read options when it starts.

  The use of a single option other than `--defaults-file` is possible but discouraged. `--defaults-file` is more flexible because it enables you to specify multiple startup options for the server by placing them in the named option file.
- You can also specify a `--local-service` option following the service name. This causes the server to run using the `LocalService` Windows account that has limited system privileges. This account is available only for Windows XP or newer. If both `--defaults-file` and `--local-service` are given following the service name, they can be in any order.

For a MySQL server that is installed as a Windows service, the following rules determine the service name and option files that the server uses:

- If the service-installation command specifies no service name or the default service name (`MySQL`) following the `--install` option, the server uses the a service name of `MySQL` and reads options from the `[mysqld]` group in the standard option files.
- If the service-installation command specifies a service name other than `MySQL` following the `--install` option, the server uses that service name. It reads options from the `[mysqld]` group and the group that has the same name as the service in the standard option files. This enables you to use the `[mysqld]` group for options that should be used by all MySQL services, and an option group with the service name for use by the server installed with that service name.
- If the service-installation command specifies a `--defaults-file` option after the service name, the server reads options the same way as described in the previous item, except that it reads options only from the the named file and ignores the standard option files.

As a more complex example, consider the following command:

```
C:\> "C:\Program Files\MySQL\MySQL Server 5.7\bin\mysqld"
          --install MySQL --defaults-file=C:\my-opts.cnf
```

Here, the default service name (`MySQL`) is given after the `--install` option. If no `--defaults-file` option had been given, this command would have the effect of causing the server to read the `[mysqld]` group from the standard option files. However, because the `--defaults-file` option is present, the server reads options from the `[mysqld]` option group, and only from the named file.

> **Note**
>
> On Windows, if the server is started with the `--defaults-file` and `--install` options, `--install` must be first. Otherwise, `mysqld.exe` will attempt to start the MySQL server.

You can also specify options as Start parameters in the Windows `Services` utility before you start the MySQL service.

Starting the service

Once a MySQL server has been installed as a service, Windows starts the service automatically whenever Windows starts. The service also can be started immediately from the `Services` utility, or by using a `NET START MySQL` command. The `NET` command is not case sensitive.

When run as a service, mysqld has no access to a console window, so no messages can be seen there. If mysqld does not start, check the error log to see whether the server wrote any messages there to indicate the cause of the problem. The error log is located in the MySQL data directory (for example, C:\Program Files\MySQL\MySQL Server 5.7\data). It is the file with a suffix of .err.

When a MySQL server has been installed as a service, and the service is running, Windows stops the service automatically when Windows shuts down. The server also can be stopped manually by using the Services utility, the NET STOP MySQL command, or the mysqladmin shutdown command.

You also have the choice of installing the server as a manual service if you do not wish for the service to be started automatically during the boot process. To do this, use the --install-manual option rather than the --install option:

```
C:\> "C:\Program Files\MySQL\MySQL Server 5.7\bin\mysqld" --install-manual
```

Removing the service

To remove a server that is installed as a service, first stop it if it is running by executing NET STOP MySQL. Then use the --remove option to remove it:

```
C:\> "C:\Program Files\MySQL\MySQL Server 5.7\bin\mysqld" --remove
```

If mysqld is not running as a service, you can start it from the command line. For instructions, see Section 2.3.5.5, “Starting MySQL from the Windows Command Line”.

If you encounter difficulties during installation. see Section 2.3.6, “Troubleshooting a Microsoft Windows MySQL Server Installation”.

2.3.5.8 Testing The MySQL Installation

You can test whether the MySQL server is working by executing any of the following commands:

```
C:\> "C:\Program Files\MySQL\MySQL Server 5.7\bin\mysqlshow"
C:\> "C:\Program Files\MySQL\MySQL Server 5.7\bin\mysqlshow" -u root mysql
C:\> "C:\Program Files\MySQL\MySQL Server 5.7\bin\mysqladmin" version status proc
C:\> "C:\Program Files\MySQL\MySQL Server 5.7\bin\mysql" test
```

If mysqld is slow to respond to TCP/IP connections from client programs, there is probably a problem with your DNS. In this case, start mysqld with the --skip-name-resolve option and use only localhost and IP addresses in the Host column of the MySQL grant tables.

You can force a MySQL client to use a named-pipe connection rather than TCP/IP by specifying the --pipe or --protocol=PIPE option, or by specifying . (period) as the host name. Use the --socket option to specify the name of the pipe if you do not want to use the default pipe name.

Note that if you have set a password for the root account, deleted the anonymous account, or created a new user account, then to connect to the MySQL server you must use the appropriate -u and -p options with the commands shown previously. See Section 4.2.2, “Connecting to the MySQL Server”.

For more information about mysqlshow, see Section 4.5.6, “mysqlshow — Display Database, Table, and Column Information”.

2.3.6 Troubleshooting a Microsoft Windows MySQL Server Installation

When installing and running MySQL for the first time, you may encounter certain errors that prevent the MySQL server from starting. This section helps you diagnose and correct some of these errors.

Your first resource when troubleshooting server issues is the error log. The MySQL server uses the error log to record information relevant to the error that prevents the server from starting. The error log is located in the data directory specified in your `my.ini` file. The default data directory location is `C:\Program Files\MySQL\MySQL Server 5.7\data`, or `C:\ProgramData\Mysql` on Windows 7 and Windows Server 2008. The `C:\ProgramData` directory is hidden by default. You need to change your folder options to see the directory and contents. For more information on the error log and understanding the content, see Section 5.2.2, “The Error Log”.

For information regarding possible errors, also consult the console messages displayed when the MySQL service is starting. Use the `NET START MySQL` command from the command line after installing `mysqld` as a service to see any error messages regarding the starting of the MySQL server as a service. See Section 2.3.5.7, “Starting MySQL as a Windows Service”.

The following examples show other common error messages you might encounter when installing MySQL and starting the server for the first time:

- If the MySQL server cannot find the `mysql` privileges database or other critical files, it displays these messages:

  ```
  System error 1067 has occurred.
  Fatal error: Can't open and lock privilege tables:
  Table 'mysql.user' doesn't exist
  ```

  These messages often occur when the MySQL base or data directories are installed in different locations than the default locations (`C:\Program Files\MySQL\MySQL Server 5.7` and `C:\Program Files\MySQL\MySQL Server 5.7\data`, respectively).

  This situation can occur when MySQL is upgraded and installed to a new location, but the configuration file is not updated to reflect the new location. In addition, old and new configuration files might conflict. Be sure to delete or rename any old configuration files when upgrading MySQL.

  If you have installed MySQL to a directory other than `C:\Program Files\MySQL\MySQL Server 5.7`, ensure that the MySQL server is aware of this through the use of a configuration (`my.ini`) file. Put the `my.ini` file in your Windows directory, typically `C:\WINDOWS`. To determine its exact location from the value of the `WINDIR` environment variable, issue the following command from the command prompt:

  ```
  C:\> echo %WINDIR%
  ```

  You can create or modify an option file with any text editor, such as Notepad. For example, if MySQL is installed in `E:\mysql` and the data directory is `D:\MySQLdata`, you can create the option file and set up a `[mysqld]` section to specify values for the `basedir` and `datadir` options:

  ```
  [mysqld]
  # set basedir to your installation path
  basedir=E:/mysql
  # set datadir to the location of your data directory
  datadir=D:/MySQLdata
  ```

  Note that Windows path names are specified in option files using (forward) slashes rather than backslashes. If you do use backslashes, double them:

  ```
  [mysqld]
  # set basedir to your installation path
  basedir=C:\\Program Files\\MySQL\\MySQL Server 5.7
  # set datadir to the location of your data directory
  datadir=D:\\MySQLdata
  ```

The rules for use of backslash in option file values are given in Section 4.2.3.3, “Using Option Files”.

If you change the `datadir` value in your MySQL configuration file, you must move the contents of the existing MySQL data directory before restarting the MySQL server.

See Section 2.3.5.2, “Creating an Option File”.

- If you reinstall or upgrade MySQL without first stopping and removing the existing MySQL service and install MySQL using the MySQL Installer, you might see this error:

  ```
  Error: Cannot create Windows service for MySql. Error: 0
  ```

  This occurs when the Configuration Wizard tries to install the service and finds an existing service with the same name.

  One solution to this problem is to choose a service name other than `mysql` when using the configuration wizard. This enables the new service to be installed correctly, but leaves the outdated service in place. Although this is harmless, it is best to remove old services that are no longer in use.

  To permanently remove the old `mysql` service, execute the following command as a user with administrative privileges, on the command line:

  ```
  C:\> sc delete mysql
  [SC] DeleteService SUCCESS
  ```

  If the `sc` utility is not available for your version of Windows, download the `delsrv` utility from http://www.microsoft.com/windows2000/techinfo/reskit/tools/existing/delsrv-o.asp and use the `delsrv mysql` syntax.

2.3.7 Upgrading MySQL on Windows

To upgrade MySQL on Windows, follow these steps:

1. Review Section 2.10.1, “Upgrading MySQL”, for additional information on upgrading MySQL that is not specific to Windows.

2. Always back up your current MySQL installation before performing an upgrade. See Section 7.2, “Database Backup Methods”.

3. Download the latest Windows distribution of MySQL from http://dev.mysql.com/downloads/.

4. Before upgrading MySQL, stop the server. If the server is installed as a service, stop the service with the following command from the command prompt:

   ```
   C:\> NET STOP MySQL
   ```

   If you are not running the MySQL server as a service, use `mysqladmin` to stop it. For example, before upgrading from MySQL 5.6 to 5.7, use `mysqladmin` from MySQL 5.6 as follows:

   ```
   C:\> "C:\Program Files\MySQL\MySQL Server 5.6\bin\mysqladmin" -u root shutdown
   ```

   Note

   If the MySQL `root` user account has a password, invoke `mysqladmin` with the `-p` option and enter the password when prompted.

5. Before upgrading to MySQL 5.7 from a version previous to 4.1.5, or from a version of MySQL installed from a Zip archive to a version of MySQL installed with the MySQL Installation Wizard, you must first manually remove the previous installation and MySQL service (if the server is installed as a service).

   To remove the MySQL service, use the following command:

   ```
   C:\> C:\mysql\bin\mysqld --remove
   ```

   If you do not remove the existing service, the MySQL Installation Wizard may fail to properly install the new MySQL service.

6. If you are using the MySQL Installer, start it as described in Section 2.3.3, “Installing MySQL on Microsoft Windows Using MySQL Installer”.

7. If you are upgrading MySQL from a Zip archive, extract the archive. You may either overwrite your existing MySQL installation (usually located at `C:\mysql`), or install it into a different directory, such as `C:\mysql5`. Overwriting the existing installation is recommended. However, for upgrades (as opposed to installing for the first time), you must remove the data directory from your existing MySQL installation to avoid replacing your current data files. To do so, follow these steps:

   a. Unzip the Zip archive in some location other than your current MySQL installation

   b. Remove the data directory

   c. Rezip the Zip archive

   d. Unzip the modified Zip archive on top of your existing installation

   Alternatively:

   a. Unzip the Zip archive in some location other than your current MySQL installation

   b. Remove the data directory

   c. Move the data directory from the current MySQL installation to the location of the just-removed data directory

   d. Remove the current MySQL installation

   e. Move the unzipped installation to the location of the just-removed installation

8. If you were running MySQL as a Windows service and you had to remove the service earlier in this procedure, reinstall the service. (See Section 2.3.5.7, “Starting MySQL as a Windows Service”.)

9. Restart the server. For example, use `NET START MySQL` if you run MySQL as a service, or invoke `mysqld` directly otherwise.

10. As Administrator, run `mysql_upgrade` to check your tables, attempt to repair them if necessary, and update your grant tables if they have changed so that you can take advantage of any new capabilities. See Section 4.4.7, “`mysql_upgrade` — Check and Upgrade MySQL Tables”.

11. If you encounter errors, see Section 2.3.6, “Troubleshooting a Microsoft Windows MySQL Server Installation”.

2.3.8 Windows Postinstallation Procedures

GUI tools exist that perform most of the tasks described below, including:

- MySQL Installer: Used to install and upgrade MySQL products.
- MySQL Workbench: Manages the MySQL server and edits SQL queries.
- MySQL Notifier: Starts, stops, or restarts the MySQL server, and monitors its status.
- MySQL for Excel: Edits MySQL data with Microsoft Excel.

On Windows, you need not create the data directory and the grant tables. MySQL Windows distributions include the grant tables with a set of preinitialized accounts in the `mysql` database under the data directory. Regarding passwords, if you installed MySQL using the MySQL Installer, you may have already assigned passwords to the accounts. (See Section 2.3.3, “Installing MySQL on Microsoft Windows Using MySQL Installer”.) Otherwise, use the password-assignment procedure given in Section 2.9.2, “Securing the Initial MySQL Accounts”.

Before setting up passwords, you might want to try running some client programs to make sure that you can connect to the server and that it is operating properly. Make sure that the server is running (see Section 2.3.5.4, “Starting the Server for the First Time”), and then issue the following commands to verify that you can retrieve information from the server. You may need to specify directory different from `C:\mysql\bin` on the command line. If you used the MySQL Installer, the default directory is `C:\Program Files\MySQL\MySQL Server 5.7`, and the `mysql` and `mysqlshow` client programs are in `C:\Program Files\MySQL\MySQL Server 5.7\bin`. See Section 2.3.3, “Installing MySQL on Microsoft Windows Using MySQL Installer”, for more information.

Use `mysqlshow` to see what databases exist:

```
C:\> C:\mysql\bin\mysqlshow
+--------------------+
|     Databases      |
+--------------------+
| information_schema |
| mysql              |
| test               |
+--------------------+
```

The list of installed databases may vary, but will always include the minimum of `mysql` and `information_schema`. In most cases, the `test` database will also be installed automatically.

The preceding command (and commands for other MySQL programs such as `mysql`) may not work if the correct MySQL account does not exist. For example, the program may fail with an error, or you may not be able to view all databases. If you installed using MySQL Installer, then the `root` user will have been created automatically with the password you supplied. In this case, you should use the `-u root` and `-p` options. (You will also need to use the `-u root` and `-p` options if you have already secured the initial MySQL accounts.) With `-p`, you will be prompted for the `root` password. For example:

```
C:\> C:\mysql\bin\mysqlshow -u root -p
Enter password: (enter root password here)
+--------------------+
|     Databases      |
+--------------------+
| information_schema |
| mysql              |
| test               |
+--------------------+
```

If you specify a database name, `mysqlshow` displays a list of the tables within the database:

```
C:\> C:\mysql\bin\mysqlshow mysql
Database: mysql
+---------------------------+
|          Tables           |
+---------------------------+
| columns_priv              |
| db                        |
| event                     |
| func                      |
| help_category             |
| help_keyword              |
| help_relation             |
| help_topic                |
| host                      |
| plugin                    |
| proc                      |
| procs_priv                |
| servers                   |
| tables_priv               |
| time_zone                 |
| time_zone_leap_second     |
| time_zone_name            |
| time_zone_transition      |
| time_zone_transition_type |
| user                      |
+---------------------------+
```

Use the `mysql` program to select information from a table in the `mysql` database:

```
C:\> C:\mysql\bin\mysql -e "SELECT Host,Db,User FROM mysql.db"
+------+--------+------+
| host | db     | user |
+------+--------+------+
| %    | test   |      |
| %    | test_% |      |
+------+--------+------+
```

For more information about `mysqlshow` and `mysql`, see Section 4.5.6, “`mysqlshow` — Display Database, Table, and Column Information”, and Section 4.5.1, “`mysql` — The MySQL Command-Line Tool”.

If you are running a version of Windows that supports services, you can set up the MySQL server to run automatically when Windows starts. See Section 2.3.5.7, “Starting MySQL as a Windows Service”.

2.4 Installing MySQL on Mac OS X

MySQL for Mac OS X is available in a number of different forms:

- Native Package Installer format, which uses the native Mac OS X installer to walk you through the installation of MySQL. For more information, see Section 2.4.2, “Installing MySQL on Mac OS X Using Native Packages”. You can use the package installer with Mac OS X 10.3 and later, and the package is available for both PowerPC and Intel architectures, and 32-bit and 64-bit architectures. There is no Universal Binary available using the package installation method. The user you use to perform the installation must have administrator privileges.

- Tar package format, which uses a file packaged using the Unix `tar` and `gzip` commands. To use this method, you will need to open a `Terminal` window. You do not need administrator privileges using this method, as you can install the MySQL server anywhere using this method. For more information on using this method, you can use the generic instructions for using a tarball, Section 2.2, “Installing MySQL on Unix/Linux Using Generic Binaries”.You can use the package installer with Mac OS X 10.3 and later, and available for both PowerPC and Intel architectures, and both 32-bit and 64-bit architectures. A

Universal Binary, incorporating both Power PC and Intel architectures and 32-bit and 64-bit binaries is available.

In addition to the core installation, the Package Installer also includes Section 2.4.3, “Installing the MySQL Startup Item” and Section 2.4.4, “Installing and Using the MySQL Preference Pane”, both of which simplify the management of your installation.

- Mac OS X server includes a version of MySQL as standard. If you want to use a more recent version than that supplied with the Mac OS X server release, you can make use of the package or tar formats. For more information on using the MySQL bundled with Mac OS X, see Section 2.4.5, “Using the Bundled MySQL on Mac OS X Server”.

For additional information on using MySQL on Mac OS X, see Section 2.4.1, “General Notes on Installing MySQL on Mac OS X”.

2.4.1 General Notes on Installing MySQL on Mac OS X

You should keep the following issues and notes in mind:

- The default location for the MySQL Unix socket is different on Mac OS X and Mac OS X Server depending on the installation type you chose. The following table shows the default locations by installation type.

  Table 2.5 MySQL Unix Socket Locations on Mac OS X by Installation Type

Installation Type	Socket Location
Package Installer from MySQL	`/tmp/mysql.sock`
Tarball from MySQL	`/tmp/mysql.sock`
MySQL Bundled with Mac OS X Server	`/var/mysql/mysql.sock`

  To prevent issues, you should either change the configuration of the socket used within your application (for example, changing `php.ini`), or you should configure the socket location using a MySQL configuration file and the `socket` option. For more information, see Section 5.1.3, “Server Command Options”.

- You may need (or want) to create a specific `mysql` user to own the MySQL directory and data. On Mac OS X 10.4 and lower you can do this by using the `Netinfo Manager` application, located within the `Utilities` folder within the `Applications` folder. On Mac OS X 10.5 and later you can do this through the `Directory Utility`. From Mac OS X 10.5 and later (including Mac OS X Server 10.5) the `mysql` should already exist. For use in single user mode, an entry for `_mysql` (note the underscore prefix) should already exist within the system `/etc/passwd` file.

- Due to a bug in the Mac OS X package installer, you may see this error message in the destination disk selection dialog:

  ```
  You cannot install this software on this disk. (null)
  ```

  If this error occurs, click the `Go Back` button once to return to the previous screen. Then click `Continue` to advance to the destination disk selection again, and you should be able to choose the destination disk correctly. We have reported this bug to Apple and it is investigating this problem.

- If you get an “insecure startup item disabled” error when MySQL launches, use the following procedure. Adjust the pathnames appropriately for your system.

  1. Modify the `mysql.script` using this command (enter it on a single line):

```
shell> sudo /Applications/TextEdit.app/Contents/MacOS/TextEdit
  /usr/local/mysql/support-files/mysql.server
```

2. Locate the option file that defines the `basedir` value and modify it to contain these lines:

   ```
   basedir=/usr/local/mysql
   datadir=/usr/local/mysql/data
   ```

   In the `/Library/StartupItems/MySQLCOM/` directory, make the following group ID changes from `staff` to `wheel`:

   ```
   shell> sudo chgrp wheel MySQLCOM StartupParameters.plist
   ```

3. Start the server from System Preferences or Terminal.app.

- Because the MySQL package installer installs the MySQL contents into a version and platform specific directory, you can use this to upgrade and migrate your database between versions. You will need to either copy the `data` directory from the old version to the new version, or alternatively specify an alternative `datadir` value to set location of the data directory.

- You might want to add aliases to your shell's resource file to make it easier to access commonly used programs such as `mysql` and `mysqladmin` from the command line. The syntax for `bash` is:

  ```
  alias mysql=/usr/local/mysql/bin/mysql
  alias mysqladmin=/usr/local/mysql/bin/mysqladmin
  ```

  For `tcsh`, use:

  ```
  alias mysql /usr/local/mysql/bin/mysql
  alias mysqladmin /usr/local/mysql/bin/mysqladmin
  ```

  Even better, add `/usr/local/mysql/bin` to your `PATH` environment variable. You can do this by modifying the appropriate startup file for your shell. For more information, see Section 4.2.1, "Invoking MySQL Programs".

- After you have copied over the MySQL database files from the previous installation and have successfully started the new server, you should consider removing the old installation files to save disk space. Additionally, you should also remove older versions of the Package Receipt directories located in `/Library/Receipts/mysql-*VERSION*.pkg`.

2.4.2 Installing MySQL on Mac OS X Using Native Packages

You can install MySQL on Mac OS X 10.3.x ("Panther") or newer using a Mac OS X binary package in DMG format instead of the binary tarball distribution. Please note that older versions of Mac OS X (for example, 10.1.x or 10.2.x) are *not* supported by this package.

The package is located inside a disk image (`.dmg`) file that you first need to mount by double-clicking its icon in the Finder. It should then mount the image and display its contents.

> **Note**
>
> Before proceeding with the installation, be sure to stop all running MySQL server instances by using either the MySQL Manager Application (on Mac OS X Server) or `mysqladmin shutdown` on the command line.

When installing from the package version, you should also install the MySQL Preference Pane, which will enable you to control the startup and execution of your MySQL server from System Preferences. For more information, see Section 2.4.4, “Installing and Using the MySQL Preference Pane”.

When installing using the package installer, the files are installed into a directory within /usr/local matching the name of the installation version and platform. For example, the installer file mysql-5.1.39-osx10.5-x86_64.pkg installs MySQL into /usr/local/mysql-5.1.39-osx10.5-x86_64 . The following table shows the layout of the installation directory.

Table 2.6 MySQL Installation Layout on Mac OS X

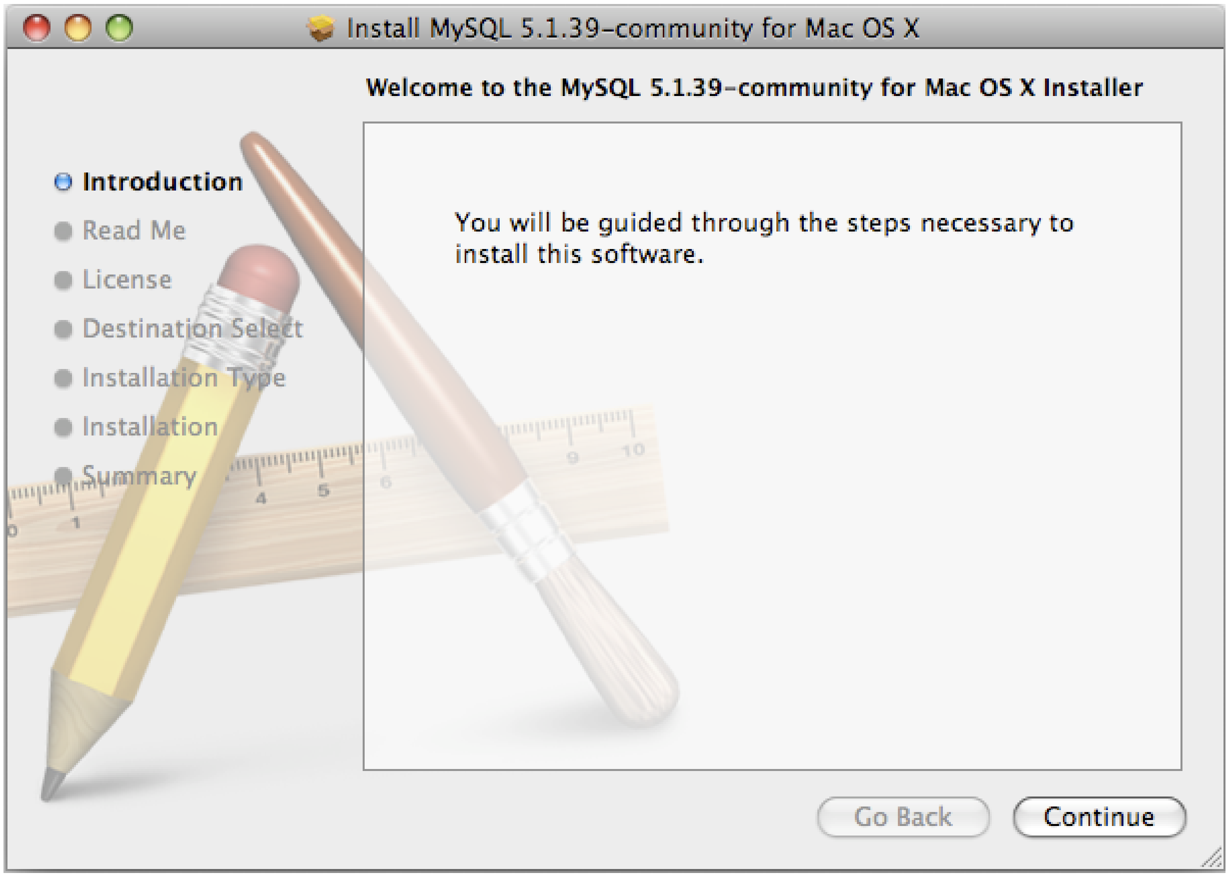
Directory	Contents of Directory
bin	Client programs and the mysqld server
data	Log files, databases
docs	Manual in Info format
include	Include (header) files
lib	Libraries
man	Unix manual pages
mysql-test	MySQL test suite
scripts	mysql_install_db
share	Miscellaneous support files, including error messages, sample configuration files, SQL for database installation
sql-bench	Benchmarks
support-files	Scripts and sample configuration files
/tmp/mysql.sock	Location of the MySQL Unix socket

During the package installer process, a symbolic link from /usr/local/mysql to the version/platform specific directory created during installation will be created automatically.

1. Download and open the MySQL package installer, which is provided on a disk image (.dmg) that includes the main MySQL installation package, the MySQLStartupItem.pkg installation package, and the MySQL.prefPane. Double-click the disk image to open it.

2. Double-click the MySQL installer package. It will be named according to the version of MySQL you have downloaded. For example, if you have downloaded MySQL 5.1.39, double-click mysql-5.1.39-osx10.5-x86.pkg.

3. You will be presented with the opening installer dialog. Click Continue to begin installation.

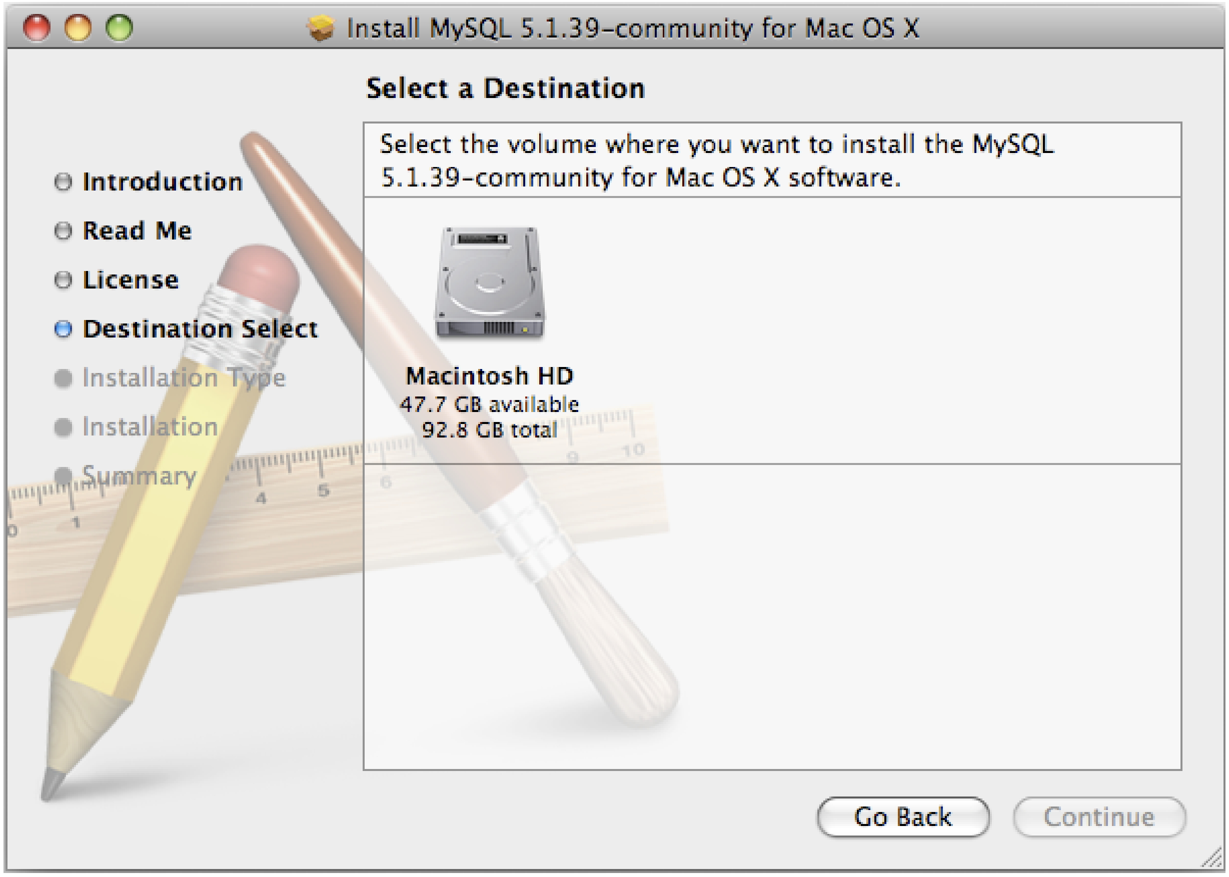

Install MySQL 5.1.39-community for Mac OS X
Welcome to the MySQL 5.1.39-community for Mac OS X Installer
Introduction
Read Me
License
Destination Select
Installation Type
Installation
Summary
Go Back
Continue

You will be guided through the steps necessary to install this software.

4. A copy of the installation instructions and other important information relevant to this installation are displayed. Click Continue .

5. If you have downloaded the community version of MySQL, you will be shown a copy of the relevant GNU General Public License. Click Continue .

6. Select the drive you want to use to install the MySQL Startup Item. The drive must have a valid, bootable, Mac OS X operating system installed. Click Continue.



Select the volume where you want to install the MySQL 5.1.39-community for Mac OS X software.

Macintosh HD
47.7 GB available
92.8 GB total

7. You will be asked to confirm the details of the installation, including the space required for the installation. To change the drive on which the startup item is installed, click either Go Back or Change Install Location.... To install the startup item, click Install.

8. Once the installation has been completed successfully, you will be shown an **Install Succeeded** message.

For convenience, you may also want to install the startup item and preference pane. See Section 2.4.3, “Installing the MySQL Startup Item”, and Section 2.4.4, “Installing and Using the MySQL Preference Pane”.

2.4.3 Installing the MySQL Startup Item

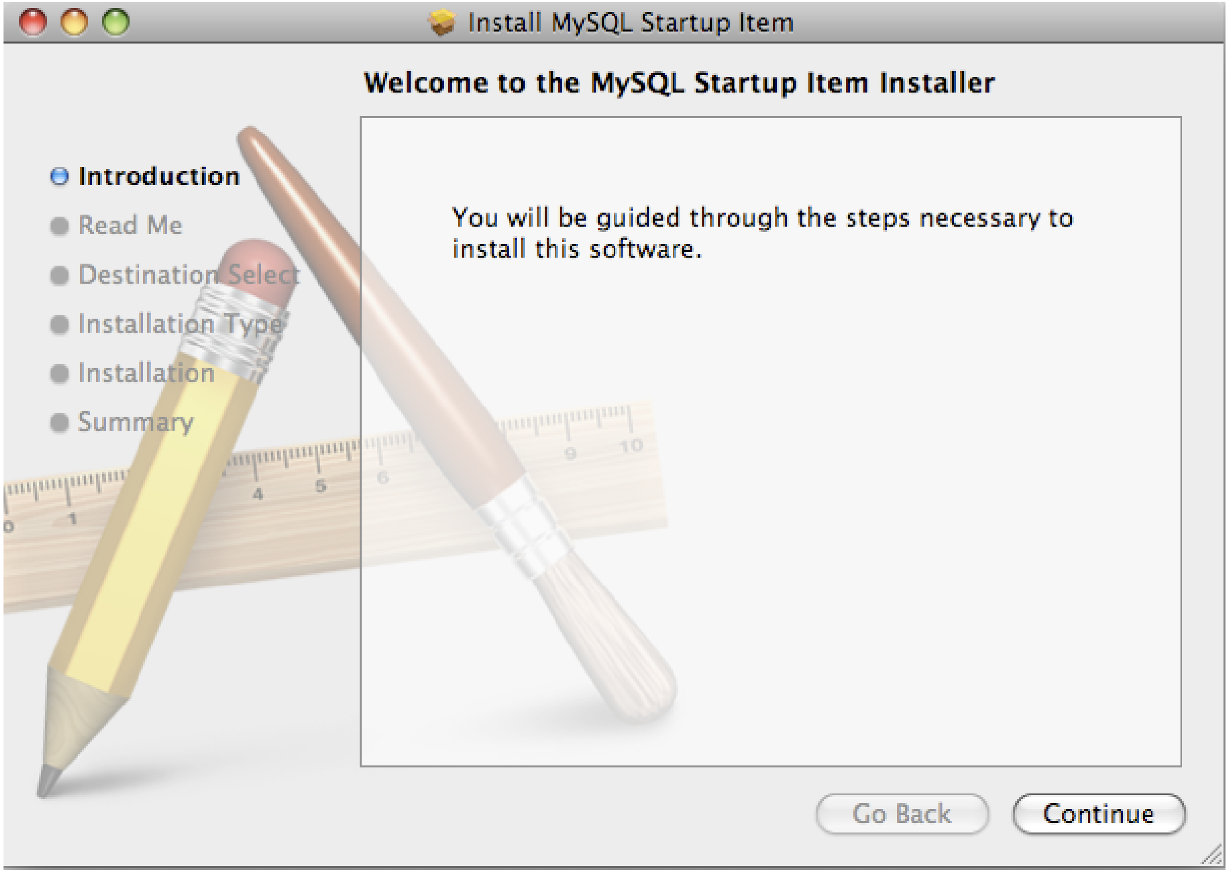
The MySQL Installation Package includes a startup item that can be used to automatically start and stop MySQL.

To install the MySQL Startup Item:

1. Download and open the MySQL package installer, which is provided on a disk image (`.dmg`) that includes the main MySQL installation package, the `MySQLStartupItem.pkg` installation package, and the `MySQL.prefPane`. Double-click the disk image to open it.

2. Double-click the `MySQLStartItem.pkg` file to start the installation process.

3. You will be presented with the **Install MySQL Startup Item** dialog.



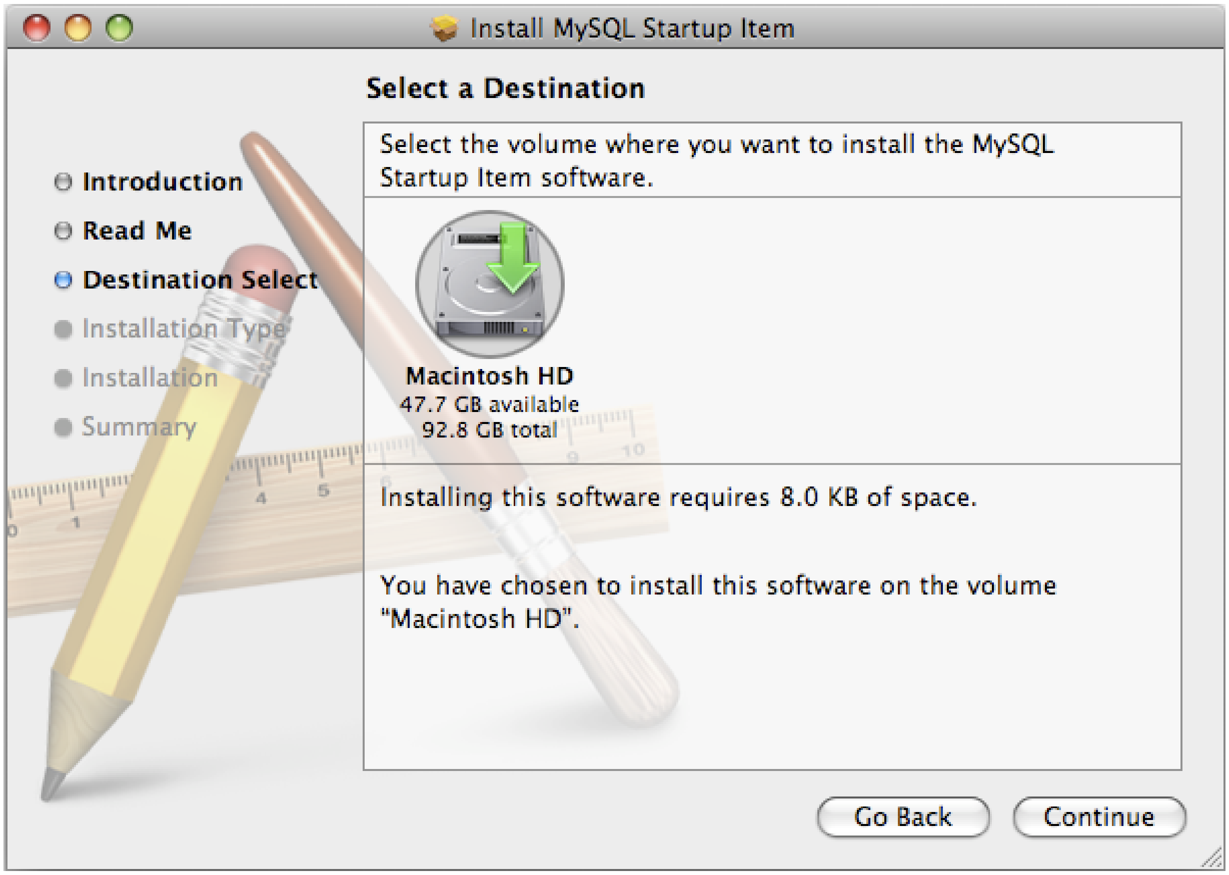
Welcome to the MySQL Startup Item Installer

You will be guided through the steps necessary to install this software.

Click Continue to continue the installation process.

4. A copy of the installation instructions and other important information relevant to this installation are displayed. Click Continue .

5. Select the drive you want to use to install the MySQL Startup Item. The drive must have a valid, bootable, Mac OS X operating system installed. Click Continue.



Select a Destination

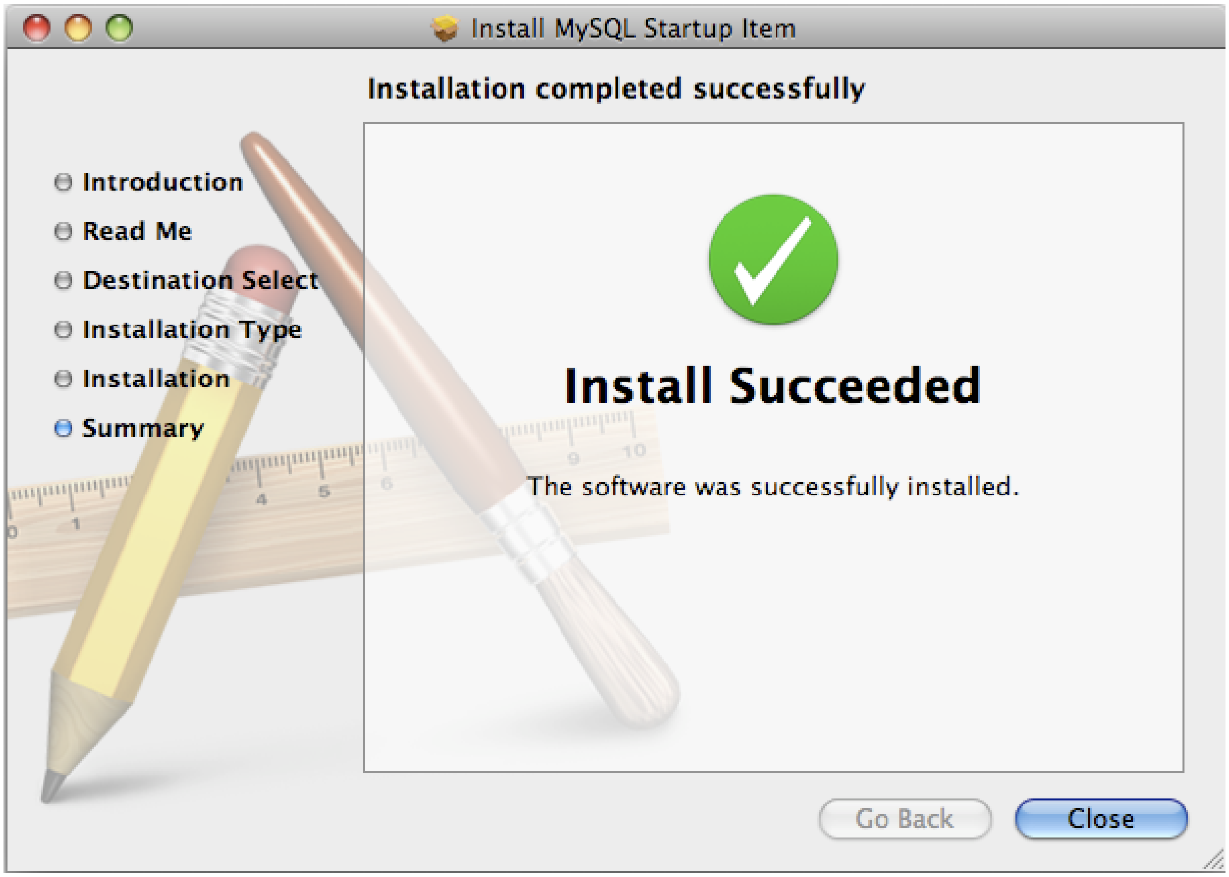
Select the volume where you want to install the MySQL Startup Item software.

Installing this software requires 8.0 KB of space.

You have chosen to install this software on the volume "Macintosh HD".

6. You will be asked to confirm the details of the installation. To change the drive on which the startup item is installed, click either Go Back or Change Install Location.... To install the startup item, click Install.

7. Once the installation has been completed successfully, you will be shown an **Install Succeeded** message.



Installation completed successfully

Install Succeeded

The software was successfully installed.

The Startup Item for MySQL is installed into `/Library/StartupItems/MySQLCOM`. The Startup Item installation adds a variable `MYSQLCOM=-YES-` to the system configuration file `/etc/hostconfig`. If you want to disable the automatic startup of MySQL, change this variable to `MYSQLCOM=-NO-`.

After the installation, you can start and stop MySQL by running the following commands in a terminal window. You must have administrator privileges to perform these tasks, and you may be prompted for your password.

If you have installed the Startup Item, use this command to start the server:

```
shell> sudo /Library/StartupItems/MySQLCOM/MySQLCOM start
```

If you have installed the Startup Item, use this command to stop the server:

```
shell> sudo /Library/StartupItems/MySQLCOM/MySQLCOM stop
```

2.4.4 Installing and Using the MySQL Preference Pane

The MySQL Package installer disk image also includes a custom MySQL Preference Pane that enables you to start, stop, and control automated startup during boot of your MySQL installation.

To install the MySQL Preference Pane:

1. Download and open the MySQL package installer package, which is provided on a disk image (`.dmg`) that includes the main MySQL installation package, the `MySQLStartupItem.pkg` installation package, and the `MySQL.prefPane`. Double-click the disk image to open it.

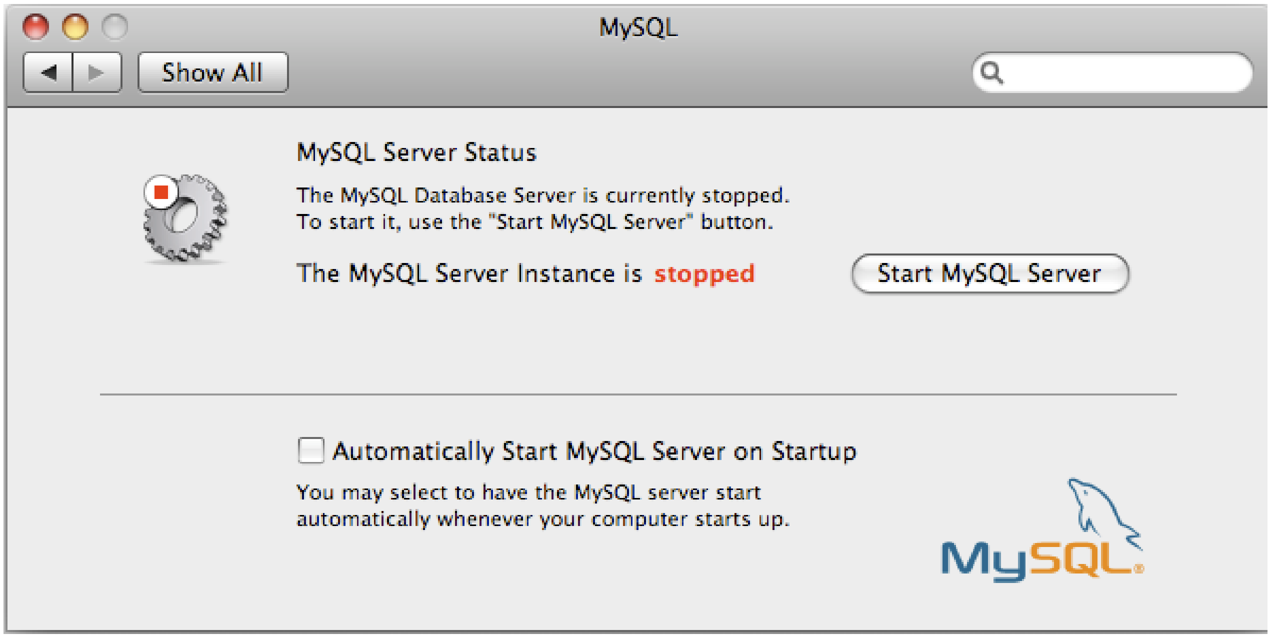
2. Double-click the `MySQL.prefPane`. The MySQL System Preferences will open.

3. If this is the first time you have installed the preference pane, you will be asked to confirm installation and whether you want to install the preference pane for all users, or only the current user. To install the preference pane for all users you will need administrator privileges. If necessary, you will be prompted for the username and password for a user with administrator privileges.

4. If you already have the MySQL Preference Pane installed, you will be asked to confirm whether you want to overwrite the existing MySQL Preference Pane.

> **Note**
>
> The MySQL Preference Pane only starts and stops MySQL installation installed from the MySQL package installation that have been installed in the default location.

Once the MySQL Preference Pane has been installed, you can control your MySQL server instance using the preference pane. To use the preference pane, open the **System Preferences...** from the Apple menu. Select the MySQL preference pane by clicking the MySQL logo within the **Other** section of the preference panes list.



MySQL Server Status

The MySQL Database Server is currently stopped.
To start it, use the "Start MySQL Server" button.

The MySQL Server Instance is stopped

□ Automatically Start MySQL Server on Startup

You may select to have the MySQL server start automatically whenever your computer starts up.

The MySQL Preference Pane shows the current status of the MySQL server, showing **stopped** (in red) if the server is not running and **running** (in green) if the server has already been started. The preference pane also shows the current setting for whether the MySQL server has been set to start automatically.

- **To start MySQL using the preference pane:**

  Click Start MySQL Server. You may be prompted for the username and password of a user with administrator privileges to start the MySQL server.

- **To stop MySQL using the preference pane:**

  Click Stop MySQL Server. You may be prompted for the username and password of a user with administrator privileges to stop the MySQL server.

- **To automatically start the MySQL server when the system boots:**

Check the check box next to **Automatically Start MySQL Server on Startup**.

- **To disable automatic MySQL server startup when the system boots:**

  Uncheck the check box next to **Automatically Start MySQL Server on Startup**.

You can close the System Preferences... window once you have completed your settings.

2.4.5 Using the Bundled MySQL on Mac OS X Server

If you are running Mac OS X Server, a version of MySQL should already be installed. The following table shows the versions of MySQL that ship with Mac OS X Server versions.

Table 2.7 MySQL Versions Preinstalled with Mac OS X Server

Mac OS X Server Version	MySQL Version
10.2-10.2.2	3.23.51
10.2.3-10.2.6	3.23.53
10.3	4.0.14
10.3.2	4.0.16
10.4.0	4.1.10a
10.5.0	5.0.45
10.6.0	5.0.82

The following table shows the installation layout of MySQL on Mac OS X Server.

Table 2.8 MySQL Directory Layout for Preinstalled MySQL Installations on Mac OS X Server

Directory	Contents of Directory
/usr/bin	Client programs
/var/mysql	Log files, databases
/usr/libexec	The mysqld server
/usr/share/man	Unix manual pages
/usr/share/mysql/mysql-test	MySQL test suite
/usr/share/mysql	Miscellaneous support files, including error messages, character set files, sample configuration files, SQL for database installation
/var/mysql/mysql.sock	Location of the MySQL Unix socket

Additional Resources

- For more information on managing the bundled MySQL instance in Mac OS X Server 10.5, see Mac OS X Server: Web Technologies Administration For Version 10.5 Leopard.
- For more information on managing the bundled MySQL instance in Mac OS X Server 10.6, see Mac OS X Server: Web Technologies Administration Version 10.6 Snow Leopard.
- The MySQL server bundled with Mac OS X Server does not include the MySQL client libraries and header files required to access and use MySQL from a third-party driver, such as Perl DBI or PHP. For more information on obtaining and installing MySQL libraries, see Mac OS X Server version 10.5: MySQL libraries available for download. Alternatively, you can ignore the bundled MySQL server and install MySQL from the package or tarball installation.

2.5 Installing MySQL on Linux

Linux supports a number of different solutions for installing MySQL. We recommend that you use one of the distributions from Oracle, for which several methods for installation are available:

- Installing from a generic binary package in `.tar.gz` format. See Section 2.2, "Installing MySQL on Unix/Linux Using Generic Binaries" for more information.
- Extracting and compiling MySQL from a source distribution. For detailed instructions, see Section 2.8, "Installing MySQL from Source".
- Installing with Yum using the MySQL Yum repository. For detailed instructions, see Section 2.5.1, "Installing MySQL on Linux Using the MySQL Yum Repository".
- Installing using a precompiled RPM package. For more information, see Section 2.5.3, "Installing MySQL on Linux Using RPM Packages".
- Installing using a precompiled Debian package. For more information, see Section 2.5.4, "Installing MySQL on Linux Using Debian Packages".

As an alternative, you can use the package manager on your system to automatically download and install MySQL with packages from the native software repositories of your Linux distribution. These native packages are often several versions behind the currently available release. You will also normally be unable to install development milestone releases (DMRs), as these are not usually made available in the native repositories. For more information on using the native package installers, see Section 2.5.5, "Installing MySQL on Linux Using Native Package Managers".

Note

For many Linux installations, you will want to set up MySQL to be started automatically when your machine starts. Many of the native package installations perform this operation for you, but for source, binary and RPM solutions you may need to set this up separately. The required script, `mysql.server`, can be found in the `support-files` directory under the MySQL installation directory or in a MySQL source tree. You can install it as `/etc/init.d/mysql` for automatic MySQL startup and shutdown. See Section 2.9.1.2, "Starting and Stopping MySQL Automatically".

2.5.1 Installing MySQL on Linux Using the MySQL Yum Repository

MySQL provides a Yum-style software repository for the following Linux platforms:

- EL5, EL6, and EL7-based platforms (for example, the relevant versions of Red Hat Enterprise Linux, Oracle Linux, and CentOS)
- Fedora 19 and 20

Currently, the MySQL Yum repository for the above-mentioned platforms provides RPM packages for installing the MySQL server, client, MySQL Workbench, MySQL Utilities (not available for EL5-based platforms), Connector/ODBC, and Connector/Python (not available for EL5-based platforms).

Before You Start

As a popular, open-source software, MySQL, in its original or re-packaged form, is widely installed on many systems from various sources, including different software download sites, software repositories, and so on. The following instructions assume that no versions of MySQL (whether distributed by Oracle or other parties) have already been installed on your system; if that is not the case, see Section 2.10.1.1, "Upgrading MySQL with the MySQL Yum Repository" or Section 2.5.2, "Replacing a Third-Party Distribution of MySQL Using the MySQL Yum Repository".

Steps for a Fresh Installation of the latest GA Version of MySQL

Follow the steps below to install the latest GA version of MySQL with the MySQL Yum repository:

Adding the MySQL Yum Repository

First, add the MySQL Yum repository to your system's repository list. This is a one-time operation, which can be performed by installing an RPM provided by MySQL. Follow these steps:

a. Go to the Download MySQL Yum Repository page (http://dev.mysql.com/downloads/repo) in the MySQL Developer Zone.

b. Select and download the release package for your platform.

c. Install the downloaded release package with the following command (except for EL5-based systems), replacing *platform-and-version-specific* with the name of the downloaded RPM file:

```
shell> sudo yum localinstall platform-and-version-specific.rpm
```

For an EL6-based system, the command is in the form of:

```
shell> sudo yum localinstall mysql-community-release-el6-{version-number}.noarch.rpm
```

For an EL7-based system:

```
shell> sudo yum localinstall mysql-community-release-el7-{version-number}.noarch.rpm
```

For Fedora 19:

```
shell> sudo yum localinstall mysql-community-release-fc19-{version-number}.noarch.rpm
```

For Fedora 20:

```
shell> sudo yum localinstall mysql-community-release-fc20-{version-number}.noarch.rpm
```

For an EL5-based system, use the following command instead:

```
shell> sudo rpm -Uvh mysql-community-release-el5-{version-number}.noarch.rpm
```

The installation command adds the MySQL Yum repository to your system's repository and downloads the GnuPG key to check the integrity of the software packages. See Section 2.1.4.2, "Signature Checking Using GnuPG" for details on GnuPG key checking.

You can check that the MySQL Yum repository has been successfully added by the following command:

```
shell> yum repolist enabled | grep "mysql.*-community.*"
```

Note

Once the MySQL Yum repository is enabled on your system, any system-wide update by the `yum update` command will upgrade MySQL packages on your system and also replace any native third-party packages, if Yum finds replacements for them from within the MySQL Yum repository; see Section 2.10.1.1, “Upgrading MySQL with the MySQL Yum Repository” and, for a discussion on some possible effects of that on your system, see Upgrading to the Shared Client Libraries.

2. Installing MySQL with Yum

Install MySQL by the following command:

```
shell> sudo yum install mysql-server
```

This installs the package for MySQL server (`mysql-community-server`) and also packages for the components required to run the server, including packages for the client (`mysql-community-client`), the common error messages and character sets for client and server (`mysql-community-common`), and the shared client libraries (`mysql-community-libs`).

3. Starting and Stopping the MySQL Server

Start the MySQL server with the following command:

```
shell> sudo service mysqld start
```

This is a sample output of the above command:

```
Starting mysqld:[ OK ]
```

You can check the status of the MySQL server with the following command:

```
shell> sudo service mysqld status
```

This is a sample output of the above command:

```
mysqld (pid 3066) is running.
```

Stop the MySQL server with the following command:

```
shell> sudo service mysqld stop
```

4. Securing the MySQL Installation

Always run the program `mysql_secure_installation` to secure your MySQL installation:

```
shell> mysql_secure_installation
```

`mysql_secure_installation` allows you to perform important operations like setting root password, removing anonymous users, and so on. The program is safe and easy to use. It is important to remember the root password you set though. See Section 4.4.5, “`mysql_secure_installation` — Improve MySQL Installation Security” for details.

For more information on the postinstallation procedures, see Section 2.9, “Postinstallation Setup and Testing”.

Installing Additional MySQL Products and Components with Yum

You can use Yum to install and manage individual components of MySQL. Some of these components are hosted in sub-repositories of the MySQL Yum repository: for example, the MySQL Connectors are to be found in the MySQL Connectors Community sub-repository, and the MySQL Workbench in MySQL Tools Community. You can use the following command to list the packages for all the MySQL components available for your platform from the MySQL Yum repository:

```
shell> sudo yum --disablerepo=\* --enablerepo='mysql*-community*' list available
```

Install any packages of your choice with the following command, replacing *package-name* with name of the package:

```
shell> sudo yum install package-name
```

For example, to install MySQL Workbench:

```
shell> sudo yum install mysql-workbench-community
```

To install the shared client libraries:

```
shell> sudo yum install mysql-community-libs
```

Steps for a Fresh Installation of a Developer Milestone Release (DMR) of MySQL

Follow the steps below to install a developer milestone release (DMR) of MySQL with the MySQL Yum repository:

> **Warning**
>
> Developer milestone releases (DMRs) are for use at your own risk. Significant development changes take place in milestone releases and you may encounter compatibility issues, such as data format changes that require attention in addition to the usual procedure of running mysql_upgrade. For example, you may find it necessary to dump your data with mysqldump before the upgrade and reload it afterward.

1. *Add the MySQL Yum repository* by following the instructions given in Adding the MySQL Yum Repository.

2. *Enable and disable the appropriate sub-repositories*. Inside the MySQL Yum repositories, different release series of the MySQL Community Server are hosted in different sub-repositories. Sub-repository for the latest GA series (currently 5.6) is enabled by default, and sub-repositories for all other series (for example, the 5.7 series, currently still in developer milestone release (DMR) status) are disabled by default. Use this command to see all the sub-repositories in the MySQL Yum repository:

   ```
   shell> yum repolist all | grep "mysql.*-community.*"
   ```

   To install the latest release from a specific series other than the latest GA series, simply disable the sub-repository for the latest GA series and enable the sub-repository for the specific series before running the `yum install` command. This is how it can be done, if, for example, you want to install the 5.7 DMR series:

   ```
   shell> sudo yum-config-manager --disable mysql56-community
   ```

```
shell> sudo yum-config-manager --enable mysql57-community-dmr
```

You can also enable and disable sub-repositories by editing manually the `/etc/yum.repos.d/mysql-community.repo` file. This is a typical entry for a sub-repository in the file:

```
# Enable to use MySQL 5.6
[mysql56-community]
name=MySQL 5.6 Community Server
baseurl=//repo.mysql.com/yum/mysql-5.6-community/el/5/$basearch/
enabled=1
gpgcheck=1
gpgkey=file:/etc/pki/rpm-gpg/RPM-GPG-KEY-mysql
```

Find the entry for the sub-repository you want to configure, and edit the `enabled=` line. Make `enabled=0` to disable a sub-repository, or `enabled=1` to enable a sub-repository.

You can verify that the enabling and disabling of sub-repositories have been done correctly by run the following command and check its output:

```
shell> yum repolist enabled | grep "mysql.*-community.*"
```

> **Note**
>
> You can only enable sub-repository for one release series at a time. When sub-repositories for more than one release series are enabled, the latest series will be used by Yum.

Then, install the MySQL server from the chosen series by the command:

```
shell> sudo yum install mysql-server
```

3. Follow the instructions given in Securing the MySQL Installation and Starting and Stopping the MySQL Server.

Updating MySQL with Yum

Besides installation, you can also perform updates for MySQL products and components using the MySQL Yum repository. See Section 2.10.1.1, “Upgrading MySQL with the MySQL Yum Repository” for details.

2.5.2 Replacing a Third-Party Distribution of MySQL Using the MySQL Yum Repository

Different distributions of MySQL are distributed by different parties through their own software repositories or download sites. You can replace a third-party distribution of MySQL using the MySQL Yum repository in a few steps.

1. Backing Up Your Database

To avoid loss of data, always back up your database before trying to replace your MySQL installation using the MySQL Yum repository. See Chapter 7, *Backup and Recovery* on how to back up your database.

2. Stopping Yum from Receiving MySQL Packages from Third-Party, Non-Native Repositories

Before you can use the MySQL Yum repository for installing (or updating) MySQL, you must stop your system from receiving MySQL packages from any third-party, non-native Yum repositories.

One way to check whether Yum is now receiving third-party MySQL distributions from other repositories is to use the following command:

```
shell> yum list installed mysql\*
```

This is a sample output for the command:

```

mysql.i686              5.1.69-1.el6_4      @updates
mysql-libs.i686         5.1.69-1.el6_4      @updates
mysql-server.i686       5.1.69-1.el6_4      @updates
```

The output shows the names of the packages of the third-party MySQL distribution that are installed and, on the right-hand side, the repository (which is named `updates`, a native repository for the Linux distribution) from which they were installed.

However, sometimes the names of the packages of the third-party distribution might not contain the string “mysql” in it. It might be useful to search also with this command:

```
shell> yum --disablerepo=\* provides mysql\*
```

The following is a sample output of the command:

```

MariaDB-compat-10.0.4-1.i686 ...
...
Repo        : installed
Matched from:
Other       : mysql-libs



MariaDB-server-10.0.4-1.i686 ...
...
Repo        : installed
Matched from:
Other       : mysql-server
```

From the result we can see the names of some of the packages for the installed third-party distribution of MySQL (`MariaDB-server` and `MariaDB-compat`). To try to get an exhaustive list of packages installed for this third-party distribution of MySQL, it might be helpful to search for installed packages of similar names with, for example, the following command:

```
shell> yum list installed mariadb\*
```

This is a sample output for the command:

```

MariaDB-common.i686                     10.0.4-1                       @mariadb
MariaDB-compat.i686                     10.0.4-1                       @mariadb
MariaDB-server.i686                     10.0.4-1                       @mariadb
```

From the command output, we can identify all the installed packages (`MariaDB-common`, `MariaDB-compat`, and `MariaDB-server`) and the third-party Yum repository from which they were installed (named `mariadb`).

The next step is to stop Yum from receiving packages from the third-party Yum repository:

```
shell> sudo yum-config-manager --disable mariadb
```

Note

For platforms like Fedora 19 and 20 that install MySQL from the native repositories, this step is usually not required, unless you have explicitly added a third-party Yum repository for MySQL packages.

Adding the MySQL Yum Repository

Once the third-party Yum repository has been disabled, add the MySQL Yum repository to your system's repository list by following the instructions given in Adding the MySQL Yum Repository.

Uninstalling the Third-Party MySQL Distribution and Installing MySQL with the MySQL Yum Repository

The installed third-party MySQL distribution must first be uninstalled before you can use the MySQL Yum repository to install MySQL, or the installation process will give an error.

Assuming that, as in the example above, the third-part MySQL packages you have found are named `MariaDB-common`, `MariaDB-compat`, and `MariaDB-server`, uninstall them with the following command:

```
shell> sudo yum remove MariaDB-common MariaDB-compat MariaDB-server
```

Note

If your third-party MySQL distribution was not installed by Yum or by an RPM installer, you will not be able to detect and then uninstall it by Yum. If you are not sure what to do in that case, consult a system administrator or the original third-party distributor.

Then, install MySQL from the MySQL Yum repository with the following command:

```
shell> sudo yum install mysql-server
```

The MySQL server and other components required to run the server, including the client, the shared client libraries, and the common error messages and character sets for client and server, are now installed from the MySQL Yum repository. To install more components for MySQL, see Installing Additional MySQL Products and Components with Yum. Follow the postinstallation procedures explained in Section 2.9, “Postinstallation Setup and Testing”.

2.5.3 Installing MySQL on Linux Using RPM Packages

Note

To install or upgrade to MySQL 5.7.2, be sure to read the special instructions at the end of this section.

The recommended way to install MySQL on RPM-based Linux distributions that use `glibc` is by using the RPM packages provided by MySQL. There are two methods for doing so: for EL5, EL6, or EL7-based platforms and Fedora 19 or 20, this can be done using the MySQL Yum repository (see Section 2.5.1, “Installing MySQL on Linux Using the MySQL Yum Repository” for details); for other platforms, we provide various RPM packages that work for different platforms, and this section explains how these packages work.

For non-RPM Linux distributions, you can install MySQL using a `.tar.gz` package. See Section 2.2, “Installing MySQL on Unix/Linux Using Generic Binaries”.

Installations created from our Linux RPM distributions result in files under the system directories shown in the following table.

Table 2.9 MySQL Installation Layout for Linux RPM Packages

Directory	Contents of Directory
`/usr/bin`	Client programs and scripts
`/usr/sbin`	The `mysqld` server
`/var/lib/mysql`	Log files, databases
`/usr/share/info`	Manual in Info format
`/usr/share/man`	Unix manual pages
`/usr/include/mysql`	Include (header) files
`/usr/lib/mysql`	Libraries
`/usr/share/mysql`	Miscellaneous support files, including error messages, character set files, sample configuration files, SQL for database installation
`/usr/share/sql-bench`	Benchmarks

Note

RPM distributions of MySQL are also provided by other vendors. Be aware that they may differ from those built by us in features, capabilities, and conventions (including communication setup), and that the instructions in this manual do not necessarily apply to installing them. The vendor's instructions should be consulted instead. Because of these differences, RPM packages built by us check whether such RPMs built by other vendors are installed. If so, the RPM does not install and produces a message explaining this.

Conflicts can arise when an RPM from another vendor is already installed, such as when a vendor's convention about which files belong with the server and which belong with the client library differ from the breakdown used for Oracle packages. In such cases, attempts to install an Oracle RPM with `rpm -i` may result in messages that files in the RPM to be installed conflict with files from an installed package (denoted `mysql-libs` in the following paragraphs).

We provide a `MySQL-shared-compat` package with each MySQL release. This package is meant to replace `mysql-libs` and provides a replacement-compatible client library for older MySQL series. `MySQL-shared-compat` is set up to make `mysql-libs` obsolete, but `rpm` explicitly refuses to replace obsoleted packages when invoked with `-i` (unlike `-U`), which is why installation with `rpm -i` produces a conflict.

`MySQL-shared-compat` can safely be installed alongside `mysql-libs` because libraries are installed to different locations. Therefore, it is possible to install shared-compat first, then manually remove `mysql-libs` before continuing with the installation. After mysql-libs is removed, the dynamic linker stops looking for the client library in the location where `mysql-libs` puts it, and the library provided by the `MySQL-shared-compat` package takes over.

> Another alternative is to install packages using yum. In a directory containing all RPM packages for a MySQL release, yum install MySQL*rpm installs them in the correct order and removes mysql-libs in one step without conflicts.

In most cases, you need to install only the MySQL-server and MySQL-client packages to get a functional MySQL installation. The other packages are not required for a standard installation.

As of MySQL 5.7.4, MySQL deployments installed using RPM packages are secure by default and have these characteristics:

- The installation process creates a single root account, 'root'@'localhost', automatically generates a random password for this account, and marks the password expired.
- The initial random root password is written to the .mysql_secret file in the directory named by the HOME environment variable. Depending on operating system, using a command such as sudo may cause the value of HOME to refer to the home directory of the root system user. .mysql_secret is created with mode 600 to be accessible only to the system user for whom it is created.

  If .mysql_secret already exists, the new password information is appended to it. Each password entry includes a timestamp so that in the event of multiple install operations it is possible to determine the password associated with each one.
- No anonymous-user MySQL accounts are created.
- No test database is created.

As a result of these actions, it is necessary after installation to start the server, connect as root using the password written to the .mysql_secret file, and select a new root password. Until this is done, root cannot do anything else. To change the password, you can use the SET PASSWORD statement (for example, with the mysql client). You can also use mysqladmin or mysql_secure_installation.

Before MySQL 5.7.4, new RPM install operations produce similar deployment characteristics, except that multiple root accounts may be created, and the test database is created.

For upgrades, if your installation was originally produced by installing multiple RPM packages, it is best to upgrade all the packages, not just some. For example, if you previously installed the server and client RPMs, do not upgrade just the server RPM.

If the data directory exists at RPM installation time, the installation process does not modify existing data. This has the effect, for example, that accounts in the grant tables are not initialized to the default set of accounts.

If you get a dependency failure when trying to install MySQL packages (for example, error: removing these packages would break dependencies: libmysqlclient.so.10 is needed by ...), you should also install the MySQL-shared-compat package, which includes the shared libraries for older releases for backward compatibility.

The RPM packages shown in the following list are available. The names shown here use a suffix of .linux_glibc2.5.i386.rpm, but particular packages can have different suffixes, described later.

- MySQL-server-*VERSION*.linux_glibc2.5.i386.rpm

  The MySQL server. You need this unless you only want to connect to a MySQL server running on another machine.
- MySQL-client-*VERSION*.linux_glibc2.5.i386.rpm

  The standard MySQL client programs. You probably always want to install this package.

- MySQL-devel-*VERSION*.linux_glibc2.5.i386.rpm

  The libraries and include files that are needed if to compile other MySQL clients, such as the Perl modules. Install this RPM if you intend to compile C API applications.

- MySQL-shared-*VERSION*.linux_glibc2.5.i386.rpm

  This package contains the shared libraries (libmysqlclient.so*) that certain languages and applications need to dynamically load and use MySQL. It contains single-threaded and thread-safe libraries. Install this RPM if you intend to compile or run C API applications that depend on the shared client library.

- MySQL-shared-compat-*VERSION*.linux_glibc2.5.i386.rpm

  This package includes the shared libraries for older releases, but not the libraries for the current release. It contains single-threaded and thread-safe libraries. Install this package if you have applications installed that are dynamically linked against older versions of MySQL but you want to upgrade to the current version without breaking the library dependencies.

  The MySQL-shared-compat RPM package enables users of Red Hat-provided mysql-*-5.1 RPM packages to migrate to Oracle-provided MySQL-*-5.5 packages. MySQL-shared-compat replaces the Red Hat mysql-libs package by replacing libmysqlclient.so files of the latter package, thus satisfying dependencies of other packages on mysql-libs. This change affects only users of Red Hat (or Red Hat-compatible) RPM packages. Nothing is different for users of Oracle RPM packages.

- MySQL-embedded-*VERSION*.linux_glibc2.5.i386.rpm

  The embedded MySQL server library.

- MySQL-test-*VERSION*.linux_glibc2.5.i386.rpm

  This package includes the MySQL test suite.

- MySQL-*VERSION*.src.rpm

  This contains the source code for all of the previous packages. It can also be used to rebuild the RPMs on other architectures (for example, Alpha or SPARC).

The suffix of RPM package names (following the *VERSION* value) has the following syntax:

```
.PLATFORM.CPU.rpm
```

The *PLATFORM* and *CPU* values indicate the type of system for which the package is built. *PLATFORM* indicates the platform and *CPU* indicates the processor type or family.

All packages are dynamically linked against glibc 2.5. The *PLATFORM* value indicates whether the package is platform independent or intended for a specific platform, as shown in the following table.

Table 2.10 MySQL Linux Installation Packages

***PLATFORM* Value**	**Intended Use**
linux_glibc25	Platform independent, should run on any Linux distribution that supports glibc 2.5
rhel5, rhel6	Red Hat Enterprise Linux 5 or 6
el6	Enterprise Linux 6
sles10, sles11	SuSE Linux Enterprise Server 10 or 11

In MySQL 5.7, only linux_glibc2.5 packages are available currently.

The *CPU* value indicates the processor type or family for which the package is built.

Table 2.11 MySQL Installation Packages for Linux CPU Identifiers

CPU Value	Intended Processor Type or Family
i386, i586, i686	Pentium processor or better, 32 bit
x86_64	64-bit x86 processor
ia64	Itanium (IA-64) processor

To see all files in an RPM package (for example, a MySQL-server RPM), run a command like this:

```
shell> rpm -qpl MySQL-server-VERSION.linux_glibc2.5.i386.rpm
```

To perform a standard minimal installation, install the server and client RPMs:

```
shell> rpm -i MySQL-server-VERSION.linux_glibc2.5.i386.rpm
shell> rpm -i MySQL-client-VERSION.linux_glibc2.5.i386.rpm
```

To install only the client programs, install just the client RPM:

```
shell> rpm -i MySQL-client-VERSION.linux_glibc2.5.i386.rpm
```

RPM provides a feature to verify the integrity and authenticity of packages before installing them. To learn more about this feature, see Section 2.1.4, “Verifying Package Integrity Using MD5 Checksums or GnuPG”.

The server RPM places data under the /var/lib/mysql directory. The RPM also creates a login account for a user named mysql (if one does not exist) to use for running the MySQL server, and creates the appropriate entries in /etc/init.d/ to start the server automatically at boot time. (This means that if you have performed a previous installation and have made changes to its startup script, you may want to make a copy of the script so that you do not lose it when you install a newer RPM.) See Section 2.9.1.2, “Starting and Stopping MySQL Automatically”, for more information on how MySQL can be started automatically on system startup.

In MySQL 5.7, during a new installation, the server boot scripts are installed, but the MySQL server is not started at the end of the installation, since the status of the server during an unattended installation is not known.

In MySQL 5.7, during an upgrade installation using the RPM packages, if the MySQL server is running when the upgrade occurs, the MySQL server is stopped, the upgrade occurs, and the MySQL server is restarted. If the MySQL server is not already running when the RPM upgrade occurs, the MySQL server is not started at the end of the installation.

If something goes wrong, you can find more information in the binary installation section. See Section 2.2, “Installing MySQL on Unix/Linux Using Generic Binaries”.

Note

The accounts that are listed in the MySQL grant tables initially have no passwords. After starting the server, you should set up passwords for them using the instructions in Section 2.9, “Postinstallation Setup and Testing”.

During RPM installation, a user named mysql and a group named mysql are created on the system. This is done using the useradd, groupadd, and usermod commands. Those commands require appropriate administrative privileges, which is required for locally managed users and groups (as listed in the `/etc/passwd` and `/etc/group` files) by the RPM installation process being run by `root`.

If you log in as the `mysql` user, you may find that MySQL displays “Invalid (old?) table or database name” errors that mention `.mysqlgui`, `lost+found`, `.mysqlgui`, `.bash_history`, `.fonts.cache-1`, `.lesshst`, `.mysql_history`, `.profile`, `.viminfo`, and similar files created by MySQL or operating system utilities. You can safely ignore these error messages or remove the files or directories that cause them if you do not need them.

For nonlocal user management (LDAP, NIS, and so forth), the administrative tools may require additional authentication (such as a password), and will fail if the installing user does not provide this authentication. Even if they fail, the RPM installation will not abort but succeed, and this is intentional. If they failed, some of the intended transfer of ownership may be missing, and it is recommended that the system administrator then manually ensures some appropriate user and group exists and manually transfers ownership following the actions in the RPM spec file.

In MySQL 5.7.2, the RPM spec file has been updated, which has the following consequences:

- For a non-upgrade installation (no existing MySQL version installed), it possible to install MySQL using `yum`.
- For upgrades, it is necessary to clean up any earlier MySQL installations. In effect, the update is performed by removing the old installations and installing the new one.

Additional details follow.

For a non-upgrade installation of MySQL 5.7.2, it is possible to install using `yum`:

```
shell> yum install MySQL-server-NEWVERSION.linux_glibc2.5.i386.rpm
```

For upgrades to MySQL 5.7.2, the upgrade is performed by removing the old installation and installing the new one. To do this, use the following procedure:

1. Remove the existing 5.7.*X* installation. *`OLDVERSION`* is the version to remove.

   ```
   shell> rpm -e MySQL-server-OLDVERSION.linux_glibc2.5.i386.rpm
   ```

   Repeat this step for all installed MySQL RPMs.

2. Install the new version. *`NEWVERSION`* is the version to install.

   ```
   shell> rpm -ivh MySQL-server-NEWVERSION.linux_glibc2.5.i386.rpm
   ```

Alternatively, the removal and installation can be done using `yum`:

```
shell> yum remove MySQL-server-OLDVERSION.linux_glibc2.5.i386.rpm
shell> yum install MySQL-server-NEWVERSION.linux_glibc2.5.i386.rpm
```

2.5.4 Installing MySQL on Linux Using Debian Packages

Oracle provides Debian packages for installation on Debian or Debian-like Linux systems. To obtain a package, see Section 2.1.3, “How to Get MySQL”.

Note

Debian distributions of MySQL are also provided by other vendors. Be aware that they may differ from those built by us in features, capabilities, and conventions

> (including communication setup), and that the instructions in this manual do not necessarily apply to installing them. The vendor's instructions should be consulted instead.

Debian package files have names in `mysql-MVER-DVER-CPU.deb` format. *MVER* is the MySQL version and *DVER* is the Debian version. The *CPU* value indicates the processor type or family for which the package is built, as shown in the following table.

Table 2.12 MySQL Installation Packages for Linux CPU Identifiers

CPU **Value**	**Intended Processor Type or Family**
`i686`	Pentium processor or better, 32 bit
`x86_64`	64-bit x86 processor

After downloading a Debian package, use the following command to install it;

```
shell> dpkg -i mysql-MVER-DVER-CPU.deb
```

The Debian package installs files in the `/opt/mysql/server-5.7` directory.

You may also need to install the `libaio` library if it is not already present on your system:

```
shell> apt-get install libaio1
```

2.5.5 Installing MySQL on Linux Using Native Package Managers

Many Linux distributions include a version of the MySQL server, client tools, and development components in their standard package management system. This section provides basic instructions for installing MySQL using those package management systems.

> **Important**
>
> Native packages are often several versions behind the currently available release. You will also normally be unable to install development milestone releases (DMRs), as these are not usually made available in the native repositories. Before proceeding, we recommend that you check out the other installation options described in Section 2.5, “Installing MySQL on Linux”.

Distribution specific instructions are shown below:

- **Red Hat Linux, Fedora, CentOS**

  > **Note**
  >
  > For EL5, EL6, or EL7-based Linux platforms and Fedora 19 or 20, you can install MySQL using the MySQL Yum repository. See Section 2.5.1, “Installing MySQL on Linux Using the MySQL Yum Repository” for details.

  For Red Hat and similar distributions, the MySQL distribution is divided into a number of separate packages, `mysql` for the client tools, `mysql-server` for the server and associated tools, and `mysql-libs` for the libraries. The libraries are required if you want to provide connectivity from different languages and environments such as Perl, Python and others.

  To install, use the `yum` command to specify the packages that you want to install. For example:

  ```
  root-shell> yum install mysql mysql-server mysql-libs mysql-server
  ```

```
Loaded plugins: presto, refresh-packagekit
Setting up Install Process
Resolving Dependencies
--> Running transaction check
---> Package mysql.x86_64 0:5.1.48-2.fc13 set to be updated
---> Package mysql-libs.x86_64 0:5.1.48-2.fc13 set to be updated
---> Package mysql-server.x86_64 0:5.1.48-2.fc13 set to be updated
--> Processing Dependency: perl-DBD-MySQL for package: mysql-server-5.1.48-2.fc13.x86_64
--> Running transaction check
---> Package perl-DBD-MySQL.x86_64 0:4.017-1.fc13 set to be updated
--> Finished Dependency Resolution

Dependencies Resolved

================================================================================
 Package               Arch          Version               Repository      Size
================================================================================
Installing:
 mysql                 x86_64        5.1.48-2.fc13         updates        889 k
 mysql-libs            x86_64        5.1.48-2.fc13         updates        1.2 M
 mysql-server          x86_64        5.1.48-2.fc13         updates        8.1 M
Installing for dependencies:
 perl-DBD-MySQL        x86_64        4.017-1.fc13          updates        136 k

Transaction Summary
================================================================================
Install       4 Package(s)
Upgrade       0 Package(s)

Total download size: 10 M
Installed size: 30 M
Is this ok [y/N]: y
Downloading Packages:
Setting up and reading Presto delta metadata
Processing delta metadata
Package(s) data still to download: 10 M
(1/4): mysql-5.1.48-2.fc13.x86_64.rpm                    | 889 kB     00:04
(2/4): mysql-libs-5.1.48-2.fc13.x86_64.rpm               | 1.2 MB     00:06
(3/4): mysql-server-5.1.48-2.fc13.x86_64.rpm             | 8.1 MB     00:40
(4/4): perl-DBD-MySQL-4.017-1.fc13.x86_64.rpm            | 136 kB     00:00
--------------------------------------------------------------------------------
Total                                           201 kB/s |  10 MB     00:52
Running rpm_check_debug
Running Transaction Test
Transaction Test Succeeded
Running Transaction
  Installing     : mysql-libs-5.1.48-2.fc13.x86_64                          1/4
  Installing     : mysql-5.1.48-2.fc13.x86_64                               2/4
  Installing     : perl-DBD-MySQL-4.017-1.fc13.x86_64                       3/4
  Installing     : mysql-server-5.1.48-2.fc13.x86_64                        4/4

Installed:
  mysql.x86_64 0:5.1.48-2.fc13            mysql-libs.x86_64 0:5.1.48-2.fc13
  mysql-server.x86_64 0:5.1.48-2.fc13

Dependency Installed:
  perl-DBD-MySQL.x86_64 0:4.017-1.fc13

Complete!
```

MySQL and the MySQL server should now be installed. A sample configuration file is installed into `/etc/my.cnf`. An init script, to start and stop the server, will have been installed into `/etc/init.d/mysqld`. To start the MySQL server use `service`:

```
root-shell> service mysqld start
```

To enable the server to be started and stopped automatically during boot, use chkconfig:

```
root-shell> chkconfig --levels 235 mysqld on
```

Which enables the MySQL server to be started (and stopped) automatically at the specified the run levels.

The database tables will have been automatically created for you, if they do not already exist. You should, however, run mysql_secure_installation to set the root passwords on your server.

- **Debian, Ubuntu, Kubuntu**

  On Debian and related distributions, there are two packages, mysql-client and mysql-server, for the client and server components respectively. You should specify an explicit version, for example mysql-client-5.1, to ensure that you install the version of MySQL that you want.

  To download and install, including any dependencies, use the apt-get command, specifying the packages that you want to install.

  Note

  Before installing, make sure that you update your apt-get index files to ensure you are downloading the latest available version.

  A sample installation of the MySQL packages might look like this (some sections trimmed for clarity):

```
root-shell> apt-get install mysql-client-5.1 mysql-server-5.1
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer required:
  linux-headers-2.6.28-11 linux-headers-2.6.28-11-generic
Use 'apt-get autoremove' to remove them.
The following extra packages will be installed:
  bsd-mailx libdbd-mysql-perl libdbi-perl libhtml-template-perl
  libmysqlclient15off libmysqlclient16 libnet-daemon-perl libplrpc-perl mailx
  mysql-common postfix
Suggested packages:
  dbishell libipc-sharedcache-perl tinyca procmail postfix-mysql postfix-pgsql
  postfix-ldap postfix-pcre sasl2-bin resolvconf postfix-cdb
The following NEW packages will be installed
  bsd-mailx libdbd-mysql-perl libdbi-perl libhtml-template-perl
  libmysqlclient15off libmysqlclient16 libnet-daemon-perl libplrpc-perl mailx
  mysql-client-5.1 mysql-common mysql-server-5.1 postfix
0 upgraded, 13 newly installed, 0 to remove and 182 not upgraded.
Need to get 1907kB/25.3MB of archives.
After this operation, 59.5MB of additional disk space will be used.
Do you want to continue [Y/n]? Y
Get: 1 http://gb.archive.ubuntu.com jaunty-updates/main mysql-common 5.1.30really5.0.75-0ubuntu10.5 [63.6kB]
Get: 2 http://gb.archive.ubuntu.com jaunty-updates/main libmysqlclient15off 5.1.30really5.0.75-0ubuntu10.5 [
Fetched 1907kB in 9s (205kB/s)
Preconfiguring packages ...
Selecting previously deselected package mysql-common.
(Reading database ... 121260 files and directories currently installed.)
...
Processing 1 added doc-base file(s)...
Registering documents with scrollkeeper...
Setting up libnet-daemon-perl (0.43-1) ...
Setting up libplrpc-perl (0.2020-1) ...
Setting up libdbi-perl (1.607-1) ...
Setting up libmysqlclient15off (5.1.30really5.0.75-0ubuntu10.5) ...

Setting up libdbd-mysql-perl (4.008-1) ...
Setting up libmysqlclient16 (5.1.31-1ubuntu2) ...

Setting up mysql-client-5.1 (5.1.31-1ubuntu2) ...

Setting up mysql-server-5.1 (5.1.31-1ubuntu2) ...
 * Stopping MySQL database server mysqld
   ...done.
2013-09-24T13:03:09.048353Z 0 [Note] InnoDB: 5.7.5 started; log sequence number 1566036
2013-09-24T13:03:10.057269Z 0 [Note] InnoDB: Starting shutdown...
2013-09-24T13:03:10.857032Z 0 [Note] InnoDB: Shutdown completed; log sequence number 1566036
 * Starting MySQL database server mysqld
   ...done.
 * Checking for corrupt, not cleanly closed and upgrade needing tables.
...
Processing triggers for libc6 ...
ldconfig deferred processing now taking place
```

Note

The `apt-get` command will install a number of packages, including the MySQL server, in order to provide the typical tools and application environment. This can mean that you install a large number of packages in addition to the main MySQL package.

During installation, the initial database will be created, and you will be prompted for the MySQL root password (and confirmation). A configuration file will have been created in `/etc/mysql/my.cnf`. An init script will have been created in `/etc/init.d/mysql`.

The server will already be started. You can manually start and stop the server using:

```
root-shell> service mysql [start|stop]
```

The service will automatically be added to the 2, 3 and 4 run levels, with stop scripts in the single, shutdown and restart levels.

- **Gentoo Linux**

  As a source-based distribution, installing MySQL on Gentoo involves downloading the source, patching the Gentoo specifics, and then compiling the MySQL server and installing it. This process is handled automatically by the `emerge` command. Depending on the version of MySQL that you want to install, you may need to unmask the specific version that you want for your chosen platform.

  The MySQL server and client tools are provided within a single package, `dev-db/mysql`. You can obtain a list of the versions available to install by looking at the portage directory for the package:

```
root-shell> ls /usr/portage/dev-db/mysql/mysql-5.1*
mysql-5.1.39-r1.ebuild
mysql-5.1.44-r1.ebuild
mysql-5.1.44-r2.ebuild
mysql-5.1.44-r3.ebuild
mysql-5.1.44.ebuild
mysql-5.1.45-r1.ebuild
mysql-5.1.45.ebuild
mysql-5.1.46.ebuild
```

  To install a specific MySQL version, you must specify the entire atom. For example:

```
root-shell> emerge =dev-db/mysql-5.1.46
```

A simpler alternative is to use the `virtual/mysql-5.1` package, which will install the latest version:

```
root-shell> emerge =virtual/mysql-5.1
```

If the package is masked (because it is not tested or certified for the current platform), use the `ACCEPT_KEYWORDS` environment variable. For example:

```
root-shell> ACCEPT_KEYWORDS="~x86" emerge =virtual/mysql-5.1
```

After installation, you should create a new database using `mysql_install_db`, and set the password for the root user on MySQL. You can use the configuration interface to set the password and create the initial database:

```
root-shell> emerge --config =dev-db/mysql-5.1.46
```

A sample configuration file will have been created for you in `/etc/mysql/my.cnf`, and an init script will have been created in `/etc/init.d/mysql`.

To enable MySQL to start automatically at the normal (default) run levels, you can use:

```
root-shell> rc-update add mysql default
```

2.6 Installing MySQL on Solaris and OpenSolaris

MySQL on Solaris and OpenSolaris is available in a number of different formats.

- For information on installing using the native Solaris PKG format, see Section 2.6.1, “Installing MySQL on Solaris Using a Solaris PKG”.

- On OpenSolaris, the standard package repositories include MySQL packages specially built for OpenSolaris that include entries for the Service Management Framework (SMF) to enable control of the installation using the SMF administration commands. For more information, see Section 2.6.2, “Installing MySQL on OpenSolaris Using IPS”.

- To use a standard `tar` binary installation, use the notes provided in Section 2.2, “Installing MySQL on Unix/Linux Using Generic Binaries”. Check the notes and hints at the end of this section for Solaris specific notes that you may need before or after installation.

To obtain a binary MySQL distribution for Solaris in tarball or PKG format, http://dev.mysql.com/downloads/mysql/5.7.html.

Additional notes to be aware of when installing and using MySQL on Solaris:

- If you want to use MySQL with the `mysql` user and group, use the `groupadd` and `useradd` commands:

```
groupadd mysql
useradd -g mysql mysql
```

- If you install MySQL using a binary tarball distribution on Solaris, you may run into trouble even before you get the MySQL distribution unpacked, as the Solaris `tar` cannot handle long file names. This means that you may see errors when you try to unpack MySQL.

If this occurs, you must use GNU tar (gtar) to unpack the distribution. In Solaris 10 and OpenSolaris gtar is normally located in /usr/sfw/bin/gtar, but may not be included in the default path definition.

- When using Solaris 10 for x86_64, you should mount any file systems on which you intend to store InnoDB files with the forcedirectio option. (By default mounting is done without this option.) Failing to do so will cause a significant drop in performance when using the InnoDB storage engine on this platform.
- If you would like MySQL to start automatically, you can copy support-files/mysql.server to /etc/init.d and create a symbolic link to it named /etc/rc3.d/S99mysql.server.
- If too many processes try to connect very rapidly to mysqld, you should see this error in the MySQL log:

  ```
  Error in accept: Protocol error
  ```

  You might try starting the server with the --back_log=50 option as a workaround for this.
- To configure the generation of core files on Solaris you should use the coreadm command. Because of the security implications of generating a core on a setuid() application, by default, Solaris does not support core files on setuid() programs. However, you can modify this behavior using coreadm. If you enable setuid() core files for the current user, they will be generated using the mode 600 and owned by the superuser.

2.6.1 Installing MySQL on Solaris Using a Solaris PKG

You can install MySQL on Solaris and OpenSolaris using a binary package using the native Solaris PKG format instead of the binary tarball distribution.

To use this package, download the corresponding mysql-VERSION-solaris10-PLATFORM.pkg.gz file, then uncompress it. For example:

```
shell> gunzip mysql-5.7.5-solaris10-x86_64.pkg.gz
```

To install a new package, use pkgadd and follow the onscreen prompts. You must have root privileges to perform this operation:

```
shell> pkgadd -d mysql-5.7.5-solaris10-x86_64.pkg

The following packages are available:
  1  mysql     MySQL Community Server (GPL)
               (i86pc) 5.7.5

Select package(s) you wish to process (or 'all' to process
all packages). (default: all) [?,??,q]:
```

The PKG installer installs all of the files and tools needed, and then initializes your database if one does not exist. To complete the installation, you should set the root password for MySQL as provided in the instructions at the end of the installation. Alternatively, you can run the mysql_secure_installation script that comes with the installation.

By default, the PKG package installs MySQL under the root path /opt/mysql. You can change only the installation root path when using pkgadd, which can be used to install MySQL in a different Solaris zone. If you need to install in a specific directory, use a binary tar file distribution.

The pkg installer copies a suitable startup script for MySQL into /etc/init.d/mysql. To enable MySQL to startup and shutdown automatically, you should create a link between this file and the init script directories. For example, to ensure safe startup and shutdown of MySQL you could use the following commands to add the right links:

```
shell> ln /etc/init.d/mysql /etc/rc3.d/S91mysql
shell> ln /etc/init.d/mysql /etc/rc0.d/K02mysql
```

To remove MySQL, the installed package name is mysql. You can use this in combination with the pkgrm command to remove the installation.

To upgrade when using the Solaris package file format, you must remove the existing installation before installing the updated package. Removal of the package does not delete the existing database information, only the server, binaries and support files. The typical upgrade sequence is therefore:

```
shell> mysqladmin shutdown
shell> pkgrm mysql
shell> pkgadd -d mysql-5.7.5-solaris10-x86_64.pkg
shell> mysqld_safe &
shell> mysql_upgrade
```

You should check the notes in Section 2.10, “Upgrading or Downgrading MySQL” before performing any upgrade.

2.6.2 Installing MySQL on OpenSolaris Using IPS

OpenSolaris includes standard packages for MySQL in the core repository. The MySQL packages are based on a specific release of MySQL and updated periodically. For the latest release you must use either the native Solaris PKG, tar, or source installations. The native OpenSolaris packages include SMF files so that you can easily control your MySQL installation, including automatic startup and recovery, using the native service management tools.

To install MySQL on OpenSolaris, use the pkg command. You will need to be logged in as root, or use the pfexec tool, as shown in the example below:

```
shell> pfexec pkg install SUNWmysql57
```

The package set installs three individual packages, SUNWmysql57lib, which contains the MySQL client libraries; SUNWmysql57r which contains the root components, including SMF and configuration files; and SUNWmysql57u which contains the scripts, binary tools and other files. You can install these packages individually if you only need the corresponding components.

The MySQL files are installed into /usr/mysql which symbolic links for the sub directories (bin, lib, etc.) to a version specific directory. For MySQL 5.7, the full installation is located in /usr/mysql/5.7. The default data directory is /var/mysql/5.7/data. The configuration file is installed in /etc/mysql/5.7/my.cnf. This layout permits multiple versions of MySQL to be installed, without overwriting the data and binaries from other versions.

Once installed, you must run mysql_install_db to initialize the database, and use the mysql_secure_installation to secure your installation.

Using SMF to manage your MySQL installation

Once installed, you can start and stop your MySQL server using the installed SMF configuration. The service name is mysql, or if you have multiple versions installed, you should use the full version name, for example mysql:version_57. To start and enable MySQL to be started at boot time:

```
shell> svcadm enable mysql
```

To disable MySQL from starting during boot time, and shut the MySQL server down if it is running, use:

```
shell> svcadm disable mysql
```

To restart MySQL, for example after a configuration file changes, use the `restart` option:

```
shell> svcadm restart mysql
```

You can also use SMF to configure the data directory and enable full 64-bit mode. For example, to set the data directory used by MySQL:

```
shell> svccfg
svc:> select mysql:version_57
svc:/application/database/mysql:version_57> setprop mysql/data=/data0/mysql
```

By default, the 32-bit binaries are used. To enable the 64-bit server on 64-bit platforms, set the `enable_64bit` parameter. For example:

```
svc:/application/database/mysql:version_57> setprop mysql/enable_64bit=1
```

You need to refresh the SMF after settings these options:

```
shell> svcadm refresh mysql
```

2.7 Installing MySQL on FreeBSD

This section provides information about installing MySQL on variants of FreeBSD Unix.

You can install MySQL on FreeBSD by using the binary distribution provided by Oracle. For more information, see Section 2.2, “Installing MySQL on Unix/Linux Using Generic Binaries”.

The easiest (and preferred) way to install MySQL is to use the `mysql-server` and `mysql-client` ports available at http://www.freebsd.org/. Using these ports gives you the following benefits:

- A working MySQL with all optimizations enabled that are known to work on your version of FreeBSD.
- Automatic configuration and build.
- Startup scripts installed in `/usr/local/etc/rc.d`.
- The ability to use `pkg_info -L` to see which files are installed.
- The ability to use `pkg_delete` to remove MySQL if you no longer want it on your machine.

The MySQL build process requires GNU make (`gmake`) to work. If GNU `make` is not available, you must install it first before compiling MySQL.

To install using the ports system:

```
# cd /usr/ports/databases/mysql51-server
# make
...
# cd /usr/ports/databases/mysql51-client
# make
```

```
...
```

The standard port installation places the server into `/usr/local/libexec/mysqld`, with the startup script for the MySQL server placed in `/usr/local/etc/rc.d/mysql-server`.

Some additional notes on the BSD implementation:

- To remove MySQL after installation using the ports system:

  ```
  # cd /usr/ports/databases/mysql51-server
  # make deinstall
  ...
  # cd /usr/ports/databases/mysql51-client
  # make deinstall
  ...
  ```

- If you get problems with the current date in MySQL, setting the `TZ` variable should help. See Section 2.11, “Environment Variables”.

2.8 Installing MySQL from Source

Building MySQL from the source code enables you to customize build parameters, compiler optimizations, and installation location. For a list of systems on which MySQL is known to run, see http://www.mysql.com/support/supportedplatforms/database.html.

Before you proceed with an installation from source, check whether Oracle produces a precompiled binary distribution for your platform and whether it works for you. We put a great deal of effort into ensuring that our binaries are built with the best possible options for optimal performance. Instructions for installing binary distributions are available in Section 2.2, “Installing MySQL on Unix/Linux Using Generic Binaries”.

Source Installation Methods

There are two methods for installing MySQL from source:

- Use a standard MySQL source distribution. To obtain a standard distribution, see Section 2.1.3, “How to Get MySQL”. For instructions on building from a standard distribution, see Section 2.8.2, “Installing MySQL Using a Standard Source Distribution”.

  Standard distributions are available as compressed `tar` files, Zip archives, or RPM packages. Distribution files have names of the form `mysql-`*`VERSION`*`.tar.gz`, `mysql-`*`VERSION`*`.zip`, or `mysql-`*`VERSION`*`.rpm`, where *`VERSION`* is a number like `5.7.5`. File names for source distributions can be distinguished from those for precompiled binary distributions in that source distribution names are generic and include no platform name, whereas binary distribution names include a platform name indicating the type of system for which the distribution is intended (for example, `pc-linux-i686` or `winx64`).

- Use a MySQL development tree. Development trees have not necessarily received the same level of testing as standard release distributions, so this installation method is usually required only if you need the most recent code changes. For information on building from one of the development trees, see Section 2.8.3, “Installing MySQL Using a Development Source Tree”.

Source Installation System Requirements

Installation of MySQL from source requires several development tools. Some of these tools are needed no matter whether you use a standard source distribution or a development source tree. Other tool requirements depend on which installation method you use.

To install MySQL from source, your system must have the following tools, regardless of installation method:

- CMake, which is used as the build framework on all platforms. CMake can be downloaded from http://www.cmake.org.
- A good make program. Although some platforms come with their own make implementations, it is highly recommended that you use GNU make 3.75 or newer. It may already be available on your system as gmake. GNU make is available from http://www.gnu.org/software/make/.
- A working ANSI C++ compiler. GCC 4.2.1 or later, Sun Studio 12 or later, Visual Studio 2010 or later, and many current vendor-supplied compilers are known to work.
- Perl is needed if you intend to run test scripts. Most Unix-like systems include Perl. On Windows, you can use a version such as ActiveState Perl.

To install MySQL from a standard source distribution, one of the following tools is required to unpack the distribution file:

- For a .tar.gz compressed tar file: GNU gunzip to uncompress the distribution and a reasonable tar to unpack it. If your tar program supports the z option, it can both uncompress and unpack the file.

  GNU tar is known to work. The standard tar provided with some operating systems is not able to unpack the long file names in the MySQL distribution. You should download and install GNU tar, or if available, use a preinstalled version of GNU tar. Usually this is available as gnutar, gtar, or as tar within a GNU or Free Software directory, such as /usr/sfw/bin or /usr/local/bin. GNU tar is available from http://www.gnu.org/software/tar/.
- For a .zip Zip archive: WinZip or another tool that can read .zip files.
- For an .rpm RPM package: The rpmbuild program used to build the distribution unpacks it.

To install MySQL from a development source tree, the following additional tools are required:

- To obtain the source tree, you must have Bazaar installed. The Bazaar VCS Web site has instructions for downloading and installing Bazaar on different platforms. Bazaar is supported on any platform that supports Python, and is therefore compatible with any Linux, Unix, Windows, or Mac OS X host.
- bison is needed to generate sql_yacc.cc from sql_yacc.yy You should use the latest version of bison where possible. Versions 1.75 and 2.1 are known to work. There have been reported problems with bison 1.875. If you experience problems, upgrade to a later, rather than earlier, version.

  bison is available from http://www.gnu.org/software/bison/. bison for Windows can be downloaded from http://gnuwin32.sourceforge.net/packages/bison.htm. Download the package labeled “Complete package, excluding sources”. On Windows, the default location for bison is the C:\Program Files\GnuWin32 directory. Some utilities may fail to find bison because of the space in the directory name. Also, Visual Studio may simply hang if there are spaces in the path. You can resolve these problems by installing into a directory that does not contain a space; for example C:\GnuWin32.
- On OpenSolaris and Solaris Express, m4 must be installed in addition to bison. m4 is available from http://www.gnu.org/software/m4/.

Note

If you have to install any programs, modify your PATH environment variable to include any directories in which the programs are located. See Section 4.2.4, “Setting Environment Variables”.

If you run into problems and need to file a bug report, please use the instructions in Section 1.7, “How to Report Bugs or Problems”.

2.8.1 MySQL Layout for Source Installation

By default, when you install MySQL after compiling it from source, the installation step installs files under /usr/local/mysql. The component locations under the installation directory are the same as for binary distributions. See Table 2.3, “MySQL Installation Layout for Generic Unix/Linux Binary Package”, and Section 2.3.1, “MySQL Installation Layout on Microsoft Windows”. To configure installation locations different from the defaults, use the options described at Section 2.8.4, “MySQL Source-Configuration Options”.

2.8.2 Installing MySQL Using a Standard Source Distribution

To install MySQL from a standard source distribution:

1. Verify that your system satisfies the tool requirements listed at Section 2.8, “Installing MySQL from Source”.

2. Obtain a distribution file using the instructions in Section 2.1.3, “How to Get MySQL”.

3. Configure, build, and install the distribution using the instructions in this section.

4. Perform postinstallation procedures using the instructions in Section 2.9, “Postinstallation Setup and Testing”.

In MySQL 5.7, CMake is used as the build framework on all platforms. The instructions given here should enable you to produce a working installation. For additional information on using CMake to build MySQL, see How to Build MySQL Server with CMake.

If you start from a source RPM, use the following command to make a binary RPM that you can install. If you do not have rpmbuild, use rpm instead.

```
shell> rpmbuild --rebuild --clean MySQL-VERSION.src.rpm
```

The result is one or more binary RPM packages that you install as indicated in Section 2.5.3, “Installing MySQL on Linux Using RPM Packages”.

The sequence for installation from a compressed tar file or Zip archive source distribution is similar to the process for installing from a generic binary distribution (see Section 2.2, “Installing MySQL on Unix/Linux Using Generic Binaries”), except that it is used on all platforms and includes steps to configure and compile the distribution. For example, with a compressed tar file source distribution on Unix, the basic installation command sequence looks like this:

```
# Preconfiguration setup
shell> groupadd mysql
shell> useradd -r -g mysql mysql
# Beginning of source-build specific instructions
shell> tar zxvf mysql-VERSION.tar.gz
shell> cd mysql-VERSION
shell> cmake .
shell> make
shell> make install
# End of source-build specific instructions
# Postinstallation setup
shell> cd /usr/local/mysql
shell> chown -R mysql .
shell> chgrp -R mysql .
shell> scripts/mysql_install_db --user=mysql
shell> chown -R root .
shell> chown -R mysql data
shell> bin/mysqld_safe --user=mysql &
# Next command is optional
shell> cp support-files/mysql.server /etc/init.d/mysql.server
```

`mysql_install_db` creates a default option file named `my.cnf` in the base installation directory. This file is created from a template included in the distribution package named `my-default.cnf`. For more information, see Using a Sample Default Server Configuration File.

A more detailed version of the source-build specific instructions is shown following.

Note

The procedure shown here does not set up any passwords for MySQL accounts. After following the procedure, proceed to Section 2.9, “Postinstallation Setup and Testing”, for postinstallation setup and testing.

Perform Preconfiguration Setup

On Unix, set up the `mysql` user and group that will be used to run and execute the MySQL server and own the database directory. For details, see Creating a `mysql` System User and Group, in Section 2.2, “Installing MySQL on Unix/Linux Using Generic Binaries”. Then perform the following steps as the `mysql` user, except as noted.

Obtain and Unpack the Distribution

Pick the directory under which you want to unpack the distribution and change location into it.

Obtain a distribution file using the instructions in Section 2.1.3, “How to Get MySQL”.

Unpack the distribution into the current directory:

- To unpack a compressed `tar` file, `tar` can uncompress and unpack the distribution if it has `z` option support:

  ```
  shell> tar zxvf mysql-VERSION.tar.gz
  ```

  If your `tar` does not have `z` option support, use `gunzip` to unpack the distribution and `tar` to unpack it:

  ```
  shell> gunzip < mysql-VERSION.tar.gz | tar xvf -
  ```

  Alternatively, `CMake` can uncompress and unpack the distribution:

  ```
  shell> cmake -E tar zxvf mysql-VERSION.tar.gz
  ```

- To unpack a Zip archive, use `WinZip` or another tool that can read `.zip` files.

Unpacking the distribution file creates a directory named `mysql-VERSION`.

Configure the Distribution

Change location into the top-level directory of the unpacked distribution:

```
shell> cd mysql-VERSION
```

Configure the source directory. The minimum configuration command includes no options to override configuration defaults:

```
shell> cmake .
```

On Windows, specify the development environment. For example, the following commands configure MySQL for 32-bit or 64-bit builds, respectively:

```
shell> cmake . -G "Visual Studio 10 2010"
shell> cmake . -G "Visual Studio 10 2010 Win64"
```

On Mac OS X, to use the Xcode IDE:

```
shell> cmake . -G Xcode
```

When you run `cmake`, you might want to add options to the command line. Here are some examples:

- `-DBUILD_CONFIG=mysql_release`: Configure the source with the same build options used by Oracle to produce binary distributions for official MySQL releases.
- `-DCMAKE_INSTALL_PREFIX=dir_name`: Configure the distribution for installation under a particular location.
- `-DCPACK_MONOLITHIC_INSTALL=1`: Cause `make package` to generate a single installation file rather than multiple files.
- `-DWITH_DEBUG=1`: Build the distribution with debugging support.

For a more extensive list of options, see Section 2.8.4, "MySQL Source-Configuration Options".

To list the configuration options, use one of the following commands:

```
shell> cmake . -L   # overview
shell> cmake . -LH  # overview with help text
shell> cmake . -LAH # all params with help text
shell> ccmake .     # interactive display
```

If `CMake` fails, you might need to reconfigure by running it again with different options. If you do reconfigure, take note of the following:

- If `CMake` is run after it has previously been run, it may use information that was gathered during its previous invocation. This information is stored in `CMakeCache.txt`. When `CMake` starts up, it looks for that file and reads its contents if it exists, on the assumption that the information is still correct. That assumption is invalid when you reconfigure.
- Each time you run `CMake`, you must run `make` again to recompile. However, you may want to remove old object files from previous builds first because they were compiled using different configuration options.

To prevent old object files or configuration information from being used, run these commands on Unix before re-running `CMake`:

```
shell> make clean
shell> rm CMakeCache.txt
```

Or, on Windows:

```
shell> devenv MySQL.sln /clean
shell> del CMakeCache.txt
```

If you build out of the source tree (as described later), the CMakeCache.txt file and all built files are in the build directory, so you can remove that directory to object files and cached configuration information.

If you are going to send mail to a MySQL mailing list to ask for configuration assistance, first check the files in the CMakeFiles directory for useful information about the failure. To file a bug report, please use the instructions in Section 1.7, “How to Report Bugs or Problems”.

Build the Distribution

On Unix:

```
shell> make
shell> make VERBOSE=1
```

The second command sets VERBOSE to show the commands for each compiled source.

Use gmake instead on systems where you are using GNU make and it has been installed as gmake.

On Windows:

```
shell> devenv MySQL.sln /build RelWithDebInfo
```

It is possible to build out of the source tree to keep the tree clean. If the top-level source directory is named mysql-src under your current working directory, you can build in a directory named bld at the same level like this:

```
shell> mkdir bld
shell> cd bld
shell> cmake ../mysql-src
```

The build directory need not actually be outside the source tree. For example, to build in a directory directory, you can build in a directory named bld under the top-level source tree, do this, starting with mysql-src as your current working directory:

```
shell> mkdir bld
shell> cd bld
shell> cmake ..
```

If you have multiple source trees at the same level (for example, to build multiple versions of MySQL), the second strategy can be advantageous. The first strategy places all build directories at the same level, which requires that you choose a unique name for each. With the second strategy, you can use the same name for the build directory within each source tree.

If you have gotten to the compilation stage, but the distribution does not build, see Section 2.8.5, “Dealing with Problems Compiling MySQL”, for help. If that does not solve the problem, please enter it into our bugs database using the instructions given in Section 1.7, “How to Report Bugs or Problems”. If you have installed the latest versions of the required tools, and they crash trying to process our configuration files, please report that also. However, if you get a command not found error or a similar problem for required tools, do not report it. Instead, make sure that all the required tools are installed and that your PATH variable is set correctly so that your shell can find them.

Install the Distribution

On Unix:

```
shell> make install
```

This installs the files under the configured installation directory (by default, /usr/local/mysql). You might need to run the command as root.

To install in a specific directory, add a DESTDIR parameter to the command line:

```
shell> make install DESTDIR="/opt/mysql"
```

Alternatively, generate installation package files that you can install where you like:

```
shell> make package
```

This operation produces one or more .tar.gz files that can be installed like generic binary distribution packages. See Section 2.2, “Installing MySQL on Unix/Linux Using Generic Binaries”. If you run CMake with -DCPACK_MONOLITHIC_INSTALL=1, the operation produces a single file. Otherwise, it produces multiple files.

On Windows, generate the data directory, then create a .zip archive installation package:

```
shell> devenv MySQL.sln /build RelWithDebInfo /project initial_database
shell> devenv MySQL.sln /build RelWithDebInfo /project package
```

You can install the resulting .zip archive where you like. See Section 2.3.5, “Installing MySQL on Microsoft Windows Using a noinstall Zip Archive”.

Perform Postinstallation Setup

The remainder of the installation process involves setting up the configuration file, creating the core databases, and starting the MySQL server. For instructions, see Section 2.9, “Postinstallation Setup and Testing”.

Note

The accounts that are listed in the MySQL grant tables initially have no passwords. After starting the server, you should set up passwords for them using the instructions in Section 2.9, “Postinstallation Setup and Testing”.

2.8.3 Installing MySQL Using a Development Source Tree

This section discusses how to install MySQL from the latest development source code. Development trees have not necessarily received the same level of testing as standard release distributions, so this installation method is usually required only if you need the most recent code changes. Do not use a development tree for production systems. If your goal is simply to get MySQL up and running on your system, you should use a standard release distribution (either a binary or source distribution). See Section 2.1.3, “How to Get MySQL”.

MySQL development projects are hosted on Launchpad. MySQL projects, including MySQL Server, MySQL Workbench, and others are available from the Oracle/MySQL Engineering page. For the repositories related only to MySQL Server, see the MySQL Server page.

To install MySQL from a development source tree, your system must satisfy the tool requirements listed at Section 2.8, “Installing MySQL from Source”, including the requirements for Bazaar and bison.

To create a local branch of the MySQL development tree on your machine, use this procedure:

1. To obtain a copy of the MySQL source code, you must create a new Bazaar branch. If you do not already have a Bazaar repository directory set up, you must initialize a new directory:

   ```
   shell> mkdir mysql-server
   shell> bzr init-repo --trees mysql-server
   ```

   This is a one-time operation.

2. Assuming that you have an initialized repository directory, you can branch from the public MySQL server repositories to create a local source tree. To create a branch of a specific version:

   ```
   shell> cd mysql-server
   shell> bzr branch lp:mysql-server/5.7 mysql-5.7
   ```

   This is a one-time operation per source tree. You can branch the source trees for several versions of MySQL under the mysql-server directory.

3. The initial download will take some time to complete, depending on the speed of your connection. Please be patient. Once you have downloaded the first tree, additional trees should take significantly less time to download.

4. When building from the Bazaar branch, you may want to create a copy of your active branch so that you can make configuration and other changes without affecting the original branch contents. You can achieve this by branching from the original branch:

   ```
   shell> bzr branch mysql-5.7 mysql-5.7-build
   ```

5. To obtain changes made after you have set up the branch initially, update it using the pull option periodically. Use this command in the top-level directory of the local copy:

   ```
   shell> bzr pull
   ```

   To examine the changeset comments for the tree, use the log option to bzr:

   ```
   shell> bzr log
   ```

   You can also browse changesets, comments, and source code online at the Launchpad MySQL Server page.

   If you see diffs (changes) or code that you have a question about, do not hesitate to send email to the MySQL internals mailing list. See Section 1.6.1, “MySQL Mailing Lists”. If you think you have a better idea on how to do something, send an email message to the list with a patch.

After you have the local branch, you can build MySQL server from the source code. For information, see Section 2.8.2, “Installing MySQL Using a Standard Source Distribution”, except that you skip the part about obtaining and unpacking the distribution.

Be careful about installing a build from a distribution source tree on a production machine. The installation command may overwrite your live release installation. If you already have MySQL installed and do not want to overwrite it, run CMake with values for the CMAKE_INSTALL_PREFIX, MYSQL_TCP_PORT, and

MYSQL_UNIX_ADDR options different from those used by your production server. For additional information about preventing multiple servers from interfering with each other, see Section 5.3, “Running Multiple MySQL Instances on One Machine”.

Play hard with your new installation. For example, try to make new features crash. Start by running make test. See Section 22.1.2, “The MySQL Test Suite”.

2.8.4 MySQL Source-Configuration Options

The CMake program provides a great deal of control over how you configure a MySQL source distribution. Typically, you do this using options on the CMake command line. For information about options supported by CMake, run either of these commands in the top-level source directory:

```
shell> cmake . -LH
shell> ccmake .
```

You can also affect CMake using certain environment variables. See Section 2.11, “Environment Variables”.

The following table shows the available CMake options. In the Default column, PREFIX stands for the value of the CMAKE_INSTALL_PREFIX option, which specifies the installation base directory. This value is used as the parent location for several of the installation subdirectories.

Table 2.13 MySQL Source-Configuration Option Reference (CMake)

Formats	Description	Default	Introduced	Removed
BUILD_CONFIG	Use same build options as official releases			
CMAKE_BUILD_TYPE	Type of build to produce	RelWithDebInfo		
CMAKE_C_FLAGS	Flags for C Compiler			
CMAKE_CXX_FLAGS	Flags for C++ Compiler			
CMAKE_INSTALL_PREFIX	Installation base directory	/usr/local/mysql		
COMPILATION_COMMENT	Comment about compilation environment			
CPACK_MONOLITHIC_INSTALL	Whether package build produces single file	OFF		
DEFAULT_CHARSET	The default server character set	latin1		
DEFAULT_COLLATION	The default server collation	latin1_swedish_ci		
DISABLE_PSI_COND	Exclude Performance Schema condition instrumentation	OFF	5.7.3	
DISABLE_PSI_FILE	Exclude Performance Schema file instrumentation	OFF	5.7.3	
DISABLE_PSI_IDLE	Exclude Performance Schema idle instrumentation	OFF	5.7.3	
DISABLE_PSI_MEMORY	Exclude Performance Schema memory instrumentation	OFF	5.7.3	
DISABLE_PSI_METADATA	Exclude Performance Schema metadata instrumentation	OFF	5.7.3	

Formats	Description	Default	Introduced	Removed
DISABLE_PSI_MUTEX	Exclude Performance Schema mutex instrumentation	OFF	5.7.3	
DISABLE_PSI_RWLOCK	Exclude Performance Schema rwlock instrumentation	OFF	5.7.3	
DISABLE_PSI_SOCKET	Exclude Performance Schema socket instrumentation	OFF	5.7.3	
DISABLE_PSI_SP	Exclude Performance Schema stored program instrumentation	OFF	5.7.3	
DISABLE_PSI_STAGE	Exclude Performance Schema stage instrumentation	OFF	5.7.3	
DISABLE_PSI_STATEMENT	Exclude Performance Schema statement instrumentation	OFF	5.7.3	
DISABLE_PSI_STATEMENT_DIGEST	Exclude Performance Schema statement_digest instrumentation	OFF	5.7.3	
DISABLE_PSI_TABLE	Exclude Performance Schema table instrumentation	OFF	5.7.3	
ENABLE_DEBUG_SYNC	Whether to enable Debug Sync support	ON		
ENABLE_DOWNLOADS	Whether to download optional files	OFF		
ENABLE_DTRACE	Whether to include DTrace support			
ENABLE_GCOV	Whether to include gcov support			
ENABLE_GPROF	Enable gprof (optimized Linux builds only)	OFF		
ENABLED_LOCAL_INFILE	Whether to enable LOCAL for LOAD DATA INFILE	OFF		
ENABLED_PROFILING	Whether to enable query profiling code	ON		
IGNORE_AIO_CHECK	With -DBUILD_CONFIG=mysql_release, ignore libaio check	OFF		
INNODB_PAGE_ATOMIC_REF_COUNT	Enable or disable atomic page reference counting	ON	5.7.4	
INSTALL_BINDIR	User executables directory	PREFIX/bin		
INSTALL_DOCDIR	Documentation directory	PREFIX/docs		
INSTALL_DOCREADMEDIR	README file directory	PREFIX		
INSTALL_INCLUDEDIR	Header file directory	PREFIX/include		
INSTALL_INFODIR	Info file directory	PREFIX/docs		
INSTALL_LAYOUT	Select predefined installation layout	STANDALONE		

Formats	Description	Default	Introduced	Removed
INSTALL_LIBDIR	Library file directory	PREFIX/lib		
INSTALL_MANDIR	Manual page directory	PREFIX/man		
INSTALL_MYSQLSHAREDIR	Shared data directory	PREFIX/share		
INSTALL_MYSQLTESTDIR	mysql-test directory	PREFIX/mysql-test		
INSTALL_PLUGINDIR	Plugin directory	PREFIX/lib/plugin		
INSTALL_SBINDIR	Server executable directory	PREFIX/bin		
INSTALL_SCRIPTDIR	Scripts directory	PREFIX/scripts		
INSTALL_SHAREDIR	aclocal/mysql.m4 installation directory	PREFIX/share		
INSTALL_SQLBENCHDIR	sql-bench directory	PREFIX		
INSTALL_SUPPORTFILESDIR	Extra support files directory	PREFIX/support-files		
MAX_INDEXES	Maximum indexes per table	64	5.7.1	
MYSQL_DATADIR	Data directory			
MYSQL_MAINTAINER_MODE	Whether to enable MySQL maintainer-specific development environment	OFF		
MYSQL_PROJECT_NAME	Windows/Mac OS X project name	3306		
MYSQL_TCP_PORT	TCP/IP port number	3306		
MYSQL_UNIX_ADDR	Unix socket file	/tmp/mysql.sock		
ODBC_INCLUDES	ODBC includes directory			
ODBC_LIB_DIR	ODBC library directory			
OPTIMIZER_TRACE	Whether to support optimizer tracing			
SYSCONFDIR	Option file directory			
TMPDIR	tmpdir default value		5.7.4	
WITH_ASAN	Enable AddressSanitizer	OFF	5.7.3	
WITH_AUTHENTICATION_PAM	Build PAM authentication plugin	OFF		
WITH_CLIENT_PROTOCOL_TRACING	Build client-side protocol tracing framework	ON	5.7.2	
WITH_DEBUG	Whether to include debugging support	OFF		
WITH_DEFAULT_COMPILER_OPTIONS	Whether to use default compiler options	ON		
WITH_DEFAULT_FEATURE_SET	Whether to use default feature set	ON		
WITH_EDITLINE	Which libedit/editline library to use	bundled	5.7.2	

Formats	Description	Default	Introduced	Removed
WITH_EMBEDDED_SERVER	Whether to build embedded server	OFF		
WITH_xxx_STORAGE_ENGINE	Compile storage engine xxx statically into server			
WITH_EXTRA_CHARSETS	Which extra character sets to include	all		
WITH_INNODB_MEMCACHED	Whether to generate memcached shared libraries.	OFF		
WITH_LIBEVENT	Which libevent library to use	bundled		
WITH_LIBWRAP	Whether to include libwrap (TCP wrappers) support	OFF		
WITH_MSAN	Enable MemorySanitizer	OFF	5.7.4	
WITH_SSL	Type of SSL support	no		
WITH_TEST_TRACE_PLUGIN	Build test protocol trace plugin	OFF	5.7.2	
WITH_UNIXODBC	Enable unixODBC support	OFF		
WITH_ZLIB	Type of zlib support	system		
WITHOUT_xxx_STORAGE_ENGINE	Exclude storage engine xxx from build			
WITHOUT_SERVER	Do not build the server	OFF		

The following sections provide more information about CMake options.

- General Options
- Installation Layout Options
- Feature Options
- Compiler Flags

For boolean options, the value may be specified as 1 or ON to enable the option, or as 0 or OFF to disable the option.

Many options configure compile-time defaults that can be overridden at server startup. For example, the CMAKE_INSTALL_PREFIX, MYSQL_TCP_PORT, and MYSQL_UNIX_ADDR options that configure the default installation base directory location, TCP/IP port number, and Unix socket file can be changed at server startup with the --basedir, --port, and --socket options for mysqld. Where applicable, configuration option descriptions indicate the corresponding mysqld startup option.

General Options

- -DBUILD_CONFIG=mysql_release

  This option configures a source distribution with the same build options used by Oracle to produce binary distributions for official MySQL releases.

- -DCMAKE_BUILD_TYPE=*type*

  The type of build to produce:

  - `RelWithDebInfo`: Enable optimizations and generate debugging information. This is the default MySQL build type.
  - `Debug`: Disable optimizations and generate debugging information. This build type is also used if the `WITH_DEBUG` option is enabled. That is, `-DWITH_DEBUG=1` has the same effect as `-DCMAKE_BUILD_TYPE=Debug`.

- `-DCPACK_MONOLITHIC_INSTALL=bool`

  This option affects whether the `make package` operation produces multiple installation package files or a single file. If disabled, the operation produces multiple installation package files, which may be useful if you want to install only a subset of a full MySQL installation. If enabled, it produces a single file for installing everything.

Installation Layout Options

The `CMAKE_INSTALL_PREFIX` option indicates the base installation directory. Other options with names of the form `INSTALL_xxx` that indicate component locations are interpreted relative to the prefix and their values are relative pathnames. Their values should not include the prefix.

- `-DCMAKE_INSTALL_PREFIX=dir_name`

  The installation base directory.

  This value can be set at server startup with the `--basedir` option.

- `-DINSTALL_BINDIR=dir_name`

  Where to install user programs.

- `-DINSTALL_DOCDIR=dir_name`

  Where to install documentation.

- `-DINSTALL_DOCREADMEDIR=dir_name`

  Where to install `README` files.

- `-DINSTALL_INCLUDEDIR=dir_name`

  Where to install header files.

- `-DINSTALL_INFODIR=dir_name`

  Where to install Info files.

- `-DINSTALL_LAYOUT=name`

  Select a predefined installation layout:

  - `STANDALONE`: Same layout as used for `.tar.gz` and `.zip` packages. This is the default.
  - `RPM`: Layout similar to RPM packages.
  - `SVR4`: Solaris package layout.
  - `DEB`: DEB package layout (experimental).

You can select a predefined layout but modify individual component installation locations by specifying other options. For example:

```
shell> cmake . -DINSTALL_LAYOUT=SVR4 -DMYSQL_DATADIR=/var/mysql/data
```

- `-DINSTALL_LIBDIR=dir_name`

  Where to install library files.

- `-DINSTALL_MANDIR=dir_name`

  Where to install manual pages.

- `-DINSTALL_MYSQLSHAREDIR=dir_name`

  Where to install shared data files.

- `-DINSTALL_MYSQLTESTDIR=dir_name`

  Where to install the `mysql-test` directory. As of MySQL 5.7.2, to suppress installation of this directory, explicitly set the option to the empty value (`-DINSTALL_MYSQLTESTDIR=`).

- `-DINSTALL_PLUGINDIR=dir_name`

  The location of the plugin directory.

  This value can be set at server startup with the `--plugin_dir` option.

- `-DINSTALL_SBINDIR=dir_name`

  Where to install the `mysqld` server.

- `-DINSTALL_SCRIPTDIR=dir_name`

  Where to install `mysql_install_db`.

- `-DINSTALL_SHAREDIR=dir_name`

  Where to install `aclocal/mysql.m4`.

- `-DINSTALL_SQLBENCHDIR=dir_name`

  Where to install the `sql-bench` directory. To suppress installation of this directory, explicitly set the option to the empty value (`-DINSTALL_SQLBENCHDIR=`).

- `-DINSTALL_SUPPORTFILESDIR=dir_name`

  Where to install extra support files.

- `-DMYSQL_DATADIR=dir_name`

  The location of the MySQL data directory.

  This value can be set at server startup with the `--datadir` option.

- `-DODBC_INCLUDES=dir_name`

  The location of the ODBC includes directory, and may be used while configuring Connector/ODBC.

- `-DODBC_LIB_DIR=dir_name`

  The location of the ODBC library directory, and may be used while configuring Connector/ODBC.

- `-DSYSCONFDIR=dir_name`

  The default `my.cnf` option file directory.

  This location cannot be set at server startup, but you can start the server with a given option file using the `--defaults-file=file_name` option, where *`file_name`* is the full path name to the file.

- `-DTMPDIR=dir_name`

  The default location to use for the `tmpdir` system variable. If unspecified, the value defaults to `P_tmpdir` in `<stdio.h>`. This option was added in MySQL 5.7.4.

Storage Engine Options

Storage engines are built as plugins. You can build a plugin as a static module (compiled into the server) or a dynamic module (built as a dynamic library that must be installed into the server using the `INSTALL PLUGIN` statement or the `--plugin-load` option before it can be used). Some plugins might not support static or dynamic building.

The `MyISAM`, `MERGE`, `MEMORY`, and `CSV` engines are mandatory (always compiled into the server) and need not be installed explicitly.

To compile a storage engine statically into the server, use `-DWITH_engine_STORAGE_ENGINE=1`. Some permissible *`engine`* values are `ARCHIVE`, `BLACKHOLE`, `EXAMPLE`, `FEDERATED`, `INNOBASE` (`InnoDB`), `PARTITION` (partitioning support), and `PERFSCHEMA` (Performance Schema). Examples:

```
-DWITH_INNOBASE_STORAGE_ENGINE=1
-DWITH_ARCHIVE_STORAGE_ENGINE=1
-DWITH_BLACKHOLE_STORAGE_ENGINE=1
-DWITH_PERFSCHEMA_STORAGE_ENGINE=1
```

As of MySQL 5.7.4, to exclude a storage engine from the build, use `-DWITH_engine_STORAGE_ENGINE=0`. Examples:

```
-DWITH_EXAMPLE_STORAGE_ENGINE=0
-DWITH_FEDERATED_STORAGE_ENGINE=0
-DWITH_PARTITION_STORAGE_ENGINE=0
```

Before MySQL 5.7.4, to exclude a storage engine from the build, use `-DWITHOUT_engine_STORAGE_ENGINE=1`. (That syntax also works in 5.7.4 or later, but `-DWITH_engine_STORAGE_ENGINE=0` is preferred.) Examples:

```
-DWITHOUT_EXAMPLE_STORAGE_ENGINE=1
-DWITHOUT_FEDERATED_STORAGE_ENGINE=1
-DWITHOUT_PARTITION_STORAGE_ENGINE=1
```

If neither `-DWITH_engine_STORAGE_ENGINE` nor `-DWITHOUT_engine_STORAGE_ENGINE` are specified for a given storage engine, the engine is built as a shared module, or excluded if it cannot be built as a shared module.

Feature Options

- `-DCOMPILATION_COMMENT=string`

A descriptive comment about the compilation environment.

- `-DDEFAULT_CHARSET=charset_name`

  The server character set. By default, MySQL uses the `latin1` (cp1252 West European) character set.

  `charset_name` may be one of `binary`, `armscii8`, `ascii`, `big5`, `cp1250`, `cp1251`, `cp1256`, `cp1257`, `cp850`, `cp852`, `cp866`, `cp932`, `dec8`, `eucjpms`, `euckr`, `gb2312`, `gbk`, `geostd8`, `greek`, `hebrew`, `hp8`, `keybcs2`, `koi8r`, `koi8u`, `latin1`, `latin2`, `latin5`, `latin7`, `macce`, `macroman`, `sjis`, `swe7`, `tis620`, `ucs2`, `ujis`, `utf8`, `utf8mb4`, `utf16`, `utf16le`, `utf32`. The permissible character sets are listed in the `cmake/character_sets.cmake` file as the value of `CHARSETS_AVAILABLE`.

  This value can be set at server startup with the `--character_set_server` option.

- `-DDEFAULT_COLLATION=collation_name`

  The server collation. By default, MySQL uses `latin1_swedish_ci`. Use the `SHOW COLLATION` statement to determine which collations are available for each character set.

  This value can be set at server startup with the `--collation_server` option.

- `-DDISABLE_PSI_COND=bool`

  Whether to exclude the Performance Schema condition instrumentation. The default is `OFF` (include). This option was added in MySQL 5.7.3.

- `-DDISABLE_PSI_FILE=bool`

  Whether to exclude the Performance Schema file instrumentation. The default is `OFF` (include). This option was added in MySQL 5.7.3.

- `-DDISABLE_PSI_IDLE=bool`

  Whether to exclude the Performance Schema idle instrumentation. The default is `OFF` (include). This option was added in MySQL 5.7.3.

- `-DDISABLE_PSI_MEMORY=bool`

  Whether to exclude the Performance Schema memory instrumentation. The default is `OFF` (include). This option was added in MySQL 5.7.3.

- `-DDISABLE_PSI_METADATA=bool`

  Whether to exclude the Performance Schema metadata instrumentation. The default is `OFF` (include). This option was added in MySQL 5.7.3.

- `-DDISABLE_PSI_MUTEX=bool`

  Whether to exclude the Performance Schema mutex instrumentation. The default is `OFF` (include). This option was added in MySQL 5.7.3.

- `-DDISABLE_PSI_RWLOCK=bool`

  Whether to exclude the Performance Schema rwlock instrumentation. The default is `OFF` (include). This option was added in MySQL 5.7.3.

- `-DDISABLE_PSI_SOCKET=bool`

Whether to exclude the Performance Schema socket instrumentation. The default is `OFF` (include). This option was added in MySQL 5.7.3.

- `-DDISABLE_PSI_SP=bool`

  Whether to exclude the Performance Schema stored program instrumentation. The default is `OFF` (include). This option was added in MySQL 5.7.3.

- `-DDISABLE_PSI_STAGE=bool`

  Whether to exclude the Performance Schema stage instrumentation. The default is `OFF` (include). This option was added in MySQL 5.7.3.

- `-DDISABLE_PSI_STATEMENT=bool`

  Whether to exclude the Performance Schema statement instrumentation. The default is `OFF` (include). This option was added in MySQL 5.7.3.

- `-DDISABLE_PSI_STATEMENT_DIGEST=bool`

  Whether to exclude the Performance Schema statement_digest instrumentation. The default is `OFF` (include). This option was added in MySQL 5.7.3.

- `-DDISABLE_PSI_TABLE=bool`

  Whether to exclude the Performance Schema table instrumentation. The default is `OFF` (include). This option was added in MySQL 5.7.3.

- `-DENABLE_DEBUG_SYNC=bool`

  Whether to compile the Debug Sync facility into the server. This facility is used for testing and debugging. This option is enabled by default, but has no effect unless MySQL is configured with debugging enabled. If debugging is enabled and you want to disable Debug Sync, use `-DENABLE_DEBUG_SYNC=0`.

  When compiled in, Debug Sync is disabled by default at runtime. To enable it, start `mysqld` with the `--debug-sync-timeout=N` option, where *`N`* is a timeout value greater than 0. (The default value is 0, which disables Debug Sync.) *`N`* becomes the default timeout for individual synchronization points.

  For a description of the Debug Sync facility and how to use synchronization points, see MySQL Internals: Test Synchronization.

- `-DENABLE_DOWNLOADS=bool`

  Whether to download optional files. For example, with this option enabled, `CMake` downloads the Google Test distribution that is used by the test suite to run unit tests.

- `-DENABLE_DTRACE=bool`

  Whether to include support for DTrace probes. For information about DTrace, wee Section 5.4, “Tracing `mysqld` Using DTrace”

- `-DENABLE_GCOV=bool`

  Whether to include gcov support (Linux only).

- `-DENABLE_GPROF=bool`

Whether to enable gprof (optimized Linux builds only).

- -DENABLED_LOCAL_INFILE=*bool*

  Whether to enable LOCAL capability in the client library for LOAD DATA INFILE.

  This option controls client-side LOCAL capability, but the capability can be set on the server side at server startup with the --local-infile option. See Section 6.1.6, "Security Issues with LOAD DATA LOCAL".

- -DENABLED_PROFILING=*bool*

  Whether to enable query profiling code (for the SHOW PROFILE and SHOW PROFILES statements).

- -DIGNORE_AIO_CHECK=*bool*

  If the -DBUILD_CONFIG=mysql_release option is given on Linux, the libaio library must be linked in by default. If you do not have libaio or do not want to install it, you can suppress the check for it by specifying -DIGNORE_AIO_CHECK=1.

- -DINNODB_PAGE_ATOMIC_REF_COUNT=*bool*

  Whether to enable or disable atomic page reference counting. Fetching and releasing pages from the buffer pool and tracking the page state are expensive and complex operations. Using a page mutex to track these operations does not scale well. With INNODB_PAGE_ATOMIC_REF_COUNT=ON (default), fetch and release is tracked using atomics where available. For platforms that do not support atomics, set INNODB_PAGE_ATOMIC_REF_COUNT=OFF to disable atomic page reference counting.

  When atomic page reference counting is enabled (default), "[Note] InnoDB: Using atomics to ref count buffer pool pages" is printed to the error log at server startup. If atomic page reference counting is disabled, "[Note] InnoDB: Using mutexes to ref count buffer pool pages" is printed instead.

  This build option was introduced with the fix for MySQL Bug #68079.

- -DMAX_INDEXES=*num*

  The maximum number of indexes per table. The default is 64. The maximum is 255. Values smaller than 64 are ignored and the default of 64 is used.

- -DMYSQL_MAINTAINER_MODE=*bool*

  Whether to enable a MySQL maintainer-specific development environment. If enabled, this option causes compiler warnings to become errors.

- -DMYSQL_PROJECT_NAME=*name*

  For Windows or Mac OS X, the project name to incorporate into the project file name.

- -DMYSQL_TCP_PORT=*port_num*

  The port number on on which the server listens for TCP/IP connections. The default is 3306.

  This value can be set at server startup with the --port option.

- -DMYSQL_UNIX_ADDR=*file_name*

The Unix socket file path on which the server listens for socket connections. This must be an absolute path name. The default is `/tmp/mysql.sock`.

This value can be set at server startup with the `--socket` option.

- `-DOPTIMIZER_TRACE=bool`

  Whether to support optimizer tracing. See MySQL Internals: Tracing the Optimizer.

- `-DWITH_ASAN=bool`

  Whether to enable AddressSanitizer, for compilers that support it. The default is off. This option was added in MySQL 5.7.3.

- `-DWITH_AUTHENTICATION_PAM=bool`

  Whether to build the PAM authentication plugin, for source trees that include this plugin. (See The PAM Authentication Plugin.) Beginning with MySQL 5.7.2, if this option is specified and the plugin cannot e compiled, the build fails.

- `-DWITH_CLIENT_PROTOCOL_TRACING=bool`

  Whether to build the client-side protocol tracing framework into the client library. By default, this option is enabled. This option was added in MySQL 5.7.2.

  For information about writing protocol trace client plugins, see Section 22.2.4.11, “Writing Protocol Trace Plugins”.

  See also the `WITH_TEST_TRACE_PLUGIN` option.

- `-DWITH_DEBUG=bool`

  Whether to include debugging support.

  Configuring MySQL with debugging support enables you to use the `--debug="d,parser_debug"` option when you start the server. This causes the Bison parser that is used to process SQL statements to dump a parser trace to the server's standard error output. Typically, this output is written to the error log.

- `-DWITH_DEFAULT_FEATURE_SET=bool`

  Whether to use the flags from `cmake/build_configurations/feature_set.cmake`.

- `-DWITH_EDITLINE=value`

  Which `libedit`/`editline` library to use. The permitted values are `bundled` (the default) and `system`.

  `WITH_EDITLINE` was added in MySQL 5.7.2. It replaces `WITH_LIBEDIT`, which has been removed.

- `-DWITH_EMBEDDED_SERVER=bool`

  Whether to build the `libmysqld` embedded server library.

- `-DWITH_EXTRA_CHARSETS=name`

  Which extra character sets to include:

  - `all`: All character sets. This is the default.

  - `complex`: Complex character sets.
  - `none`: No extra character sets.

- `-DWITH_INNODB_EXTRA_DEBUG=`*`bool`*

  Whether to include extra InnoDB debugging support.

  Enabling `WITH_INNODB_EXTRA_DEBUG` turns on extra InnoDB debug checks. This option can only be enabled when `WITH_DEBUG` is enabled.

- `-DWITH_INNODB_MEMCACHED=`*`bool`*

  Whether to generate memcached shared libraries (`libmemcached.so` and `innodb_engine.so`).

- `-DWITH_LIBEVENT=`*`string`*

  Which `libevent` library to use. Permitted values are `bundled` (default), `system`, and `yes`. If you specify `system` or `yes`, the system `libevent` library is used if present. If the system library is not found, the bundled `libevent` library is used. The `libevent` library is required by `InnoDB` memcached.

- `-DWITH_LIBWRAP=`*`bool`*

  Whether to include `libwrap` (TCP wrappers) support.

- `-DWITH_MSAN=`*`bool`*

  Whether to enable MemorySanitizer, for compilers that support it. The default is off. This option was added in MySQL 5.7.4.

- `-DWITH_SSL={`*`ssl_type`*|*`path_name`*}

  The type of SSL support to include or the path name to the OpenSSL installation to use.

  - *`ssl_type`* can be one of the following values:
    - `yes`: Use the system SSL library if present, else the library bundled with the distribution.
    - `bundled`: Use the SSL library bundled with the distribution. This is the default.
    - `system`: Use the system SSL library.
  - *`path_name`* is the path name to the OpenSSL installation to use. Using this can be preferable to using the *`ssl_type`* value of `system`, for it can prevent CMake from detecting and using an older or incorrect OpenSSL version installed on the system. (Another permitted way to do the same thing is to set the `CMAKE_PREFIX_PATH` option to *`path_name`*.)

  For information about using SSL support, see Section 6.3.11, “Using SSL for Secure Connections”.

- `-DWITH_TEST_TRACE_PLUGIN=`*`bool`*

  Whether to build the test protocol trace client plugin (see Using the Test Protocol Trace Plugin). By default, this option is disabled. Enabling this option has no effect unless the `WITH_CLIENT_PROTOCOL_TRACING` option is enabled. If MySQL is configured with both options enabled, the `libmysqlclient` client library is built with the test protocol trace plugin built in, and all the standard MySQL clients load the plugin. However, even when the test plugin is enabled, it has no effect by default. Control over the plugin is afforded using environment variables; see Using the Test Protocol Trace Plugin.

This option was added in MySQL 5.7.2.

> **Note**
>
> Do *not* enable the WITH_TEST_TRACE_PLUGIN option if you want to use your own protocol trace plugins because only one such plugin can be loaded at a time and an error occurs for attempts to load a second one. If you have already built MySQL with the test protocol trace plugin enabled to see how it works, you must rebuild MySQL without it before you can use your own plugins.

For information about writing trace plugins, see Section 22.2.4.11, “Writing Protocol Trace Plugins”.

- -DWITH_UNIXODBC=*1*

  Enables unixODBC support, for Connector/ODBC.

- -DWITH_ZLIB=*zlib_type*

  Some features require that the server be built with compression library support, such as the COMPRESS() and UNCOMPRESS() functions, and compression of the client/server protocol. The WITH_ZLIB indicates the source of zlib support:

  - bundled: Use the zlib library bundled with the distribution.
  - system: Use the system zlib library. This is the default.

- -DWITHOUT_SERVER=*bool*

  Whether to build without the MySQL server. The default is OFF, which does build the server.

Compiler Flags

- -DCMAKE_C_FLAGS="*flags*"

  Flags for the C Compiler.

- -DCMAKE_CXX_FLAGS="*flags*"

  Flags for the C++ Compiler.

- -DWITH_DEFAULT_COMPILER_OPTIONS=*bool*

  Whether to use the flags from cmake/build_configurations/compiler_options.cmake.

> **Note**
>
> All optimization flags were carefully chosen and tested by the MySQL build team. Overriding them can lead to unexpected results and is done at your own risk.

To specify your own C and C++ compiler flags, for flags that do not affect optimization, use the CMAKE_C_FLAGS and CMAKE_CXX_FLAGS CMake options.

When providing your own compiler flags, you might want to specify CMAKE_BUILD_TYPE as well.

For example, to create a 32-bit release build on a 64-bit Linux machine, do this:

```
shell> mkdir bld
shell> cd bld
shell> cmake .. -DCMAKE_C_FLAGS=-m32 \
         -DCMAKE_CXX_FLAGS=-m32 \
         -DCMAKE_BUILD_TYPE=RelWithDebInfo
```

If you set flags that affect optimization (`-Onumber`), you must set the `CMAKE_C_FLAGS_build_type` and/or `CMAKE_CXX_FLAGS_build_type` options, where `build_type` corresponds to the `CMAKE_BUILD_TYPE` value. To specify a different optimization for the default build type (`RelWithDebInfo`) set the `CMAKE_C_FLAGS_RELWITHDEBINFO` and `CMAKE_CXX_FLAGS_RELWITHDEBINFO` options. For example, to compile on Linux with `-O3` and with debug symbols, do this:

```
shell> cmake .. -DCMAKE_C_FLAGS_RELWITHDEBINFO="-O3 -g" \
         -DCMAKE_CXX_FLAGS_RELWITHDEBINFO="-O3 -g"
```

2.8.5 Dealing with Problems Compiling MySQL

The solution to many problems involves reconfiguring. If you do reconfigure, take note of the following:

- If `CMake` is run after it has previously been run, it may use information that was gathered during its previous invocation. This information is stored in `CMakeCache.txt`. When `CMake` starts up, it looks for that file and reads its contents if it exists, on the assumption that the information is still correct. That assumption is invalid when you reconfigure.
- Each time you run `CMake`, you must run `make` again to recompile. However, you may want to remove old object files from previous builds first because they were compiled using different configuration options.

To prevent old object files or configuration information from being used, run the following commands before re-running `CMake`:

On Unix:

```
shell> make clean
shell> rm CMakeCache.txt
```

On Windows:

```
shell> devenv MySQL.sln /clean
shell> del CMakeCache.txt
```

If you build outside of the source tree, remove and recreate your build directory before re-running `CMake`. For instructions on building outside of the source tree, see How to Build MySQL Server with CMake.

On some systems, warnings may occur due to differences in system include files. The following list describes other problems that have been found to occur most often when compiling MySQL:

- To define which C and C++ compilers to use, you can define the `CC` and `CXX` environment variables. For example:

  ```
  shell> CC=gcc
  shell> CXX=g++
  shell> export CC CXX
  ```

  To specify your own C and C++ compiler flags, use the `CMAKE_C_FLAGS` and `CMAKE_CXX_FLAGS` CMake options. See Compiler Flags.

To see what flags you might need to specify, invoke mysql_config with the --cflags and --cxxflags options.

- To see what commands are executed during the compile stage, after using CMake to configure MySQL, run make VERBOSE=1 rather than just make.
- If compilation fails, check whether the MYSQL_MAINTAINER_MODE option is enabled. This mode causes compiler warnings to become errors, so disabling it may enable compilation to proceed.
- If your compile fails with errors such as any of the following, you must upgrade your version of make to GNU make:

  ```
  make: Fatal error in reader: Makefile, line 18:
  Badly formed macro assignment
  ```

  Or:

  ```
  make: file `Makefile' line 18: Must be a separator (:
  ```

  Or:

  ```
  pthread.h: No such file or directory
  ```

  Solaris and FreeBSD are known to have troublesome make programs.

  GNU make 3.75 is known to work.
- The sql_yacc.cc file is generated from sql_yacc.yy. Normally, the build process does not need to create sql_yacc.cc because MySQL comes with a pregenerated copy. However, if you do need to re-create it, you might encounter this error:

  ```
  "sql_yacc.yy", line xxx fatal: default action causes potential...
  ```

  This is a sign that your version of yacc is deficient. You probably need to install a recent version of bison (the GNU version of yacc) and use that instead.

  Versions of bison older than 1.75 may report this error:

  ```
  sql_yacc.yy:#####: fatal error: maximum table size (32767) exceeded
  ```

  The maximum table size is not actually exceeded; the error is caused by bugs in older versions of bison.

For information about acquiring or updating tools, see the system requirements in Section 2.8, “Installing MySQL from Source”.

2.8.6 MySQL Configuration and Third-Party Tools

Third-party tools that need to determine the MySQL version from the MySQL source can read the VERSION file in the top-level source directory. The file lists the pieces of the version separately. For example, if the version is MySQL 5.7.4-m14, the file looks like this:

```
MYSQL_VERSION_MAJOR=5
MYSQL_VERSION_MINOR=7
MYSQL_VERSION_PATCH=4
MYSQL_VERSION_EXTRA=-m14
```

If the source is not for a General Availablility (GA) release, the MYSQL_VERSION_EXTRA value will be nonempty. For the example, the value corresponds to Milestone 14.

To construct a five-digit number from the version components, use this formula:

```

MYSQL_VERSION_MAJOR*10000 + MYSQL_VERSION_MINOR*100 + MYSQL_VERSION_PATCH
```

2.9 Postinstallation Setup and Testing

This section discusses post-installation items for Unix-like systems. If you are using Windows, see Section 2.3.8, “Windows Postinstallation Procedures”.

After installing MySQL, there are some items that you should address. For example:

- You should initialize the data directory and create the MySQL grant tables, as describe in Section 2.9.1, “Postinstallation Procedures for Unix-like Systems”.
- An important security concern is that the initial accounts in the grant tables have no passwords. You should assign passwords to prevent unauthorized access to the MySQL server. For instructions, see Section 2.9.2, “Securing the Initial MySQL Accounts”.
- Optionally, you can create time zone tables to enable recognition of named time zones. For instructions, see Section 4.4.6, “mysql_tzinfo_to_sql — Load the Time Zone Tables”.
- If you have trouble getting the server to start, see Section 2.9.1.3, “Starting and Troubleshooting the MySQL Server”.
- When you are ready to create additional user accounts, you can find information on the MySQL access control system and account management in Section 6.2, “The MySQL Access Privilege System”, and Section 6.3, “MySQL User Account Management”.

2.9.1 Postinstallation Procedures for Unix-like Systems

After installing MySQL on a Unix-like system, you must initialize the grant tables, start the server, and make sure that the server works satisfactorily. You may also wish to arrange for the server to be started and stopped automatically when your system starts and stops. You should also assign passwords to the accounts in the grant tables.

On a Unix-like system, the grant tables are set up by the mysql_install_db program. For some installation methods, this program is run for you automatically if an existing database cannot be found.

- If you install MySQL on Linux using RPM distributions, the server RPM runs mysql_install_db.
- Using the native packaging system on many platforms, including Debian Linux, Ubuntu Linux, Gentoo Linux and others, the mysql_install_db command is run for you.
- If you install MySQL on Mac OS X using a DMG distribution, the installer runs mysql_install_db.

For other platforms and installation types, including generic binary and source installs, you will need to run mysql_install_db yourself.

The following procedure describes how to initialize the grant tables (if that has not previously been done) and start the server. It also suggests some commands that you can use to test whether the server is accessible and working properly. For information about starting and stopping the server automatically, see Section 2.9.1.2, “Starting and Stopping MySQL Automatically”.

After you complete the procedure and have the server running, you should assign passwords to the accounts created by mysql_install_db and perhaps restrict access to test databases. For instructions, see Section 2.9.2, “Securing the Initial MySQL Accounts”.

In the examples shown here, the server runs under the user ID of the mysql login account. This assumes that such an account exists. Either create the account if it does not exist, or substitute the name of a different existing login account that you plan to use for running the server. For information about creating the account, see Creating a mysql System User and Group, in Section 2.2, “Installing MySQL on Unix/Linux Using Generic Binaries”.

1. Change location into the top-level directory of your MySQL installation, represented here by *BASEDIR*:

   ```
   shell> cd BASEDIR
   ```

   BASEDIR is the installation directory for your MySQL instance. It is likely to be something like /usr/local/mysql or /usr/local. The following steps assume that you have changed location to this directory.

   You will find several files and subdirectories in the *BASEDIR* directory. The most important for installation purposes are the bin and scripts subdirectories:

   - The bin directory contains client programs and the server. You should add the full path name of this directory to your PATH environment variable so that your shell finds the MySQL programs properly. See Section 2.11, “Environment Variables”.

   - The scripts directory contains the mysql_install_db script used to initialize the mysql database containing the grant tables that store the server access permissions.

2. If necessary, ensure that the distribution contents are accessible to mysql. If you installed the distribution as mysql, no further action is required. If you installed the distribution as root, its contents will be owned by root. Change its ownership to mysql by executing the following commands as root in the installation directory. The first command changes the owner attribute of the files to the mysql user. The second changes the group attribute to the mysql group.

   ```
   shell> chown -R mysql .
   shell> chgrp -R mysql .
   ```

3. If necessary, run the mysql_install_db program to set up the initial MySQL grant tables containing the privileges that determine how users are permitted to connect to the server. You will need to do this if you used a distribution type for which the installation procedure does not run the program for you.

   ```
   shell> scripts/mysql_install_db --user=mysql
   ```

   Typically, mysql_install_db needs to be run only the first time you install MySQL, so you can skip this step if you are upgrading an existing installation, However, mysql_install_db does not overwrite any existing privilege tables, so it should be safe to run in any circumstances.

   It might be necessary to specify other options such as --basedir or --datadir if mysql_install_db does not identify the correct locations for the installation directory or data directory. For example:

```
shell> scripts/mysql_install_db --user=mysql \
         --basedir=/opt/mysql/mysql \
         --datadir=/opt/mysql/mysql/data
```

The `mysql_install_db` script creates the server's data directory with `mysql` as the owner. Under the data directory, it creates directories for the `mysql` database that holds the grant tables and the `test` database that you can use to test MySQL. The script also creates privilege table entries for `root` and anonymous-user accounts. The accounts have no passwords initially. Section 2.9.2, “Securing the Initial MySQL Accounts”, describes the initial privileges. Briefly, these privileges permit the MySQL `root` user to do anything, and permit anybody to create or use databases with a name of `test` or starting with `test_`. See Section 6.2, “The MySQL Access Privilege System”, for a complete listing and description of the grant tables.

For a more secure installation, invoke `mysql_install_db` with the `--random-passwords` option. This causes it to assign a random password to the MySQL `root` accounts, set the “password expired” flag for those accounts, and remove the anonymous-user MySQL accounts. For additional details, see Section 4.4.3, “`mysql_install_db` — Initialize MySQL Data Directory”. (Install operations using RPMs for Unbreakable Linux Network are unaffected because they do not use `mysql_install_db`.)

It is important to make sure that the database directories and files are owned by the `mysql` login account so that the server has read and write access to them when you run it later. To ensure this if you run `mysql_install_db` as `root`, include the `--user` option as shown. Otherwise, you should execute the script while logged in as `mysql`, in which case you can omit the `--user` option from the command.

If you do not want to have the `test` database, you can remove it after starting the server, using the instructions in Section 2.9.2, “Securing the Initial MySQL Accounts”.

If you have trouble with `mysql_install_db` at this point, see Section 2.9.1.1, “Problems Running `mysql_install_db`”.

4. Most of the MySQL installation can be owned by `root` if you like. The exception is that the data directory must be owned by `mysql`. To accomplish this, run the following commands as `root` in the installation directory:

```
shell> chown -R root .
shell> chown -R mysql data
```

5. If the plugin directory (the directory named by the `plugin_dir` system variable) is writable by the server, it may be possible for a user to write executable code to a file in the directory using `SELECT ... INTO DUMPFILE`. This can be prevented by making `plugin_dir` read only to the server or by setting `--secure-file-priv` to a directory where `SELECT` writes can be made safely.

6. If you installed MySQL using a source distribution, you may want to optionally copy one of the provided configuration files from the `support-files` directory into your `/etc` directory. There are different sample configuration files for different use cases, server types, and CPU and RAM configurations. If you want to use one of these standard files, you should copy it to `/etc/my.cnf`, or `/etc/mysql/my.cnf` and edit and check the configuration before starting your MySQL server for the first time.

   If you do not copy one of the standard configuration files, the MySQL server will be started with the default settings.

   If you want MySQL to start automatically when you boot your machine, you can copy `support-files/mysql.server` to the location where your system has its startup files. More information can be found in the `mysql.server` script itself, and in Section 2.9.1.2, “Starting and Stopping MySQL Automatically”.

7. Start the MySQL server:

   ```
   shell> bin/mysqld_safe --user=mysql &
   ```

   It is important that the MySQL server be run using an unprivileged (non-`root`) login account. To ensure this if you run `mysqld_safe` as `root`, include the `--user` option as shown. Otherwise, you should execute the script while logged in as `mysql`, in which case you can omit the `--user` option from the command.

   For further instructions for running MySQL as an unprivileged user, see Section 6.1.5, “How to Run MySQL as a Normal User”.

   If the command fails immediately and prints `mysqld ended`, look for information in the error log (which by default is the *`host_name`*`.err` file in the data directory).

   If you neglected to create the grant tables by running `mysql_install_db` before proceeding to this step, the following message appears in the error log file when you start the server:

   ```
   mysqld: Can't find file: 'host.frm'
   ```

   This error also occurs if you run `mysql_install_db` as `root` without the `--user` option. Remove the `data` directory and run `mysql_install_db` with the `--user` option as described previously.

   If you have other problems starting the server, see Section 2.9.1.3, “Starting and Troubleshooting the MySQL Server”. For more information about `mysqld_safe`, see Section 4.3.2, “`mysqld_safe` — MySQL Server Startup Script”.

8. Use `mysqladmin` to verify that the server is running. The following commands provide simple tests to check whether the server is up and responding to connections:

   ```
   shell> bin/mysqladmin version
   shell> bin/mysqladmin variables
   ```

   The output from `mysqladmin version` varies slightly depending on your platform and version of MySQL, but should be similar to that shown here:

   ```
   shell> bin/mysqladmin version
   mysqladmin  Ver 14.12 Distrib 5.7.5, for pc-linux-gnu on i686
   ...

   Server version          5.7.5
   Protocol version        10
   Connection              Localhost via UNIX socket
   UNIX socket             /var/lib/mysql/mysql.sock
   Uptime:                 14 days 5 hours 5 min 21 sec

   Threads: 1  Questions: 366  Slow queries: 0
   Opens: 0  Flush tables: 1  Open tables: 19
   Queries per second avg: 0.000
   ```

   To see what else you can do with `mysqladmin`, invoke it with the `--help` option.

9. Verify that you can shut down the server:

   ```
   shell> bin/mysqladmin -u root shutdown
   ```

10. Verify that you can start the server again. Do this by using `mysqld_safe` or by invoking `mysqld` directly. For example:

    ```
    shell> bin/mysqld_safe --user=mysql &
    ```

    If `mysqld_safe` fails, see Section 2.9.1.3, “Starting and Troubleshooting the MySQL Server”.

11. Run some simple tests to verify that you can retrieve information from the server. The output should be similar to what is shown here:

    ```
    shell> bin/mysqlshow
    +--------------------+
    |     Databases      |
    +--------------------+
    | information_schema |
    | mysql              |
    | test               |
    +--------------------+

    shell> bin/mysqlshow mysql
    Database: mysql
    +---------------------------+
    |          Tables           |
    +---------------------------+
    | columns_priv              |
    | db                        |
    | event                     |
    | func                      |
    | help_category             |
    | help_keyword              |
    | help_relation             |
    | help_topic                |
    | host                      |
    | plugin                    |
    | proc                      |
    | procs_priv                |
    | servers                   |
    | tables_priv               |
    | time_zone                 |
    | time_zone_leap_second     |
    | time_zone_name            |
    | time_zone_transition      |
    | time_zone_transition_type |
    | user                      |
    +---------------------------+

    shell> bin/mysql -e "SELECT Host,Db,User FROM db" mysql
    +------+--------+------+
    | host | db     | user |
    +------+--------+------+
    | %    | test   |      |
    | %    | test_% |      |
    +------+--------+------+
    ```

12. There is a benchmark suite in the `sql-bench` directory (under the MySQL installation directory) that you can use to compare how MySQL performs on different platforms. The benchmark suite is written in Perl. It requires the Perl DBI module that provides a database-independent interface to the various databases, and some other additional Perl modules:

    ```
    DBI
    DBD::mysql
    Data::Dumper
    ```

```
Data::ShowTable
```

These modules can be obtained from CPAN (http://www.cpan.org/). See also Section 2.12.1, “Installing Perl on Unix”.

The `sql-bench/Results` directory contains the results from many runs against different databases and platforms. To run all tests, execute these commands:

```
shell> cd sql-bench
shell> perl run-all-tests
```

If you do not have the `sql-bench` directory, you probably installed MySQL using RPM files other than the source RPM. (The source RPM includes the `sql-bench` benchmark directory.) In this case, you must first install the benchmark suite before you can use it. There are separate benchmark RPM files named `mysql-bench-VERSION.i386.rpm` that contain benchmark code and data.

If you have a source distribution, there are also tests in its `tests` subdirectory that you can run. For example, to run `auto_increment.tst`, execute this command from the top-level directory of your source distribution:

```
shell> mysql -vvf test < ./tests/auto_increment.tst
```

The expected result of the test can be found in the `./tests/auto_increment.res` file.

13. At this point, you should have the server running. However, none of the initial MySQL accounts have a password, and the server permits permissive access to test databases. To tighten security, follow the instructions in Section 2.9.2, “Securing the Initial MySQL Accounts”.

The MySQL 5.7 installation procedure creates time zone tables in the `mysql` database but does not populate them. To do so, use the instructions in Section 10.6, “MySQL Server Time Zone Support”.

To make it more convenient to invoke programs installed in the `bin` directory under the installation directory, you can add that directory to your `PATH` environment variable setting. That enables you to run a program by typing only its name, not its entire path name. See Section 4.2.4, “Setting Environment Variables”.

2.9.1.1 Problems Running `mysql_install_db`

The purpose of the `mysql_install_db` script is to generate new MySQL privilege tables. It does not overwrite existing MySQL privilege tables, and it does not affect any other data.

If you want to re-create your privilege tables, first stop the `mysqld` server if it is running. Then rename the `mysql` directory under the data directory to save it, and then run `mysql_install_db`. Suppose that your current directory is the MySQL installation directory and that `mysql_install_db` is located in the `bin` directory and the data directory is named `data`. To rename the `mysql` database and re-run `mysql_install_db`, use these commands.

```
shell> mv data/mysql data/mysql.old
shell> scripts/mysql_install_db --user=mysql
```

When you run `mysql_install_db`, you might encounter the following problems:

- **`mysql_install_db` fails to install the grant tables**

  You may find that `mysql_install_db` fails to install the grant tables and terminates after displaying the following messages:

```
Starting mysqld daemon with databases from XXXXXX
mysqld ended
```

In this case, you should examine the error log file very carefully. The log should be located in the directory `XXXXXX` named by the error message and should indicate why `mysqld` did not start. If you do not understand what happened, include the log when you post a bug report. See Section 1.7, “How to Report Bugs or Problems”.

- **There is a `mysqld` process running**

  This indicates that the server is running, in which case the grant tables have probably been created already. If so, there is no need to run `mysql_install_db` at all because it needs to be run only once (when you install MySQL the first time).

- **Installing a second `mysqld` server does not work when one server is running**

  This can happen when you have an existing MySQL installation, but want to put a new installation in a different location. For example, you might have a production installation, but you want to create a second installation for testing purposes. Generally the problem that occurs when you try to run a second server is that it tries to use a network interface that is in use by the first server. In this case, you should see one of the following error messages:

  ```
  Can't start server: Bind on TCP/IP port:
  Address already in use
  Can't start server: Bind on unix socket...
  ```

  For instructions on setting up multiple servers, see Section 5.3, “Running Multiple MySQL Instances on One Machine”.

- **You do not have write access to the `/tmp` directory**

  If you do not have write access to create temporary files or a Unix socket file in the default location (the `/tmp` directory) or the `TMP_DIR` environment variable, if it has been set, an error occurs when you run `mysql_install_db` or the `mysqld` server.

  You can specify different locations for the temporary directory and Unix socket file by executing these commands prior to starting `mysql_install_db` or `mysqld`, where *`some_tmp_dir`* is the full path name to some directory for which you have write permission:

  ```
  shell> TMPDIR=/some_tmp_dir/
  shell> MYSQL_UNIX_PORT=/some_tmp_dir/mysql.sock
  shell> export TMPDIR MYSQL_UNIX_PORT
  ```

  Then you should be able to run `mysql_install_db` and start the server with these commands:

  ```
  shell> scripts/mysql_install_db --user=mysql
  shell> bin/mysqld_safe --user=mysql &
  ```

  If `mysql_install_db` is located in the `scripts` directory, modify the first command to `scripts/mysql_install_db`.

  See Section C.5.4.5, “How to Protect or Change the MySQL Unix Socket File”, and Section 2.11, “Environment Variables”.

There are some alternatives to running the `mysql_install_db` script provided in the MySQL distribution:

- If you want the initial privileges to be different from the standard defaults, you can modify mysql_install_db before you run it. However, it is preferable to use GRANT and REVOKE to change the privileges *after* the grant tables have been set up. In other words, you can run mysql_install_db, and then use mysql -u root mysql to connect to the server as the MySQL root user so that you can issue the necessary GRANT and REVOKE statements.

  If you want to install MySQL on several machines with the same privileges, you can put the GRANT and REVOKE statements in a file and execute the file as a script using mysql after running mysql_install_db. For example:

  ```
  shell> scripts/mysql_install_db --user=mysql
  shell> bin/mysql -u root < your_script_file
  ```

  By doing this, you can avoid having to issue the statements manually on each machine.

- It is possible to re-create the grant tables completely after they have previously been created. You might want to do this if you are just learning how to use GRANT and REVOKE and have made so many modifications after running mysql_install_db that you want to wipe out the tables and start over.

  To re-create the grant tables, remove all the .frm, .MYI, and .MYD files in the mysql database directory. Then run the mysql_install_db script again.

- You can start mysqld manually using the --skip-grant-tables option and add the privilege information yourself using mysql:

  ```
  shell> bin/mysqld_safe --user=mysql --skip-grant-tables &
  shell> bin/mysql mysql
  ```

  From mysql, manually execute the SQL commands contained in mysql_install_db. Make sure that you run mysqladmin flush-privileges or mysqladmin reload afterward to tell the server to reload the grant tables.

  Note that by not using mysql_install_db, you not only have to populate the grant tables manually, you also have to create them first.

2.9.1.2 Starting and Stopping MySQL Automatically

Generally, you start the mysqld server in one of these ways:

- Invoke mysqld directly. This works on any platform.

- Invoke mysqld_safe, which tries to determine the proper options for mysqld and then runs it with those options. This script is used on Unix and Unix-like systems. See Section 4.3.2, “mysqld_safe — MySQL Server Startup Script”.

- Invoke mysql.server. This script is used primarily at system startup and shutdown on systems that use System V-style run directories (that is, /etc/init.d and run-level specific directories), where it usually is installed under the name mysql. The mysql.server script starts the server by invoking mysqld_safe. See Section 4.3.3, “mysql.server — MySQL Server Startup Script”.

- On Mac OS X, install a separate MySQL Startup Item package to enable the automatic startup of MySQL on system startup. The Startup Item starts the server by invoking mysql.server. See Section 2.4.3, “Installing the MySQL Startup Item”, for details. A MySQL Preference Pane also provides control for starting and stopping MySQL through the System Preferences, see Section 2.4.4, “Installing and Using the MySQL Preference Pane”.

- Use the Solaris/OpenSolaris service management framework (SMF) system to initiate and control MySQL startup. For more information, see Section 2.6.2, “Installing MySQL on OpenSolaris Using IPS”.

The mysqld_safe and mysql.server scripts, Solaris/OpenSolaris SMF, and the Mac OS X Startup Item (or MySQL Preference Pane) can be used to start the server manually, or automatically at system startup time. mysql.server and the Startup Item also can be used to stop the server.

To start or stop the server manually using the mysql.server script, invoke it with start or stop arguments:

```
shell> mysql.server start
shell> mysql.server stop
```

Before mysql.server starts the server, it changes location to the MySQL installation directory, and then invokes mysqld_safe. If you want the server to run as some specific user, add an appropriate user option to the [mysqld] group of the /etc/my.cnf option file, as shown later in this section. (It is possible that you will need to edit mysql.server if you've installed a binary distribution of MySQL in a nonstandard location. Modify it to change location into the proper directory before it runs mysqld_safe. If you do this, your modified version of mysql.server may be overwritten if you upgrade MySQL in the future, so you should make a copy of your edited version that you can reinstall.)

mysql.server stop stops the server by sending a signal to it. You can also stop the server manually by executing mysqladmin shutdown.

To start and stop MySQL automatically on your server, you need to add start and stop commands to the appropriate places in your /etc/rc* files.

If you use the Linux server RPM package (MySQL-server-*VERSION*.rpm), or a native Linux package installation, the mysql.server script may be installed in the /etc/init.d directory with the name mysql. See Section 2.5.3, “Installing MySQL on Linux Using RPM Packages”, for more information on the Linux RPM packages.

Some vendors provide RPM packages that install a startup script under a different name such as mysqld.

If you install MySQL from a source distribution or using a binary distribution format that does not install mysql.server automatically, you can install it manually. The script can be found in the support-files directory under the MySQL installation directory or in a MySQL source tree.

To install mysql.server manually, copy it to the /etc/init.d directory with the name mysql, and then make it executable. Do this by changing location into the appropriate directory where mysql.server is located and executing these commands:

```
shell> cp mysql.server /etc/init.d/mysql
shell> chmod +x /etc/init.d/mysql
```

Note

Older Red Hat systems use the /etc/rc.d/init.d directory rather than /etc/init.d. Adjust the preceding commands accordingly. Alternatively, first create /etc/init.d as a symbolic link that points to /etc/rc.d/init.d:

```
shell> cd /etc
shell> ln -s rc.d/init.d .
```

After installing the script, the commands needed to activate it to run at system startup depend on your operating system. On Linux, you can use chkconfig:

```
shell> chkconfig --add mysql
```

On some Linux systems, the following command also seems to be necessary to fully enable the `mysql` script:

```
shell> chkconfig --level 345 mysql on
```

On FreeBSD, startup scripts generally should go in `/usr/local/etc/rc.d/`. The `rc(8)` manual page states that scripts in this directory are executed only if their basename matches the `*.sh` shell file name pattern. Any other files or directories present within the directory are silently ignored. In other words, on FreeBSD, you should install the `mysql.server` script as `/usr/local/etc/rc.d/mysql.server.sh` to enable automatic startup.

As an alternative to the preceding setup, some operating systems also use `/etc/rc.local` or `/etc/init.d/boot.local` to start additional services on startup. To start up MySQL using this method, you could append a command like the one following to the appropriate startup file:

```
/bin/sh -c 'cd /usr/local/mysql; ./bin/mysqld_safe --user=mysql &'
```

For other systems, consult your operating system documentation to see how to install startup scripts.

You can add options for `mysql.server` in a global `/etc/my.cnf` file. A typical `/etc/my.cnf` file might look like this:

```
[mysqld]
datadir=/usr/local/mysql/var
socket=/var/tmp/mysql.sock
port=3306
user=mysql

[mysql.server]
basedir=/usr/local/mysql
```

The `mysql.server` script supports the following options: `basedir`, `datadir`, and `pid-file`. If specified, they *must* be placed in an option file, not on the command line. `mysql.server` supports only `start` and `stop` as command-line arguments.

The following table shows which option groups the server and each startup script read from option files.

Table 2.14 MySQL Startup scripts and supported server option groups

Script	Option Groups
`mysqld`	`[mysqld]`, `[server]`, `[mysqld-*major_version*]`
`mysqld_safe`	`[mysqld]`, `[server]`, `[mysqld_safe]`
`mysql.server`	`[mysqld]`, `[mysql.server]`, `[server]`

`[mysqld-*major_version*]` means that groups with names like `[mysqld-5.6]` and `[mysqld-5.7]` are read by servers having versions 5.6.x, 5.7.x, and so forth. This feature can be used to specify options that can be read only by servers within a given release series.

For backward compatibility, `mysql.server` also reads the `[mysql_server]` group and `mysqld_safe` also reads the `[safe_mysqld]` group. However, you should update your option files to use the `[mysql.server]` and `[mysqld_safe]` groups instead when using MySQL 5.7.

For more information on MySQL configuration files and their structure and contents, see Section 4.2.3.3, “Using Option Files”.

2.9.1.3 Starting and Troubleshooting the MySQL Server

This section provides troubleshooting suggestions for problems starting the server on a Unix-like system. If you are using Windows, see Section 2.3.6, "Troubleshooting a Microsoft Windows MySQL Server Installation".

If you have problems starting the server, here are some things to try:

- Check the error log to see why the server does not start. Log files are located in the data directory (typically `C:\Program Files\MySQL\MySQL Server 5.7\data` on Windows, `/usr/local/mysql/data` for a Unix/Linux binary distribution, and `/usr/local/var` for a Unix/Linux source distribution). Look in the data directory for files with names of the form *`host_name`*`.err` and *`host_name`*`.log`, where *`host_name`* is the name of your server host. Then examine the last few lines of these files. Use `tail` to display them:

  ```
  shell> tail host_name.err
  shell> tail host_name.log
  ```

- Specify any special options needed by the storage engines you are using. You can create a `my.cnf` file and specify startup options for the engines that you plan to use. If you are going to use storage engines that support transactional tables (`InnoDB`, `NDB`), be sure that you have them configured the way you want before starting the server. If you are using `InnoDB` tables, see Section 14.2.3, "`InnoDB` Configuration" for guidelines and Section 14.2.13, "`InnoDB` Startup Options and System Variables" for option syntax.

  Although storage engines use default values for options that you omit, Oracle recommends that you review the available options and specify explicit values for any options whose defaults are not appropriate for your installation.

- Make sure that the server knows where to find the data directory. The `mysqld` server uses this directory as its current directory. This is where it expects to find databases and where it expects to write log files. The server also writes the pid (process ID) file in the data directory.

  The default data directory location is hardcoded when the server is compiled. To determine what the default path settings are, invoke `mysqld` with the `--verbose` and `--help` options. If the data directory is located somewhere else on your system, specify that location with the `--datadir` option to `mysqld` or `mysqld_safe`, on the command line or in an option file. Otherwise, the server will not work properly. As an alternative to the `--datadir` option, you can specify `mysqld` the location of the base directory under which MySQL is installed with the `--basedir`, and `mysqld` looks for the `data` directory there.

  To check the effect of specifying path options, invoke `mysqld` with those options followed by the `--verbose` and `--help` options. For example, if you change location into the directory where `mysqld` is installed and then run the following command, it shows the effect of starting the server with a base directory of `/usr/local`:

  ```
  shell> ./mysqld --basedir=/usr/local --verbose --help
  ```

  You can specify other options such as `--datadir` as well, but `--verbose` and `--help` must be the last options.

  Once you determine the path settings you want, start the server without `--verbose` and `--help`.

  If `mysqld` is currently running, you can find out what path settings it is using by executing this command:

  ```
  shell> mysqladmin variables
  ```

Or:

```
shell> mysqladmin -h host_name variables
```

host_name is the name of the MySQL server host.

- Make sure that the server can access the data directory. The ownership and permissions of the data directory and its contents must allow the server to read and modify them.

  If you get `Errcode 13` (which means `Permission denied`) when starting `mysqld`, this means that the privileges of the data directory or its contents do not permit server access. In this case, you change the permissions for the involved files and directories so that the server has the right to use them. You can also start the server as `root`, but this raises security issues and should be avoided.

  Change location into the data directory and check the ownership of the data directory and its contents to make sure the server has access. For example, if the data directory is `/usr/local/mysql/var`, use this command:

  ```
  shell> ls -la /usr/local/mysql/var
  ```

  If the data directory or its files or subdirectories are not owned by the login account that you use for running the server, change their ownership to that account. If the account is named `mysql`, use these commands:

  ```
  shell> chown -R mysql /usr/local/mysql/var
  shell> chgrp -R mysql /usr/local/mysql/var
  ```

  Even with correct ownership, MySQL might fail to start up if there is other security software running on your system that manages application access to various parts of the file system. In this case, reconfigure that software to enable `mysqld` to access the directories it uses during normal operation.

- Verify that the network interfaces the server wants to use are available.

  If either of the following errors occur, it means that some other program (perhaps another `mysqld` server) is using the TCP/IP port or Unix socket file that `mysqld` is trying to use:

  ```
  Can't start server: Bind on TCP/IP port: Address already in use
  Can't start server: Bind on unix socket...
  ```

  Use `ps` to determine whether you have another `mysqld` server running. If so, shut down the server before starting `mysqld` again. (If another server is running, and you really want to run multiple servers, you can find information about how to do so in Section 5.3, “Running Multiple MySQL Instances on One Machine”.)

  If no other server is running, execute the command `telnet` *`your_host_name`* *`tcp_ip_port_number`*. (The default MySQL port number is 3306.) Then press Enter a couple of times. If you do not get an error message like `telnet: Unable to connect to remote host: Connection refused`, some other program is using the TCP/IP port that `mysqld` is trying to use. Track down what program this is and disable it, or tell `mysqld` to listen to a different port with the `--port` option. In this case, specify the same non-default port number for client programs when connecting to the server using TCP/IP.

  Another reason the port might be inaccessible is that you have a firewall running that blocks connections to it. If so, modify the firewall settings to permit access to the port.

If the server starts but you cannot connect to it, make sure that you have an entry in /etc/hosts that looks like this:

```
127.0.0.1       localhost
```

- If you cannot get mysqld to start, try to make a trace file to find the problem by using the --debug option. See Section 22.4.3, “The DBUG Package”.

2.9.2 Securing the Initial MySQL Accounts

Part of the MySQL installation process is to set up the mysql database that contains the grant tables:

- Windows distributions contain preinitialized grant tables.
- On Unix, the mysql_install_db program populates the grant tables. Some installation methods run this program for you. Others require that you execute it manually. For details, see Section 2.9.1, “Postinstallation Procedures for Unix-like Systems”.

The mysql.user grant table defines the initial MySQL user accounts and their access privileges:

- Some accounts have the user name root. These are superuser accounts that have all privileges and can do anything. The initial root account passwords are empty, so anyone can connect to the MySQL server as root *without a password* and be granted all privileges.
  - On Windows, root accounts are created that permit connections from the local host only. Connections can be made by specifying the host name localhost, the IP address 127.0.0.1, or the IPv6 address ::1. If the user selects the **Enable root access from remote machines** option during installation, the Windows installer creates another root account that permits connections from any host.
  - On Unix, each root account permits connections from the local host. Connections can be made by specifying the host name localhost, the IP address 127.0.0.1, the IPv6 address ::1, or the actual host name or IP address.

  An attempt to connect to the host 127.0.0.1 normally resolves to the localhost account. However, this fails if the server is run with the --skip-name-resolve option, so the 127.0.0.1 account is useful in that case. The ::1 account is used for IPv6 connections.

- Some accounts are for anonymous users. These have an empty user name. The anonymous accounts have no password, so anyone can use them to connect to the MySQL server.
  - On Windows, there is one anonymous account that permits connections from the local host. Connections can be made by specifying a host name of localhost.
  - On Unix, each anonymous account permits connections from the local host. Connections can be made by specifying a host name of localhost for one of the accounts, or the actual host name or IP address for the other.

To display which accounts exist in the mysql.user table and check whether their passwords are empty, use the following statement:

```
mysql> SELECT User, Host, Password FROM mysql.user;
+------+--------------------+----------+
| User | Host               | Password |
+------+--------------------+----------+
| root | localhost          |          |
| root | myhost.example.com |          |
| root | 127.0.0.1          |          |
| root | ::1                |          |
|      | localhost          |          |
|      | myhost.example.com |          |
+------+--------------------+----------+
```

This output indicates that there are several `root` and anonymous-user accounts, none of which have passwords. The output might differ on your system, but the presence of accounts with empty passwords means that your MySQL installation is unprotected until you do something about it:

- You should assign a password to each MySQL `root` account.
- If you want to prevent clients from connecting as anonymous users without a password, you should either assign a password to each anonymous account or else remove the accounts.

In addition, the `mysql.db` table contains rows that permit all accounts to access the `test` database and other databases with names that start with `test_`. This is true even for accounts that otherwise have no special privileges such as the default anonymous accounts. This is convenient for testing but inadvisable on production servers. Administrators who want database access restricted only to accounts that have permissions granted explicitly for that purpose should remove these `mysql.db` table rows.

The following instructions describe how to set up passwords for the initial MySQL accounts, first for the `root` accounts, then for the anonymous accounts. The instructions also cover how to remove the anonymous accounts, should you prefer not to permit anonymous access at all, and describe how to remove permissive access to test databases. Replace *`newpwd`* in the examples with the password that you want to use. Replace *`host_name`* with the name of the server host. You can determine this name from the output of the preceding `SELECT` statement. For the output shown, *`host_name`* is `myhost.example.com`.

> **Note**
>
> For additional information about setting passwords, see Section 6.3.5, “Assigning Account Passwords”. If you forget your `root` password after setting it, see Section C.5.4.1, “How to Reset the Root Password”.

You might want to defer setting the passwords until later, to avoid the need to specify them while you perform additional setup or testing. However, be sure to set them before using your installation for production purposes.

To set up additional accounts, see Section 6.3.2, “Adding User Accounts”.

Assigning `root` Account Passwords

The `root` account passwords can be set several ways. The following discussion demonstrates three methods:

- Use the `SET PASSWORD` statement
- Use the `UPDATE` statement
- Use the `mysqladmin` command-line client program

To assign passwords using `SET PASSWORD`, connect to the server as `root` and issue a `SET PASSWORD` statement for each `root` account listed in the `mysql.user` table. Be sure to encrypt the password using the `PASSWORD()` function.

For Windows, do this:

```
shell> mysql -u root
mysql> SET PASSWORD FOR 'root'@'localhost' = PASSWORD('newpwd');
mysql> SET PASSWORD FOR 'root'@'127.0.0.1' = PASSWORD('newpwd');
mysql> SET PASSWORD FOR 'root'@'::1' = PASSWORD('newpwd');
mysql> SET PASSWORD FOR 'root'@'%' = PASSWORD('newpwd');
```

The last statement is unnecessary if the `mysql.user` table has no `root` account with a host value of `%`.

For Unix, do this:

```
shell> mysql -u root
mysql> SET PASSWORD FOR 'root'@'localhost' = PASSWORD('newpwd');
mysql> SET PASSWORD FOR 'root'@'127.0.0.1' = PASSWORD('newpwd');
mysql> SET PASSWORD FOR 'root'@'::1' = PASSWORD('newpwd');
mysql> SET PASSWORD FOR 'root'@'host_name' = PASSWORD('newpwd');
```

You can also use a single statement that assigns a password to all `root` accounts by using `UPDATE` to modify the `mysql.user` table directly. This method works on any platform:

```
shell> mysql -u root
mysql> UPDATE mysql.user SET Password = PASSWORD('newpwd')
    ->     WHERE User = 'root';
mysql> FLUSH PRIVILEGES;
```

The `FLUSH` statement causes the server to reread the grant tables. Without it, the password change remains unnoticed by the server until you restart it.

To assign passwords to the `root` accounts using `mysqladmin`, execute the following commands:

```
shell> mysqladmin -u root password "newpwd"
shell> mysqladmin -u root -h host_name password "newpwd"
```

Those commands apply both to Windows and to Unix. The double quotation marks around the password are not always necessary, but you should use them if the password contains spaces or other characters that are special to your command interpreter.

The `mysqladmin` method of setting the `root` account passwords does not work for the `'root'@'127.0.0.1'` or `'root'@'::1'` account. Use the `SET PASSWORD` method shown earlier.

After the `root` passwords have been set, you must supply the appropriate password whenever you connect as `root` to the server. For example, to shut down the server with `mysqladmin`, use this command:

```
shell> mysqladmin -u root -p shutdown
Enter password: (enter root password here)
```

Assigning Anonymous Account Passwords

The `mysql` commands in the following instructions include a `-p` option based on the assumption that you have set the `root` account passwords using the preceding instructions and must specify that password when connecting to the server.

To assign passwords to the anonymous accounts, connect to the server as `root`, then use either `SET PASSWORD` or `UPDATE`. Be sure to encrypt the password using the `PASSWORD()` function.

To use `SET PASSWORD` on Windows, do this:

```
shell> mysql -u root -p
Enter password: (enter root password here)
mysql> SET PASSWORD FOR ''@'localhost' = PASSWORD('newpwd');
```

To use SET PASSWORD on Unix, do this:

```
shell> mysql -u root -p
Enter password: (enter root password here)
mysql> SET PASSWORD FOR ''@'localhost' = PASSWORD('newpwd');
mysql> SET PASSWORD FOR ''@'host_name' = PASSWORD('newpwd');
```

To set the anonymous-user account passwords with a single UPDATE statement, do this (on any platform):

```
shell> mysql -u root -p
Enter password: (enter root password here)
mysql> UPDATE mysql.user SET Password = PASSWORD('newpwd')
    ->     WHERE User = '';
mysql> FLUSH PRIVILEGES;
```

The FLUSH statement causes the server to reread the grant tables. Without it, the password change remains unnoticed by the server until you restart it.

Removing Anonymous Accounts

If you prefer to remove any anonymous accounts rather than assigning them passwords, do so as follows on Windows:

```
shell> mysql -u root -p
Enter password: (enter root password here)
mysql> DROP USER ''@'localhost';
```

On Unix, remove the anonymous accounts like this:

```
shell> mysql -u root -p
Enter password: (enter root password here)
mysql> DROP USER ''@'localhost';
mysql> DROP USER ''@'host_name';
```

Securing Test Databases

By default, the mysql.db table contains rows that permit access by any user to the test database and other databases with names that start with test_. (These rows have an empty User column value, which for access-checking purposes matches any user name.) This means that such databases can be used even by accounts that otherwise possess no privileges. If you want to remove any-user access to test databases, do so as follows:

```
shell> mysql -u root -p
Enter password: (enter root password here)
mysql> DELETE FROM mysql.db WHERE Db LIKE 'test%';
mysql> FLUSH PRIVILEGES;
```

The FLUSH statement causes the server to reread the grant tables. Without it, the privilege change remains unnoticed by the server until you restart it.

With the preceding change, only users who have global database privileges or privileges granted explicitly for the test database can use it. However, if you do not want the database to exist at all, drop it:

```
mysql> DROP DATABASE test;
```

Note

On Windows, you can also perform the process described in this section using the Configuration Wizard (see The Security Options Dialog). On all platforms, the MySQL distribution includes mysql_secure_installation, a command-line utility that automates much of the process of securing a MySQL installation.

2.10 Upgrading or Downgrading MySQL

This section describes the steps to upgrade or downgrade a MySQL installation.

Upgrading is a common procedure, as you pick up bug fixes within the same MySQL release series or significant features between major MySQL releases. You perform this procedure first on some test systems to make sure everything works smoothly, and then on the production systems.

Downgrading is less common. Typically, you undo an upgrade because of some compatibility or performance issue that occurs on a production system, and was not uncovered during initial upgrade verification on the test systems. As with the upgrade procedure, perform and verify the downgrade procedure on some test systems first, before using it on a production system.

2.10.1 Upgrading MySQL

As a general rule, to upgrade from one release series to another, go to the next series rather than skipping a series. To upgrade from a release series previous to MySQL 5.6, upgrade to each successive release series in turn until you have reached MySQL 5.6, and then proceed with the upgrade to MySQL 5.7. For example, if you currently are running MySQL 5.1 and wish to upgrade to a newer series, upgrade to MySQL 5.5 first before upgrading to 5.6, and so forth. For information on upgrading to MySQL 5.6, see the *MySQL 5.6 Reference Manual*.

To upgrade to MySQL 5.7, use the items in the following checklist as a guide:

- Before any upgrade, back up your databases, including the mysql database that contains the grant tables. See Section 7.2, “Database Backup Methods”.

- Read *all* the notes in Section 2.10.1.2, “Upgrading from MySQL 5.6 to 5.7”. These notes enable you to identify upgrade issues that apply to your current MySQL installation. Some incompatibilities discussed in that section require your attention *before* upgrading. Others require some action *after* upgrading.

- Read the Release Notes as well, which provide information about features that are new in MySQL 5.7 or differ from those found in earlier MySQL releases.

- After upgrading to a new version of MySQL, run mysql_upgrade (see Section 4.4.7, “mysql_upgrade — Check and Upgrade MySQL Tables”). This program checks your tables, and attempts to repair them if necessary. It also updates your grant tables to make sure that they have the current structure so that you can take advantage of any new capabilities. (Some releases of MySQL introduce changes to the structure of the grant tables to add new privileges or features.)

  mysql_upgrade does not upgrade the contents of the help tables. For upgrade instructions, see Section 5.1.10, “Server-Side Help”.

  mysql_upgrade should not be used when the server is running with --gtid-mode=ON, since it may make changes in nontransactional system tables in the mysql database, many of which are MyISAM and cannot be changed to use a different storage engine. See GTID mode and mysql_upgrade.

- If you run MySQL Server on Windows, see Section 2.3.7, “Upgrading MySQL on Windows”.

- If you use replication, see Section 16.4.3, "Upgrading a Replication Setup", for information on upgrading your replication setup.

- If you upgrade an installation originally produced by installing multiple RPM packages, it is best to upgrade all the packages, not just some. For example, if you previously installed the server and client RPMs, do not upgrade just the server RPM.

- If you have created a user-defined function (UDF) with a given name and upgrade MySQL to a version that implements a new built-in function with the same name, the UDF becomes inaccessible. To correct this, use `DROP FUNCTION` to drop the UDF, and then use `CREATE FUNCTION` to re-create the UDF with a different nonconflicting name. The same is true if the new version of MySQL implements a built-in function with the same name as an existing stored function. See Section 9.2.4, "Function Name Parsing and Resolution", for the rules describing how the server interprets references to different kinds of functions.

For EL5, EL6, or EL7-based Linux platforms and Fedora 19 or 20, you can perform an in-place upgrade of MySQL and its components with the MySQL Yum repository. See Section 2.10.1.1, "Upgrading MySQL with the MySQL Yum Repository".

For upgrades between versions of a MySQL release series that has reached General Availability status, you can move the MySQL format files and data files between different versions on systems with the same architecture. For upgrades to a version of a MySQL release series that is in development status, that is not necessarily true. Use of development releases is at your own risk.

If you are cautious about using new versions, you can always rename your old `mysqld` before installing a newer one. For example, if you are using a version of MySQL 5.6 and want to upgrade to 5.7, rename your current server from `mysqld` to `mysqld-5.6`. If your new `mysqld` then does something unexpected, you can simply shut it down and restart with your old `mysqld`.

If problems occur, such as that the new `mysqld` server does not start or that you cannot connect without a password, verify that you do not have an old `my.cnf` file from your previous installation. You can check this with the `--print-defaults` option (for example, `mysqld --print-defaults`). If this command displays anything other than the program name, you have an active `my.cnf` file that affects server or client operation.

If, after an upgrade, you experience problems with compiled client programs, such as `Commands out of sync` or unexpected core dumps, you probably have used old header or library files when compiling your programs. In this case, check the date for your `mysql.h` file and `libmysqlclient.a` library to verify that they are from the new MySQL distribution. If not, recompile your programs with the new headers and libraries. Recompilation might also be necessary for programs compiled against the shared client library if the library major version number has changed (for example from `libmysqlclient.so.15` to `libmysqlclient.so.16`.

If your MySQL installation contains a large amount of data that might take a long time to convert after an in-place upgrade, you might find it useful to create a "dummy" database instance for assessing what conversions might be needed and the work involved to perform them. Make a copy of your MySQL instance that contains a full copy of the `mysql` database, plus all other databases without data. Run your upgrade procedure on this dummy instance to see what actions might be needed so that you can better evaluate the work involved when performing actual data conversion on your original database instance.

It is a good idea to rebuild and reinstall the Perl `DBD::mysql` module whenever you install a new release of MySQL. The same applies to other MySQL interfaces as well, such as PHP `mysql` extensions and the Python `MySQLdb` module.

2.10.1.1 Upgrading MySQL with the MySQL Yum Repository

For EL5, EL6, or EL7-based Linux platforms and Fedora 19 or 20, you can upgrade MySQL and its components to the latest GA releases with the MySQL Yum repository.

Note

Before you perform any upgrade actions, please pay attention to the following:

- If your version of MySQL is more than one series older than the latest GA series (for example, assuming the current GA release series is 5.6 and you have 5.1.x installed right now), do NOT use the following instructions to update MySQL, and do NOT enable the MySQL Yum repository on your system until you have upgraded MySQL by other means (see Section 2.10.1, “Upgrading MySQL”) to at least the last GA series before the latest one.
- Before performing any update to MySQL, follow carefully the instructions in Section 2.10.1, “Upgrading MySQL”. Among other instructions discussed there, it is especially important to back up your database before the update.
- If your MySQL installation is a third-party distribution, follow the instructions in Section 2.5.2, “Replacing a Third-Party Distribution of MySQL Using the MySQL Yum Repository” for upgrading the installation.

The Yum update performs an in-place update for MySQL (that is, replaces the old version of the software and then runs the new version off the old version's data files). It updates MySQL to the latest release in the same release series. Assuming that you already have the MySQL Yum repository on your system's repository list (see Adding the MySQL Yum Repository for details), make sure your Yum repository setup is up-to-date by running:

```
shell> sudo yum update mysql-community-release
```

You can then update MySQL and its components by the following command:

```
shell> sudo yum update mysql-server
```

Alternatively, you can update the MySQL Yum repository setup and MySQL at the same time by telling Yum to update everything on your system (this might take considerably more time):

```
shell> sudo yum update
```

Note that by default, the `yum update` command will only update MySQL to the latest version in the same release series, which means, for example, a 5.6.x installation will NOT be updated to a 5.7.x release automatically. To update to the next release series, after updating the MySQL Yum repository setup as described above, you need to first disable the sub-repository for your original version and enable the sub-repository for your target version before you run the `yum update` command for MySQL. See the instructions for doing that in Enable and Disable the Appropriate Sub-Repositories [123].

Important

For important information about upgrading from MySQL 5.6 to 5.7, see Section 2.10.1.2, “Upgrading from MySQL 5.6 to 5.7”

The MySQL server always restarts after an update by Yum. Once the server restarts, you should run `mysql_upgrade` to check and possibly resolve any incompatibilities between the old data and the upgraded software. `mysql_upgrade` also performs other functions; see Section 4.4.7, “`mysql_upgrade` — Check and Upgrade MySQL Tables” for details.

Although we recommend that you update all the MySQL components at the same time, you can also update only a specific component. You can use the following command to list all the installed packages for the MySQL components, which can all be updated with the MySQL Yum repository:

```
shell> sudo yum list installed | grep "^mysql"
```

After identifying the package name of the component of your choice, update the package with the following command, replacing *package-name* with the name of the package:

```
shell> sudo yum update package-name
```

Upgrading the Shared Client Libraries

After updating MySQL using the Yum repository, applications compiled with older versions of the shared client libraries should continue to work.

If you recompile applications and dynamically link them with the updated libraries: As typical with new versions of shared libraries where there are differences or additions in symbol versioning between the newer and older libraries (for example, between the newer, standard 5.7 shared client libraries and some older—prior or variant—versions of the shared libraries shipped natively by the Linux distributions' software repositories, or from some other sources), any applications compiled using the updated, newer shared libraries will require those updated libraries on systems where the applications are deployed. And, as expected, if those libraries are not in place, the applications requiring the shared libraries will fail. So, be sure to deploy the packages for the shared libraries from MySQL on those systems. You can do this by adding the MySQL Yum repository to the systems (see Adding the MySQL Yum Repository) and install the latest shared libraries using the instructions given in Installing Additional MySQL Products and Components with Yum.

2.10.1.2 Upgrading from MySQL 5.6 to 5.7

Note

It is good practice to back up your data before installing any new version of software. Although MySQL works very hard to ensure a high level of quality, protect your data by making a backup.

To upgrade to 5.7 from any previous version, MySQL recommends that you dump your tables with `mysqldump` before upgrading and reload the dump file after upgrading. Use the `--all-databases` option to include all databases in the dump. If your databases include stored programs, use the `--routines` and `--events` options as well.

In general, do the following when upgrading from MySQL 5.6 to 5.7:

- Read *all* the items in these sections to see whether any of them might affect your applications:
  - Section 2.10.1, “Upgrading MySQL”, has general update information.
  - The items in the change lists provided later in this section enable you to identify upgrade issues that apply to your current MySQL installation. Some incompatibilities discussed there require your attention *before* upgrading. Others should be dealt with *after* upgrading.
  - The MySQL 5.7 Release Notes describe significant new features you can use in 5.7 or that differ from those found in earlier MySQL releases. Some of these changes may result in incompatibilities.

  Note particularly any changes that are marked **Known issue** or **Incompatible change**. These incompatibilities with earlier versions of MySQL may require your attention *before you upgrade*. Our aim is to avoid these changes, but occasionally they are necessary to correct problems that would be worse than an incompatibility between releases. If any upgrade issue applicable to your installation involves an incompatibility that requires special handling, follow the instructions given in the incompatibility description. Sometimes this involves dumping and reloading tables, or use of a statement such as CHECK TABLE or REPAIR TABLE.

For dump and reload instructions, see Section 2.10.4, “Rebuilding or Repairing Tables or Indexes”. Any procedure that involves REPAIR TABLE with the USE_FRM option *must* be done before upgrading. Use of this statement with a version of MySQL different from the one used to create the table (that is, using it after upgrading) may damage the table. See Section 13.7.2.5, “REPAIR TABLE Syntax”.

- Before upgrading to a new version of MySQL, Section 2.10.3, “Checking Whether Tables or Indexes Must Be Rebuilt”, to see whether changes to table formats or to character sets or collations were made between your current version of MySQL and the version to which you are upgrading. If so and these changes result in an incompatibility between MySQL versions, you will need to upgrade the affected tables using the instructions in Section 2.10.4, “Rebuilding or Repairing Tables or Indexes”.

- After upgrading to a new version of MySQL, run mysql_upgrade (see Section 4.4.7, “mysql_upgrade — Check and Upgrade MySQL Tables”). This program checks your tables, and attempts to repair them if necessary. It also updates your grant tables to make sure that they have the current structure so that you can take advantage of any new capabilities. (Some releases of MySQL introduce changes to the structure of the grant tables to add new privileges or features.)

  mysql_upgrade does not upgrade the contents of the help tables. For upgrade instructions, see Section 5.1.10, “Server-Side Help”.

- If you run MySQL Server on Windows, see Section 2.3.7, “Upgrading MySQL on Windows”.

- If you use replication, see Section 16.4.3, “Upgrading a Replication Setup”, for information on upgrading your replication setup.

- If you use InnoDB, consider setting innodb_fast_shutdown to 0 before shutting down and upgrading your server. When you set innodb_fast_shutdown to 0, InnoDB does a slow shutdown, a full purge and an insert buffer merge before shutting down, which ensures that all data files are fully prepared in case the upgrade process modifies the file format.

If your MySQL installation contains a large amount of data that might take a long time to convert after an in-place upgrade, you might find it useful to create a “dummy” database instance for assessing what conversions might be needed and the work involved to perform them. Make a copy of your MySQL instance that contains a full copy of the mysql database, plus all other databases without data. Run your upgrade procedure on this dummy instance to see what actions might be needed so that you can better evaluate the work involved when performing actual data conversion on your original database instance.

Read *all* the items in the following sections to see whether any of them might affect your applications:

Server Changes

- **Incompatible change**: As of MySQL 5.7.4, the deprecated ERROR_FOR_DIVISION_BY_ZERO, NO_ZERO_DATE, and NO_ZERO_IN_DATE SQL modes do nothing. Instead, their previous effects are included in the effects of strict SQL mode (STRICT_ALL_TABLES or STRICT_TRANS_TABLES). In other words, strict mode now means the same thing as the previous meaning of strict mode plus the ERROR_FOR_DIVISION_BY_ZERO, NO_ZERO_DATE, and NO_ZERO_IN_DATE modes. This change reduces the number of SQL modes with an effect dependent on strict mode and makes them part of strict mode itself.

  To prepare for these SQL mode changes, it is advisable *before* upgrading to read SQL Mode Changes in MySQL 5.7. That discussion provides guidelines to assess whether your applications will be affected by these changes.

The deprecated ERROR_FOR_DIVISION_BY_ZERO, NO_ZERO_DATE, and NO_ZERO_IN_DATE SQL modes are still recognized so that statements that name them do not produce an error, but will be removed in a future version of MySQL. To make advance preparation for versions of MySQL in which these modes do not exist, applications should be modified to not refer to those mode names.

- **Incompatible change**: As of MySQL 5.7.2, the server requires account rows in the mysql.user table to have a nonempty plugin column value and disables accounts with an empty value. This requires that you upgrade as follows.

  For an upgrade in which you plan to use the data directory from your existing MySQL installation:

  1. Stop the server
  2. Upgrade MySQL in place
  3. Restart the server with the --skip-grant-tables option to disable privilege checking
  4. Run mysql_upgrade
  5. Restart the server normally (without --skip-grant-tables)

  For an upgrade in which you plan to reload a dump file generated from your existing MySQL installation:

  1. To generate the dump file, run mysqldump without the --flush-privileges option
  2. Stop the server
  3. Upgrade MySQL in place
  4. Restart the server with the --skip-grant-tables option to disable privilege checking
  5. Reload the dump file (mysql < *dump_file*)
  6. Execute mysql_upgrade
  7. Restart the server normally (without --skip-grant-tables)

  mysql_upgrade runs by default as the MySQL root user. For either of the preceding procedures, if the root password is expired when you run mysql_upgrade, you will see a message that your password is expired and that mysql_upgrade failed as a result. To correct this, reset the root password to unexpire it and run mysql_upgrade again:

  ```
  shell> mysql -u root -p
  Enter password: ****  <- enter root password here
  mysql> SET PASSWORD = PASSWORD('root-password');
  mysql> quit

  shell> mysql_upgrade
  ```

  SET PASSWORD normally does not work if the server is started with --skip-grant-tables, but the first invocation of mysql_upgrade flushes the privileges, so when you run mysql, the SET PASSWORD statement is accepted.

  After following the preceding instructions, DBAs are advised to also convert accounts that use the deprecated mysql_old_password authentication plugin to use mysql_native_password instead.

For account upgrade instructions, see Section 6.3.9.3, “Migrating Away from Pre-4.1 Password Hashing and the mysql_old_password Plugin”.

- **Incompatible change**: It is possible for a column DEFAULT value to be valid for the sql_mode value at table-creation time but invalid for the sql_mode value when rows are inserted or updated. Example:

  ```
  SET sql_mode = '';
  CREATE TABLE t (d DATE DEFAULT 0);
  SET sql_mode = 'NO_ZERO_DATE,STRICT_ALL_TABLES';
  INSERT INTO t (d) VALUES(DEFAULT);
  ```

  In this case, 0 should be accepted for the CREATE TABLE but rejected for the INSERT. However, the server did not evaluate DEFAULT values used for inserts or updates against the current sql_mode. In the example, the INSERT succeeds and inserts '0000-00-00' into the DATE column.

  As of MySQL 5.7.2, the server applies the proper sql_mode checks to generate a warning or error at insert or update time.

  A resulting incompatibility for replication if you use statement-based logging (binlog_format=STATEMENT) is that if a slave is upgraded, a nonupgraded master will execute the preceding example without error, whereas the INSERT will fail on the slave and replication will stop.

  To deal with this, stop all new statements on the master and wait until the slaves catch up. Then upgrade the slaves followed by the master. Alternatively, if you cannot stop new statements, temporarily change to row-based logging on the master (binlog_format=ROW) and wait until all slaves have processed all binary logs produced up to the point of this change. Then upgrade the slaves followed by the master and change the master back to statement-based logging.

- **Incompatible change**: Several changes were made to the audit log plugin for better compatibility with Oracle Audit Vault. For upgrading purpose, the main issue is that the format of the audit log file has changed: Information within <AUDIT_RECORD> elements previously written using attributes now is written using subelements.

  Example of old <AUDIT_RECORD> format:

  ```
  <AUDIT_RECORD
   TIMESTAMP="2013-04-15T15:27:27"
   NAME="Query"
   CONNECTION_ID="3"
   STATUS="0"
   SQLTEXT="SELECT 1"
  />
  ```

  Example of new format:

  ```
  <AUDIT_RECORD>
   <TIMESTAMP>2013-04-15T15:27:27 UTC</TIMESTAMP>
   <RECORD_ID>3998_2013-04-15T15:27:27</RECORD_ID>
   <NAME>Query</NAME>
   <CONNECTION_ID>3</CONNECTION_ID>
   <STATUS>0</STATUS>
   <STATUS_CODE>0</STATUS_CODE>
   <USER>root[root] @ localhost [127.0.0.1]</USER>
   <OS_LOGIN></OS_LOGIN>
   <HOST>localhost</HOST>
   <IP>127.0.0.1</IP>
   <COMMAND_CLASS>select</COMMAND_CLASS>
   <SQLTEXT>SELECT 1</SQLTEXT>
  ```

```
</AUDIT_RECORD>
```

If you previously used an older version of the audit log plugin, use this procedure to avoid writing new-format log entries to an existing log file that contains old-format entries:

1. Stop the server.
2. Rename the current audit log file manually. This file will contain only old-format log entries.
3. Update the server and restart it. The audit log plugin will create a new log file, which will contain only new-format log entries.

For information about the audit log plugin, see Section 6.3.13, “MySQL Enterprise Audit Log Plugin”.

SQL Changes

- A trigger can have triggers for different combinations of trigger event (`INSERT`, `UPDATE`, `DELETE`) and action time (`BEFORE`, `AFTER`), but before MySQL 5.7.2 cannot have multiple triggers that have the same trigger event and action time. MySQL 5.7.2 lifts this limitation and multiple triggers are permitted. This change has implications for upgrades.

  Suppose that you upgrade an old server that does not support multiple triggers to MySQL 5.7.2 or newer. If the new server is a replication master and has old slaves that do not support multiple triggers, an error occurs on those slaves if a trigger is created on the master for a table that already has a trigger with the same trigger event and action time. To avoid this problem, upgrade the slaves first, then upgrade the master.

- Some keywords may be reserved in MySQL 5.7 that were not reserved in MySQL 5.6. See Section 9.3, “Reserved Words”.

2.10.2 Downgrading MySQL

This section describes what to do to downgrade to an older MySQL version, in the unlikely case that the previous version worked better than the new one.

It is always a good idea to make a backup beforehand, in case a downgrade fails and leaves the instance in an unusable state.

To downgrade between General Availability (GA) status versions within the same release series, typically you just install the new binaries on top of the old ones and do not make any changes to the databases.

Downgrades between milestone releases (or from a GA release to a milestone release) within the same release series are not supported and you may encounter issues.

The following items form a checklist of things to do whenever you perform a downgrade:

- Read the upgrading section for the release series from which you are downgrading to be sure that it does not have any features you really need. See Section 2.10.1, “Upgrading MySQL”.
- If there is a downgrading section for that version, read that as well.
- To see which new features were added between the version to which you are downgrading and your current version, see the Release Notes.
- Check Section 2.10.3, “Checking Whether Tables or Indexes Must Be Rebuilt”, to see whether changes to table formats or to character sets or collations were made between your current version of MySQL

and the version to which you are downgrading. If so and these changes result in an incompatibility between MySQL versions, you will need to downgrade the affected tables using the instructions in Section 2.10.4, “Rebuilding or Repairing Tables or Indexes”.

In most cases, you can move the MySQL format files and data files between different GA versions on the same architecture as long as you stay within versions for the same release series of MySQL.

If you downgrade from one release series to another, there may be incompatibilities in table storage formats. In this case, use `mysqldump` to dump your tables before downgrading. After downgrading, reload the dump file using `mysql` or `mysqlimport` to re-create your tables. For examples, see Section 2.10.5, “Copying MySQL Databases to Another Machine”.

A typical symptom of a downward-incompatible table format change when you downgrade is that you cannot open tables. In that case, use the following procedure:

1. Stop the older MySQL server that you are downgrading to.
2. Restart the newer MySQL server you are downgrading from.
3. Dump any tables that were inaccessible to the older server by using `mysqldump` to create a dump file.
4. Stop the newer MySQL server and restart the older one.
5. Reload the dump file into the older server. Your tables should be accessible.

If system tables in the `mysql` database changed, downgrading might introduce some loss of functionality or require some adjustments. Here are some examples:

- Trigger creation requires the `TRIGGER` privilege as of MySQL 5.1. In MySQL 5.0, there is no `TRIGGER` privilege and `SUPER` is required instead. If you downgrade from MySQL 5.1 to 5.0, you will need to give the `SUPER` privilege to those accounts that had the `TRIGGER` privilege in 5.1.
- Triggers were added in MySQL 5.0, so if you downgrade from 5.0 to 4.1, you cannot use triggers at all.
- The `mysql.proc.comment` column definition changed between MySQL 5.1 and 5.5. After a downgrade from 5.5 to 5.1, this table is seen as corrupt and in need of repair. To workaround this problem, execute `mysql_upgrade` from the version of MySQL to which you downgraded.

2.10.2.1 Downgrading to MySQL 5.6

When downgrading to MySQL 5.6 from MySQL 5.7, keep in mind the following issues relating to features found in MySQL 5.7, but not in MySQL 5.6:

SQL Changes

- A trigger can have triggers for different combinations of trigger event (`INSERT`, `UPDATE`, `DELETE`) and action time (`BEFORE`, `AFTER`), but before MySQL 5.7.2 cannot have multiple triggers that have the same trigger event and action time. MySQL 5.7.2 lifts this limitation and multiple triggers are permitted. This change has implications for downgrades.

  If you downgrade a server that supports multiple triggers to an older version that does not, the downgrade has these effects:

  - For each table that has triggers, all trigger definitions remain in the `.TRG` file for the table. However, if there are multiple triggers with the same trigger event and action time, the server executes only one of them when the trigger event occurs. For information about `.TRG` files, see Table Trigger Storage.

- If triggers for the table are added or dropped subsequent to the downgrade, the server rewrites the table's .TRG file. The rewritten file retains only one trigger per combination of trigger event and action time; the others are lost.

To avoid these problems, modify your triggers before downgrading. For each table that has multiple triggers per combination of trigger event and action time, convert each such set of triggers to a single trigger as follows:

1. For each trigger, create a stored routine that contains all the code in the trigger. Values accessed using NEW and OLD can be passed to the routine using parameters. If the trigger needs a single result value from the code, you can put the code in a stored function and have the function return the value. If the trigger needs multiple result values from the code, you can put the code in a stored procedure and return the values using OUT parameters.

2. Drop all triggers for the table.

3. Create one new trigger for the table that invokes the stored routines just created. The effect for this trigger is thus the same as the multiple triggers it replaces.

2.10.3 Checking Whether Tables or Indexes Must Be Rebuilt

A binary upgrade or downgrade is one that installs one version of MySQL “in place” over an existing version, without dumping and reloading tables:

1. Stop the server for the existing version if it is running.

2. Install a different version of MySQL. This is an upgrade if the new version is higher than the original version, a downgrade if the version is lower.

3. Start the server for the new version.

In many cases, the tables from the previous version of MySQL can be used without problem by the new version. However, sometimes changes occur that require tables or table indexes to be rebuilt, as described in this section. If you have tables that are affected by any of the issues described here, rebuild the tables or indexes as necessary using the instructions given in Section 2.10.4, “Rebuilding or Repairing Tables or Indexes”.

Table Incompatibilities

After a binary upgrade to MySQL 5.1 from a MySQL 5.0 installation that contains ARCHIVE tables, accessing those tables causes the server to crash, even if you have run mysql_upgrade or CHECK TABLE ... FOR UPGRADE. To work around this problem, use mysqldump to dump all ARCHIVE tables before upgrading, and reload them into MySQL 5.1 after upgrading. The same problem occurs for binary downgrades from MySQL 5.1 to 5.0.

The upgrade problem is fixed in MySQL 5.6.4: The server can open ARCHIVE tables created in MySQL 5.0. However, it remains the recommended upgrade procedure to dump 5.0 ARCHIVE tables before upgrading and reload them after upgrading.

Index Incompatibilities

In MySQL 5.6.3, the length limit for index prefix keys is increased from 767 bytes to 3072 bytes, for InnoDB tables using ROW_FORMAT=DYNAMIC or ROW_FORMAT=COMPRESSED. See Section 14.2.6.7, “Limits on InnoDB Tables” for details. This change is also backported to MySQL 5.5.14. If you downgrade from one of these releases or higher, to an earlier release with a lower length limit, the index prefix keys could be truncated at 767 bytes or the downgrade could fail. This issue could only occur if the configuration option innodb_large_prefix was enabled on the server being downgraded.

If you perform a binary upgrade without dumping and reloading tables, you cannot upgrade directly from MySQL 4.1 to 5.1 or higher. This occurs due to an incompatible change in the MyISAM table index format in MySQL 5.0. Upgrade from MySQL 4.1 to 5.0 and repair all MyISAM tables. Then upgrade from MySQL 5.0 to 5.1 and check and repair your tables.

Modifications to the handling of character sets or collations might change the character sort order, which causes the ordering of entries in any index that uses an affected character set or collation to be incorrect. Such changes result in several possible problems:

- Comparison results that differ from previous results
- Inability to find some index values due to misordered index entries
- Misordered ORDER BY results
- Tables that CHECK TABLE reports as being in need of repair

The solution to these problems is to rebuild any indexes that use an affected character set or collation, either by dropping and re-creating the indexes, or by dumping and reloading the entire table. In some cases, it is possible to alter affected columns to use a different collation. For information about rebuilding indexes, see Section 2.10.4, “Rebuilding or Repairing Tables or Indexes”.

To check whether a table has indexes that must be rebuilt, consult the following list. It indicates which versions of MySQL introduced character set or collation changes that require indexes to be rebuilt. Each entry indicates the version in which the change occurred and the character sets or collations that the change affects. If the change is associated with a particular bug report, the bug number is given.

The list applies both for binary upgrades and downgrades. For example, Bug #27877 was fixed in MySQL 5.1.24, so it applies to upgrades from versions older than 5.1.24 to 5.1.24 or newer, and to downgrades from 5.1.24 or newer to versions older than 5.1.24.

In many cases, you can use CHECK TABLE ... FOR UPGRADE to identify tables for which index rebuilding is required. It will report this message:

```
Table upgrade required.
Please do "REPAIR TABLE `tbl_name`" or dump/reload to fix it!
```

In these cases, you can also use mysqlcheck --check-upgrade or mysql_upgrade, which execute CHECK TABLE. However, the use of CHECK TABLE applies only after upgrades, not downgrades. Also, CHECK TABLE is not applicable to all storage engines. For details about which storage engines CHECK TABLE supports, see Section 13.7.2.2, “CHECK TABLE Syntax”.

These changes cause index rebuilding to be necessary:

- MySQL 5.1.24 (Bug #27877)

  Affects indexes that use the utf8_general_ci or ucs2_general_ci collation for columns that contain 'ß' LATIN SMALL LETTER SHARP S (German). The bug fix corrected an error in the original collations but introduced an incompatibility such that 'ß' compares equal to characters with which it previously compared different.

  Affected tables can be detected by CHECK TABLE ... FOR UPGRADE as of MySQL 5.1.30 (see Bug #40053).

A workaround for this issue is implemented as of MySQL 5.1.62, 5.5.21, and 5.6.5. The workaround involves altering affected columns to use the `utf8_general_mysql500_ci` and `ucs2_general_mysql500_ci` collations, which preserve the original pre-5.1.24 ordering of `utf8_general_ci` and `ucs2_general_ci`.

- MySQL 5.0.48, 5.1.23 (Bug #27562)

  Affects indexes that use the `ascii_general_ci` collation for columns that contain any of these characters: `` '`' `` GRAVE ACCENT, `'['` LEFT SQUARE BRACKET, `'\'` REVERSE SOLIDUS, `']'` RIGHT SQUARE BRACKET, `'~'` TILDE

  Affected tables can be detected by `CHECK TABLE ... FOR UPGRADE` as of MySQL 5.1.29 (see Bug #39585).

- MySQL 5.0.48, 5.1.21 (Bug #29461)

  Affects indexes for columns that use any of these character sets: `eucjpms`, `euc_kr`, `gb2312`, `latin7`, `macce`, `ujis`

  Affected tables can be detected by `CHECK TABLE ... FOR UPGRADE` as of MySQL 5.1.29 (see Bug #39585).

2.10.4 Rebuilding or Repairing Tables or Indexes

This section describes how to rebuild a table, following changes to MySQL such as how data types or character sets are handled. For example, an error in a collation might have been corrected, requiring a table rebuild to update the indexes for character columns that use the collation. (For examples, see Section 2.10.3, “Checking Whether Tables or Indexes Must Be Rebuilt”.) You might also need to repair or upgrade a table, as indicated by a table check operation such as that performed by `CHECK TABLE`, `mysqlcheck`, or `mysql_upgrade`.

Methods for rebuilding a table include dumping and reloading it, or using `ALTER TABLE` or `REPAIR TABLE`.

Note

If you are rebuilding tables because a different version of MySQL will not handle them after a binary (in-place) upgrade or downgrade, you must use the dump-and-reload method. Dump the tables *before* upgrading or downgrading using your original version of MySQL. Then reload the tables *after* upgrading or downgrading.

If you use the dump-and-reload method of rebuilding tables only for the purpose of rebuilding indexes, you can perform the dump either before or after upgrading or downgrading. Reloading still must be done afterward.

To rebuild a table by dumping and reloading it, use `mysqldump` to create a dump file and `mysql` to reload the file:

```
shell> mysqldump db_name t1 > dump.sql
shell> mysql db_name < dump.sql
```

To rebuild all the tables in a single database, specify the database name without any following table name:

```
shell> mysqldump db_name > dump.sql
shell> mysql db_name < dump.sql
```

To rebuild all tables in all databases, use the `--all-databases` option:

```
shell> mysqldump --all-databases > dump.sql
shell> mysql < dump.sql
```

To rebuild a table with `ALTER TABLE`, use a “null” alteration; that is, an `ALTER TABLE` statement that “changes” the table to use the storage engine that it already has. For example, if `t1` is a `MyISAM` table, use this statement:

```
mysql> ALTER TABLE t1 ENGINE = MyISAM;
```

If you are not sure which storage engine to specify in the `ALTER TABLE` statement, use `SHOW CREATE TABLE` to display the table definition.

If you must rebuild a table because a table checking operation indicates that the table is corrupt or needs an upgrade, you can use `REPAIR TABLE` if that statement supports the table's storage engine. For example, to repair a `MyISAM` table, use this statement:

```
mysql> REPAIR TABLE t1;
```

For storage engines such as `InnoDB` that `REPAIR TABLE` does not support, use `mysqldump` to create a dump file and `mysql` to reload the file, as described earlier.

For specifics about which storage engines `REPAIR TABLE` supports, see Section 13.7.2.5, “`REPAIR TABLE` Syntax”.

`mysqlcheck --repair` provides command-line access to the `REPAIR TABLE` statement. This can be a more convenient means of repairing tables because you can use the `--databases` or `--all-databases` option to repair all tables in specific databases or all databases, respectively:

```
shell> mysqlcheck --repair --databases db_name ...
shell> mysqlcheck --repair --all-databases
```

For incompatibilities introduced in MySQL 5.1.24 by the fix for Bug #27877 that corrected the `utf8_general_ci` and `ucs2_general_ci` collations, a workaround is implemented as of MySQL 5.1.62, 5.5.21, and 5.6.5. Upgrade to one of those versions, then convert each affected table using one of the following methods. In each case, the workaround altering affected columns to use the `utf8_general_mysql500_ci` and `ucs2_general_mysql500_ci` collations, which preserve the original pre-5.1.24 ordering of `utf8_general_ci` and `ucs2_general_ci`.

- To convert an affected table after a binary upgrade that leaves the table files in place, alter the table to use the new collation. Suppose that the table `t1` contains one or more problematic `utf8` columns. To convert the table at the table level, use a statement like this:

  ```
  ALTER TABLE t1
  CONVERT TO CHARACTER SET utf8 COLLATE utf8_general_mysql500_ci;
  ```

  To apply the change on a column-specific basis, use a statement like this (be sure to repeat the column definition as originally specified except for the `COLLATE` clause):

  ```
  ALTER TABLE t1
  MODIFY c1 CHAR(N) CHARACTER SET utf8 COLLATE utf8_general_mysql500_ci;
  ```

- To upgrade the table using a dump and reload procedure, dump the table using mysqldump, modify the CREATE TABLE statement in the dump file to use the new collation, and reload the table.

After making the appropriate changes, CHECK TABLE should report no error.

2.10.5 Copying MySQL Databases to Another Machine

You can copy the .frm, .MYI, and .MYD files for MyISAM tables between different architectures that support the same floating-point format. (MySQL takes care of any byte-swapping issues.) See Section 14.3, “The MyISAM Storage Engine”.

In cases where you need to transfer databases between different architectures, you can use mysqldump to create a file containing SQL statements. You can then transfer the file to the other machine and feed it as input to the mysql client.

Use mysqldump --help to see what options are available.

The easiest (although not the fastest) way to move a database between two machines is to run the following commands on the machine on which the database is located:

```
shell> mysqladmin -h 'other_hostname' create db_name
shell> mysqldump db_name | mysql -h 'other_hostname' db_name
```

If you want to copy a database from a remote machine over a slow network, you can use these commands:

```
shell> mysqladmin create db_name
shell> mysqldump -h 'other_hostname' --compress db_name | mysql db_name
```

You can also store the dump in a file, transfer the file to the target machine, and then load the file into the database there. For example, you can dump a database to a compressed file on the source machine like this:

```
shell> mysqldump --quick db_name | gzip > db_name.gz
```

Transfer the file containing the database contents to the target machine and run these commands there:

```
shell> mysqladmin create db_name
shell> gunzip < db_name.gz | mysql db_name
```

You can also use mysqldump and mysqlimport to transfer the database. For large tables, this is much faster than simply using mysqldump. In the following commands, *DUMPDIR* represents the full path name of the directory you use to store the output from mysqldump.

First, create the directory for the output files and dump the database:

```
shell> mkdir DUMPDIR
shell> mysqldump --tab=DUMPDIR db_name
```

Then transfer the files in the *DUMPDIR* directory to some corresponding directory on the target machine and load the files into MySQL there:

```
shell> mysqladmin create db_name           # create database
shell> cat DUMPDIR/*.sql | mysql db_name   # create tables in database
shell> mysqlimport db_name DUMPDIR/*.txt   # load data into tables
```

Do not forget to copy the mysql database because that is where the grant tables are stored. You might have to run commands as the MySQL root user on the new machine until you have the mysql database in place.

After you import the mysql database on the new machine, execute mysqladmin flush-privileges so that the server reloads the grant table information.

2.11 Environment Variables

This section lists all the environment variables that are used directly or indirectly by MySQL. Most of these can also be found in other places in this manual.

Note that any options on the command line take precedence over values specified in option files and environment variables, and values in option files take precedence over values in environment variables.

In many cases, it is preferable to use an option file instead of environment variables to modify the behavior of MySQL. See Section 4.2.3.3, “Using Option Files”.

Variable	Description
CXX	The name of your C++ compiler (for running CMake).
CC	The name of your C compiler (for running CMake).
DBI_USER	The default user name for Perl DBI.
DBI_TRACE	Trace options for Perl DBI.
HOME	The default path for the mysql history file is $HOME/.mysql_history.
LD_RUN_PATH	Used to specify the location of libmysqlclient.so.
LIBMYSQL_ENABLE_CLEARTEXT_PLUGIN	Enable mysql_clear_password authentication plugin; see Section 6.3.9.5, “The Cleartext Client-Side Authentication Plugin”.
LIBMYSQL_PLUGIN_DIR	Directory in which to look for client plugins.
LIBMYSQL_PLUGINS	Client plugins to preload.
MYSQL_DEBUG	Debug trace options when debugging.
MYSQL_GROUP_SUFFIX	Option group suffix value (like specifying --defaults-group-suffix).
MYSQL_HISTFILE	The path to the mysql history file. If this variable is set, its value overrides the default for $HOME/.mysql_history.
MYSQL_HISTIGNORE	Patterns specifying statements that mysql should not log to $HOME/.mysql_history, or syslog if --syslog is given.
MYSQL_HOME	The path to the directory in which the server-specific my.cnf file resides.
MYSQL_HOST	The default host name used by the mysql command-line client.
MYSQL_PS1	The command prompt to use in the mysql command-line client.
MYSQL_PWD	The default password when connecting to mysqld. Note that using this is insecure. See Section 6.1.2.1, “End-User Guidelines for Password Security”.
MYSQL_TCP_PORT	The default TCP/IP port number.
MYSQL_TEST_LOGIN_FILE	The name of the .mylogin.cnf login file.
MYSQL_TEST_TRACE_CRASH	Whether the test protocol trace plugin crashes clients. See note following table
MYSQL_TEST_TRACE_DEBUG	Whether the test protocol trace plugin produces output. See note following table

Variable	Description
MYSQL_UNIX_PORT	The default Unix socket file name; used for connections to localhost.
PATH	Used by the shell to find MySQL programs.
TMPDIR	The directory where temporary files are created.
TZ	This should be set to your local time zone. See Section C.5.4.6, "Time Zone Problems".
UMASK	The user-file creation mode when creating files. See note following table.
UMASK_DIR	The user-directory creation mode when creating directories. See note following table.
USER	The default user name on Windows when connecting to mysqld.

For information about the mysql history file, see Section 4.5.1.3, "mysql Logging".

MYSQL_TEST_LOGIN_FILE is the path name of the login file (the file created by mysql_config_editor). If not set, the default value is %APPDATA%\MySQL\.mylogin.cnf directory on Windows and $HOME/.mylogin.cnf on non-Windows systems. See Section 4.6.6, "mysql_config_editor — MySQL Configuration Utility".

The MYSQL_TEST_TRACE_DEBUG and MYSQL_TRACE_TRACE_CRASH variables control the test protocol trace client plugin, if MySQL is built with that plugin enabled. For more information, see Using the Test Protocol Trace Plugin.

The UMASK and UMASK_DIR variables, despite their names, are used as modes, not masks:

- If UMASK is set, mysqld uses ($UMASK | 0600) as the mode for file creation, so that newly created files have a mode in the range from 0600 to 0666 (all values octal).

- If UMASK_DIR is set, mysqld uses ($UMASK_DIR | 0700) as the base mode for directory creation, which then is AND-ed with ~(~$UMASK & 0666), so that newly created directories have a mode in the range from 0700 to 0777 (all values octal). The AND operation may remove read and write permissions from the directory mode, but not execute permissions.

MySQL assumes that the value for UMASK or UMASK_DIR is in octal if it starts with a zero.

2.12 Perl Installation Notes

The Perl DBI module provides a generic interface for database access. You can write a DBI script that works with many different database engines without change. To use DBI, you must install the DBI module, as well as a DataBase Driver (DBD) module for each type of database server you want to access. For MySQL, this driver is the DBD::mysql module.

Perl, and the DBD::MySQL module for DBI must be installed if you want to run the MySQL benchmark scripts; see Section 8.12.2, "The MySQL Benchmark Suite".

> **Note**
>
> Perl support is not included with MySQL distributions. You can obtain the necessary modules from http://search.cpan.org for Unix, or by using the ActiveState ppm program on Windows. The following sections describe how to do this.

The DBI/DBD interface requires Perl 5.6.0, and 5.6.1 or later is preferred. DBI *does not work* if you have an older version of Perl. You should use DBD::mysql 4.009 or higher. Although earlier versions are available, they do not support the full functionality of MySQL 5.7.

2.12.1 Installing Perl on Unix

MySQL Perl support requires that you have installed MySQL client programming support (libraries and header files). Most installation methods install the necessary files. If you install MySQL from RPM files on Linux, be sure to install the developer RPM as well. The client programs are in the client RPM, but client programming support is in the developer RPM.

The files you need for Perl support can be obtained from the CPAN (Comprehensive Perl Archive Network) at http://search.cpan.org.

The easiest way to install Perl modules on Unix is to use the `CPAN` module. For example:

```
shell> perl -MCPAN -e shell
cpan> install DBI
cpan> install DBD::mysql
```

The `DBD::mysql` installation runs a number of tests. These tests attempt to connect to the local MySQL server using the default user name and password. (The default user name is your login name on Unix, and `ODBC` on Windows. The default password is “no password.”) If you cannot connect to the server with those values (for example, if your account has a password), the tests fail. You can use `force install DBD::mysql` to ignore the failed tests.

`DBI` requires the `Data::Dumper` module. It may be installed; if not, you should install it before installing `DBI`.

It is also possible to download the module distributions in the form of compressed `tar` archives and build the modules manually. For example, to unpack and build a DBI distribution, use a procedure such as this:

1. Unpack the distribution into the current directory:

   ```
   shell> gunzip < DBI-VERSION.tar.gz | tar xvf -
   ```

   This command creates a directory named `DBI-VERSION`.

2. Change location into the top-level directory of the unpacked distribution:

   ```
   shell> cd DBI-VERSION
   ```

3. Build the distribution and compile everything:

   ```
   shell> perl Makefile.PL
   shell> make
   shell> make test
   shell> make install
   ```

The `make test` command is important because it verifies that the module is working. Note that when you run that command during the `DBD::mysql` installation to exercise the interface code, the MySQL server must be running or the test fails.

It is a good idea to rebuild and reinstall the `DBD::mysql` distribution whenever you install a new release of MySQL. This ensures that the latest versions of the MySQL client libraries are installed correctly.

If you do not have access rights to install Perl modules in the system directory or if you want to install local Perl modules, the following reference may be useful: http://servers.digitaldaze.com/extensions/perl/modules.html#modules

Look under the heading “Installing New Modules that Require Locally Installed Modules.”

2.12.2 Installing ActiveState Perl on Windows

On Windows, you should do the following to install the MySQL DBD module with ActiveState Perl:

1. Get ActiveState Perl from http://www.activestate.com/Products/ActivePerl/ and install it.
2. Open a console window.
3. If necessary, set the HTTP_proxy variable. For example, you might try a setting like this:

   ```
   C:\> set HTTP_proxy=my.proxy.com:3128
   ```

4. Start the PPM program:

   ```
   C:\> C:\perl\bin\ppm.pl
   ```

5. If you have not previously done so, install DBI:

   ```
   ppm> install DBI
   ```

6. If this succeeds, run the following command:

   ```
   ppm> install DBD-mysql
   ```

This procedure should work with ActiveState Perl 5.6 or newer.

If you cannot get the procedure to work, you should install the ODBC driver instead and connect to the MySQL server through ODBC:

```
use DBI;
$dbh= DBI->connect("DBI:ODBC:$dsn",$user,$password) ||
  die "Got error $DBI::errstr when connecting to $dsn\n";
```

2.12.3 Problems Using the Perl DBI/DBD Interface

If Perl reports that it cannot find the ../mysql/mysql.so module, the problem is probably that Perl cannot locate the libmysqlclient.so shared library. You should be able to fix this problem by one of the following methods:

- Copy libmysqlclient.so to the directory where your other shared libraries are located (probably /usr/lib or /lib).
- Modify the -L options used to compile DBD::mysql to reflect the actual location of libmysqlclient.so.
- On Linux, you can add the path name of the directory where libmysqlclient.so is located to the /etc/ld.so.conf file.
- Add the path name of the directory where libmysqlclient.so is located to the LD_RUN_PATH environment variable. Some systems use LD_LIBRARY_PATH instead.

Note that you may also need to modify the -L options if there are other libraries that the linker fails to find. For example, if the linker cannot find libc because it is in /lib and the link command specifies -L/usr/lib, change the -L option to -L/lib or add -L/lib to the existing link command.

If you get the following errors from DBD::mysql, you are probably using gcc (or using an old binary compiled with gcc):

```
/usr/bin/perl: can't resolve symbol '__moddi3'
/usr/bin/perl: can't resolve symbol '__divdi3'
```

Add -L/usr/lib/gcc-lib/... -lgcc to the link command when the mysql.so library gets built (check the output from make for mysql.so when you compile the Perl client). The -L option should specify the path name of the directory where libgcc.a is located on your system.

Another cause of this problem may be that Perl and MySQL are not both compiled with gcc. In this case, you can solve the mismatch by compiling both with gcc.

Chapter 3 Tutorial

Table of Contents

This chapter provides a tutorial introduction to MySQL by showing how to use the mysql client program to create and use a simple database. mysql (sometimes referred to as the “terminal monitor” or just “monitor”) is an interactive program that enables you to connect to a MySQL server, run queries, and view the results. mysql may also be used in batch mode: you place your queries in a file beforehand, then tell mysql to execute the contents of the file. Both ways of using mysql are covered here.

To see a list of options provided by mysql, invoke it with the --help option:

```
shell> mysql --help
```

This chapter assumes that mysql is installed on your machine and that a MySQL server is available to which you can connect. If this is not true, contact your MySQL administrator. (If *you* are the administrator, you need to consult the relevant portions of this manual, such as Chapter 5, *MySQL Server Administration*.)

This chapter describes the entire process of setting up and using a database. If you are interested only in accessing an existing database, you may want to skip over the sections that describe how to create the database and the tables it contains.

Because this chapter is tutorial in nature, many details are necessarily omitted. Consult the relevant sections of the manual for more information on the topics covered here.

3.1 Connecting to and Disconnecting from the Server

To connect to the server, you will usually need to provide a MySQL user name when you invoke mysql and, most likely, a password. If the server runs on a machine other than the one where you log in, you will also need to specify a host name. Contact your administrator to find out what connection parameters you should use to connect (that is, what host, user name, and password to use). Once you know the proper parameters, you should be able to connect like this:

```
shell> mysql -h host -u user -p
Enter password: ********
```

host and user represent the host name where your MySQL server is running and the user name of your MySQL account. Substitute appropriate values for your setup. The ******** represents your password; enter it when mysql displays the Enter password: prompt.

If that works, you should see some introductory information followed by a mysql> prompt:

```
shell> mysql -h host -u user -p
Enter password: ********
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 25338 to server version: 5.7.5-standard

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql>
```

The mysql> prompt tells you that mysql is ready for you to enter commands.

If you are logging in on the same machine that MySQL is running on, you can omit the host, and simply use the following:

```
shell> mysql -u user -p
```

If, when you attempt to log in, you get an error message such as ERROR 2002 (HY000): Can't connect to local MySQL server through socket '/tmp/mysql.sock' (2), it means that the MySQL server daemon (Unix) or service (Windows) is not running. Consult the administrator or see the section of Chapter 2, *Installing and Upgrading MySQL* that is appropriate to your operating system.

For help with other problems often encountered when trying to log in, see Section C.5.2, “Common Errors When Using MySQL Programs”.

Some MySQL installations permit users to connect as the anonymous (unnamed) user to the server running on the local host. If this is the case on your machine, you should be able to connect to that server by invoking mysql without any options:

```
shell> mysql
```

After you have connected successfully, you can disconnect any time by typing QUIT (or \q) at the mysql> prompt:

```
mysql> QUIT
Bye
```

On Unix, you can also disconnect by pressing Control+D.

Most examples in the following sections assume that you are connected to the server. They indicate this by the mysql> prompt.

3.2 Entering Queries

Make sure that you are connected to the server, as discussed in the previous section. Doing so does not in itself select any database to work with, but that is okay. At this point, it is more important to find out a little about how to issue queries than to jump right in creating tables, loading data into them, and retrieving data from them. This section describes the basic principles of entering commands, using several queries you can try out to familiarize yourself with how `mysql` works.

Here is a simple command that asks the server to tell you its version number and the current date. Type it in as shown here following the `mysql>` prompt and press Enter:

```
mysql> SELECT VERSION(), CURRENT_DATE;
+--------------+--------------+
| VERSION()    | CURRENT_DATE |
+--------------+--------------+
| 5.7.1-m4-log | 2012-12-25   |
+--------------+--------------+
1 row in set (0.01 sec)
mysql>
```

This query illustrates several things about `mysql`:

- A command normally consists of an SQL statement followed by a semicolon. (There are some exceptions where a semicolon may be omitted. `QUIT`, mentioned earlier, is one of them. We'll get to others later.)

- When you issue a command, `mysql` sends it to the server for execution and displays the results, then prints another `mysql>` prompt to indicate that it is ready for another command.

- `mysql` displays query output in tabular form (rows and columns). The first row contains labels for the columns. The rows following are the query results. Normally, column labels are the names of the columns you fetch from database tables. If you're retrieving the value of an expression rather than a table column (as in the example just shown), `mysql` labels the column using the expression itself.

- `mysql` shows how many rows were returned and how long the query took to execute, which gives you a rough idea of server performance. These values are imprecise because they represent wall clock time (not CPU or machine time), and because they are affected by factors such as server load and network latency. (For brevity, the “rows in set” line is sometimes not shown in the remaining examples in this chapter.)

Keywords may be entered in any lettercase. The following queries are equivalent:

```
mysql> SELECT VERSION(), CURRENT_DATE;
mysql> select version(), current_date;
mysql> SeLeCt vErSiOn(), current_DATE;
```

Here is another query. It demonstrates that you can use `mysql` as a simple calculator:

```
mysql> SELECT SIN(PI()/4), (4+1)*5;
+------------------+---------+
| SIN(PI()/4)      | (4+1)*5 |
+------------------+---------+
| 0.70710678118655 |      25 |
+------------------+---------+
1 row in set (0.02 sec)
```

The queries shown thus far have been relatively short, single-line statements. You can even enter multiple statements on a single line. Just end each one with a semicolon:

```
mysql> SELECT VERSION(); SELECT NOW();
+--------------+
| VERSION()    |
+--------------+
| 5.6.1-m4-log |
+--------------+
1 row in set (0.00 sec)
+---------------------+
| NOW()               |
+---------------------+
| 2010-08-06 12:17:13 |
+---------------------+
1 row in set (0.00 sec)
```

A command need not be given all on a single line, so lengthy commands that require several lines are not a problem. `mysql` determines where your statement ends by looking for the terminating semicolon, not by looking for the end of the input line. (In other words, `mysql` accepts free-format input: it collects input lines but does not execute them until it sees the semicolon.)

Here is a simple multiple-line statement:

```
mysql> SELECT
    -> USER()
    -> ,
    -> CURRENT_DATE;
+---------------+--------------+
| USER()        | CURRENT_DATE |
+---------------+--------------+
| jon@localhost | 2010-08-06   |
+---------------+--------------+
```

In this example, notice how the prompt changes from `mysql>` to `->` after you enter the first line of a multiple-line query. This is how `mysql` indicates that it has not yet seen a complete statement and is waiting for the rest. The prompt is your friend, because it provides valuable feedback. If you use that feedback, you can always be aware of what `mysql` is waiting for.

If you decide you do not want to execute a command that you are in the process of entering, cancel it by typing `\c`:

```
mysql> SELECT
    -> USER()
    -> \c
mysql>
```

Here, too, notice the prompt. It switches back to `mysql>` after you type `\c`, providing feedback to indicate that `mysql` is ready for a new command.

The following table shows each of the prompts you may see and summarizes what they mean about the state that `mysql` is in.

Prompt	Meaning
`mysql>`	Ready for new command.
`->`	Waiting for next line of multiple-line command.
`'>`	Waiting for next line, waiting for completion of a string that began with a single quote (“`'`”).
`">`	Waiting for next line, waiting for completion of a string that began with a double quote (“`"`”).
`` `> ``	Waiting for next line, waiting for completion of an identifier that began with a backtick (“`` ` ``”).
`/*>`	Waiting for next line, waiting for completion of a comment that began with `/*`.

Multiple-line statements commonly occur by accident when you intend to issue a command on a single line, but forget the terminating semicolon. In this case, `mysql` waits for more input:

```
mysql> SELECT USER()
    ->
```

If this happens to you (you think you've entered a statement but the only response is a `->` prompt), most likely `mysql` is waiting for the semicolon. If you don't notice what the prompt is telling you, you might sit there for a while before realizing what you need to do. Enter a semicolon to complete the statement, and `mysql` executes it:

```
mysql> SELECT USER()
    -> ;
+---------------+
| USER()        |
+---------------+
| jon@localhost |
+---------------+
```

The `'>` and `">` prompts occur during string collection (another way of saying that MySQL is waiting for completion of a string). In MySQL, you can write strings surrounded by either “`'`” or “`"`” characters (for example, `'hello'` or `"goodbye"`), and `mysql` lets you enter strings that span multiple lines. When you see a `'>` or `">` prompt, it means that you have entered a line containing a string that begins with a “`'`” or “`"`” quote character, but have not yet entered the matching quote that terminates the string. This often indicates that you have inadvertently left out a quote character. For example:

```
mysql> SELECT * FROM my_table WHERE name = 'Smith AND age < 30;
    '>
```

If you enter this `SELECT` statement, then press **Enter** and wait for the result, nothing happens. Instead of wondering why this query takes so long, notice the clue provided by the `'>` prompt. It tells you that `mysql` expects to see the rest of an unterminated string. (Do you see the error in the statement? The string `'Smith` is missing the second single quotation mark.)

At this point, what do you do? The simplest thing is to cancel the command. However, you cannot just type `\c` in this case, because `mysql` interprets it as part of the string that it is collecting. Instead, enter the closing quote character (so `mysql` knows you've finished the string), then type `\c`:

```
mysql> SELECT * FROM my_table WHERE name = 'Smith AND age < 30;
    '> '\c
mysql>
```

The prompt changes back to `mysql>`, indicating that `mysql` is ready for a new command.

The `` `> `` prompt is similar to the `'>` and `">` prompts, but indicates that you have begun but not completed a backtick-quoted identifier.

It is important to know what the `'>`, `">`, and `` `> `` prompts signify, because if you mistakenly enter an unterminated string, any further lines you type appear to be ignored by `mysql`—including a line containing `QUIT`. This can be quite confusing, especially if you do not know that you need to supply the terminating quote before you can cancel the current command.

3.3 Creating and Using a Database

Once you know how to enter commands, you are ready to access a database.

Suppose that you have several pets in your home (your menagerie) and you would like to keep track of various types of information about them. You can do so by creating tables to hold your data and loading them with the desired information. Then you can answer different sorts of questions about your animals by retrieving data from the tables. This section shows you how to perform the following operations:

- Create a database
- Create a table
- Load data into the table
- Retrieve data from the table in various ways
- Use multiple tables

The menagerie database is simple (deliberately), but it is not difficult to think of real-world situations in which a similar type of database might be used. For example, a database like this could be used by a farmer to keep track of livestock, or by a veterinarian to keep track of patient records. A menagerie distribution containing some of the queries and sample data used in the following sections can be obtained from the MySQL Web site. It is available in both compressed `tar` file and Zip formats at http://dev.mysql.com/doc/.

Use the `SHOW` statement to find out what databases currently exist on the server:

```
mysql> SHOW DATABASES;
+----------+
| Database |
+----------+
| mysql    |
| test     |
| tmp      |
+----------+
```

The `mysql` database describes user access privileges. The `test` database often is available as a workspace for users to try things out.

The list of databases displayed by the statement may be different on your machine; `SHOW DATABASES` does not show databases that you have no privileges for if you do not have the `SHOW DATABASES` privilege. See Section 13.7.5.13, “`SHOW DATABASES` Syntax”.

If the `test` database exists, try to access it:

```
mysql> USE test
Database changed
```

`USE`, like `QUIT`, does not require a semicolon. (You can terminate such statements with a semicolon if you like; it does no harm.) The `USE` statement is special in another way, too: it must be given on a single line.

You can use the `test` database (if you have access to it) for the examples that follow, but anything you create in that database can be removed by anyone else with access to it. For this reason, you should probably ask your MySQL administrator for permission to use a database of your own. Suppose that you want to call yours `menagerie`. The administrator needs to execute a command like this:

```
mysql> GRANT ALL ON menagerie.* TO 'your_mysql_name'@'your_client_host';
```

where `your_mysql_name` is the MySQL user name assigned to you and `your_client_host` is the host from which you connect to the server.

3.3.1 Creating and Selecting a Database

If the administrator creates your database for you when setting up your permissions, you can begin using it. Otherwise, you need to create it yourself:

```
mysql> CREATE DATABASE menagerie;
```

Under Unix, database names are case sensitive (unlike SQL keywords), so you must always refer to your database as menagerie, not as Menagerie, MENAGERIE, or some other variant. This is also true for table names. (Under Windows, this restriction does not apply, although you must refer to databases and tables using the same lettercase throughout a given query. However, for a variety of reasons, the recommended best practice is always to use the same lettercase that was used when the database was created.)

Note

If you get an error such as ERROR 1044 (42000): Access denied for user 'monty'@'localhost' to database 'menagerie' when attempting to create a database, this means that your user account does not have the necessary privileges to do so. Discuss this with the administrator or see Section 6.2, "The MySQL Access Privilege System".

Creating a database does not select it for use; you must do that explicitly. To make menagerie the current database, use this command:

```
mysql> USE menagerie
Database changed
```

Your database needs to be created only once, but you must select it for use each time you begin a mysql session. You can do this by issuing a USE statement as shown in the example. Alternatively, you can select the database on the command line when you invoke mysql. Just specify its name after any connection parameters that you might need to provide. For example:

```
shell> mysql -h host -u user -p menagerie
Enter password: ********
```

Important

menagerie in the command just shown is **not** your password. If you want to supply your password on the command line after the -p option, you must do so with no intervening space (for example, as -pmypassword, not as -p mypassword). However, putting your password on the command line is not recommended, because doing so exposes it to snooping by other users logged in on your machine.

Note

You can see at any time which database is currently selected using SELECT DATABASE().

3.3.2 Creating a Table

Creating the database is the easy part, but at this point it is empty, as SHOW TABLES tells you:

```
mysql> SHOW TABLES;
```

```
Empty set (0.00 sec)
```

The harder part is deciding what the structure of your database should be: what tables you need and what columns should be in each of them.

You want a table that contains a record for each of your pets. This can be called the `pet` table, and it should contain, as a bare minimum, each animal's name. Because the name by itself is not very interesting, the table should contain other information. For example, if more than one person in your family keeps pets, you might want to list each animal's owner. You might also want to record some basic descriptive information such as species and sex.

How about age? That might be of interest, but it is not a good thing to store in a database. Age changes as time passes, which means you'd have to update your records often. Instead, it is better to store a fixed value such as date of birth. Then, whenever you need age, you can calculate it as the difference between the current date and the birth date. MySQL provides functions for doing date arithmetic, so this is not difficult. Storing birth date rather than age has other advantages, too:

- You can use the database for tasks such as generating reminders for upcoming pet birthdays. (If you think this type of query is somewhat silly, note that it is the same question you might ask in the context of a business database to identify clients to whom you need to send out birthday greetings in the current week or month, for that computer-assisted personal touch.)

- You can calculate age in relation to dates other than the current date. For example, if you store death date in the database, you can easily calculate how old a pet was when it died.

You can probably think of other types of information that would be useful in the `pet` table, but the ones identified so far are sufficient: name, owner, species, sex, birth, and death.

Use a `CREATE TABLE` statement to specify the layout of your table:

```
mysql> CREATE TABLE pet (name VARCHAR(20), owner VARCHAR(20),
    -> species VARCHAR(20), sex CHAR(1), birth DATE, death DATE);
```

`VARCHAR` is a good choice for the `name`, `owner`, and `species` columns because the column values vary in length. The lengths in those column definitions need not all be the same, and need not be `20`. You can normally pick any length from `1` to `65535`, whatever seems most reasonable to you. If you make a poor choice and it turns out later that you need a longer field, MySQL provides an `ALTER TABLE` statement.

Several types of values can be chosen to represent sex in animal records, such as `'m'` and `'f'`, or perhaps `'male'` and `'female'`. It is simplest to use the single characters `'m'` and `'f'`.

The use of the `DATE` data type for the `birth` and `death` columns is a fairly obvious choice.

Once you have created a table, `SHOW TABLES` should produce some output:

```
mysql> SHOW TABLES;
+---------------------+
| Tables in menagerie |
+---------------------+
| pet                 |
+---------------------+
```

To verify that your table was created the way you expected, use a `DESCRIBE` statement:

```
mysql> DESCRIBE pet;
+---------+-------------+------+-----+---------+-------+
| Field   | Type        | Null | Key | Default | Extra |
+---------+-------------+------+-----+---------+-------+
| name    | varchar(20) | YES  |     | NULL    |       |
| owner   | varchar(20) | YES  |     | NULL    |       |
| species | varchar(20) | YES  |     | NULL    |       |
| sex     | char(1)     | YES  |     | NULL    |       |
| birth   | date        | YES  |     | NULL    |       |
| death   | date        | YES  |     | NULL    |       |
+---------+-------------+------+-----+---------+-------+
```

You can use `DESCRIBE` any time, for example, if you forget the names of the columns in your table or what types they have.

For more information about MySQL data types, see Chapter 11, *Data Types*.

3.3.3 Loading Data into a Table

After creating your table, you need to populate it. The `LOAD DATA` and `INSERT` statements are useful for this.

Suppose that your pet records can be described as shown here. (Observe that MySQL expects dates in `'YYYY-MM-DD'` format; this may be different from what you are used to.)

name	owner	species	sex	birth	death
Fluffy	Harold	cat	f	1993-02-04	
Claws	Gwen	cat	m	1994-03-17	
Buffy	Harold	dog	f	1989-05-13	
Fang	Benny	dog	m	1990-08-27	
Bowser	Diane	dog	m	1979-08-31	1995-07-29
Chirpy	Gwen	bird	f	1998-09-11	
Whistler	Gwen	bird		1997-12-09	
Slim	Benny	snake	m	1996-04-29	

Because you are beginning with an empty table, an easy way to populate it is to create a text file containing a row for each of your animals, then load the contents of the file into the table with a single statement.

You could create a text file `pet.txt` containing one record per line, with values separated by tabs, and given in the order in which the columns were listed in the `CREATE TABLE` statement. For missing values (such as unknown sexes or death dates for animals that are still living), you can use `NULL` values. To represent these in your text file, use `\N` (backslash, capital-N). For example, the record for Whistler the bird would look like this (where the whitespace between values is a single tab character):

```
Whistler        Gwen    bird    \N      1997-12-09      \N
```

To load the text file `pet.txt` into the `pet` table, use this statement:

```
mysql> LOAD DATA LOCAL INFILE '/path/pet.txt' INTO TABLE pet;
```

If you created the file on Windows with an editor that uses `\r\n` as a line terminator, you should use this statement instead:

```
mysql> LOAD DATA LOCAL INFILE '/path/pet.txt' INTO TABLE pet
    -> LINES TERMINATED BY '\r\n';
```

(On an Apple machine running OS X, you would likely want to use LINES TERMINATED BY '\r'.)

You can specify the column value separator and end of line marker explicitly in the LOAD DATA statement if you wish, but the defaults are tab and linefeed. These are sufficient for the statement to read the file pet.txt properly.

If the statement fails, it is likely that your MySQL installation does not have local file capability enabled by default. See Section 6.1.6, “Security Issues with LOAD DATA LOCAL”, for information on how to change this.

When you want to add new records one at a time, the INSERT statement is useful. In its simplest form, you supply values for each column, in the order in which the columns were listed in the CREATE TABLE statement. Suppose that Diane gets a new hamster named “Puffball.” You could add a new record using an INSERT statement like this:

```
mysql> INSERT INTO pet
    -> VALUES ('Puffball','Diane','hamster','f','1999-03-30',NULL);
```

String and date values are specified as quoted strings here. Also, with INSERT, you can insert NULL directly to represent a missing value. You do not use \N like you do with LOAD DATA.

From this example, you should be able to see that there would be a lot more typing involved to load your records initially using several INSERT statements rather than a single LOAD DATA statement.

3.3.4 Retrieving Information from a Table

The SELECT statement is used to pull information from a table. The general form of the statement is:

```
SELECT what_to_select
FROM which_table
WHERE conditions_to_satisfy;
```

what_to_select indicates what you want to see. This can be a list of columns, or * to indicate “all columns.” *which_table* indicates the table from which you want to retrieve data. The WHERE clause is optional. If it is present, *conditions_to_satisfy* specifies one or more conditions that rows must satisfy to qualify for retrieval.

3.3.4.1 Selecting All Data

The simplest form of SELECT retrieves everything from a table:

```
mysql> SELECT * FROM pet;
+----------+--------+---------+------+------------+------------+
| name     | owner  | species | sex  | birth      | death      |
+----------+--------+---------+------+------------+------------+
| Fluffy   | Harold | cat     | f    | 1993-02-04 | NULL       |
| Claws    | Gwen   | cat     | m    | 1994-03-17 | NULL       |
| Buffy    | Harold | dog     | f    | 1989-05-13 | NULL       |
| Fang     | Benny  | dog     | m    | 1990-08-27 | NULL       |
| Bowser   | Diane  | dog     | m    | 1979-08-31 | 1995-07-29 |
| Chirpy   | Gwen   | bird    | f    | 1998-09-11 | NULL       |
| Whistler | Gwen   | bird    | NULL | 1997-12-09 | NULL       |
| Slim     | Benny  | snake   | m    | 1996-04-29 | NULL       |
| Puffball | Diane  | hamster | f    | 1999-03-30 | NULL       |
+----------+--------+---------+------+------------+------------+
```

This form of SELECT is useful if you want to review your entire table, for example, after you've just loaded it with your initial data set. For example, you may happen to think that the birth date for Bowser doesn't seem quite right. Consulting your original pedigree papers, you find that the correct birth year should be 1989, not 1979.

There are at least two ways to fix this:

- Edit the file pet.txt to correct the error, then empty the table and reload it using DELETE and LOAD DATA:

  ```
  mysql> DELETE FROM pet;
  mysql> LOAD DATA LOCAL INFILE 'pet.txt' INTO TABLE pet;
  ```

  However, if you do this, you must also re-enter the record for Puffball.

- Fix only the erroneous record with an UPDATE statement:

  ```
  mysql> UPDATE pet SET birth = '1989-08-31' WHERE name = 'Bowser';
  ```

  The UPDATE changes only the record in question and does not require you to reload the table.

3.3.4.2 Selecting Particular Rows

As shown in the preceding section, it is easy to retrieve an entire table. Just omit the WHERE clause from the SELECT statement. But typically you don't want to see the entire table, particularly when it becomes large. Instead, you're usually more interested in answering a particular question, in which case you specify some constraints on the information you want. Let's look at some selection queries in terms of questions about your pets that they answer.

You can select only particular rows from your table. For example, if you want to verify the change that you made to Bowser's birth date, select Bowser's record like this:

```
mysql> SELECT * FROM pet WHERE name = 'Bowser';
+--------+-------+---------+------+------------+------------+
| name   | owner | species | sex  | birth      | death      |
+--------+-------+---------+------+------------+------------+
| Bowser | Diane | dog     | m    | 1989-08-31 | 1995-07-29 |
+--------+-------+---------+------+------------+------------+
```

The output confirms that the year is correctly recorded as 1989, not 1979.

String comparisons normally are case-insensitive, so you can specify the name as 'bowser', 'BOWSER', and so forth. The query result is the same.

You can specify conditions on any column, not just name. For example, if you want to know which animals were born during or after 1998, test the birth column:

```
mysql> SELECT * FROM pet WHERE birth >= '1998-1-1';
+----------+-------+---------+------+------------+-------+
| name     | owner | species | sex  | birth      | death |
+----------+-------+---------+------+------------+-------+
| Chirpy   | Gwen  | bird    | f    | 1998-09-11 | NULL  |
| Puffball | Diane | hamster | f    | 1999-03-30 | NULL  |
+----------+-------+---------+------+------------+-------+
```

You can combine conditions, for example, to locate female dogs:

```
mysql> SELECT * FROM pet WHERE species = 'dog' AND sex = 'f';
+-------+--------+---------+------+------------+-------+
| name  | owner  | species | sex  | birth      | death |
+-------+--------+---------+------+------------+-------+
| Buffy | Harold | dog     | f    | 1989-05-13 | NULL  |
+-------+--------+---------+------+------------+-------+
```

The preceding query uses the AND logical operator. There is also an OR operator:

```
mysql> SELECT * FROM pet WHERE species = 'snake' OR species = 'bird';
+----------+-------+---------+------+------------+-------+
| name     | owner | species | sex  | birth      | death |
+----------+-------+---------+------+------------+-------+
| Chirpy   | Gwen  | bird    | f    | 1998-09-11 | NULL  |
| Whistler | Gwen  | bird    | NULL | 1997-12-09 | NULL  |
| Slim     | Benny | snake   | m    | 1996-04-29 | NULL  |
+----------+-------+---------+------+------------+-------+
```

AND and OR may be intermixed, although AND has higher precedence than OR. If you use both operators, it is a good idea to use parentheses to indicate explicitly how conditions should be grouped:

```
mysql> SELECT * FROM pet WHERE (species = 'cat' AND sex = 'm')
    -> OR (species = 'dog' AND sex = 'f');
+-------+--------+---------+------+------------+-------+
| name  | owner  | species | sex  | birth      | death |
+-------+--------+---------+------+------------+-------+
| Claws | Gwen   | cat     | m    | 1994-03-17 | NULL  |
| Buffy | Harold | dog     | f    | 1989-05-13 | NULL  |
+-------+--------+---------+------+------------+-------+
```

3.3.4.3 Selecting Particular Columns

If you do not want to see entire rows from your table, just name the columns in which you are interested, separated by commas. For example, if you want to know when your animals were born, select the name and birth columns:

```
mysql> SELECT name, birth FROM pet;
+----------+------------+
| name     | birth      |
+----------+------------+
| Fluffy   | 1993-02-04 |
| Claws    | 1994-03-17 |
| Buffy    | 1989-05-13 |
| Fang     | 1990-08-27 |
| Bowser   | 1989-08-31 |
| Chirpy   | 1998-09-11 |
| Whistler | 1997-12-09 |
| Slim     | 1996-04-29 |
| Puffball | 1999-03-30 |
+----------+------------+
```

To find out who owns pets, use this query:

```
mysql> SELECT owner FROM pet;
+--------+
| owner  |
+--------+
| Harold |
| Gwen   |
| Harold |
| Benny  |
| Diane  |
| Gwen   |
| Gwen   |
| Benny  |
| Diane  |
+--------+
```

Notice that the query simply retrieves the `owner` column from each record, and some of them appear more than once. To minimize the output, retrieve each unique output record just once by adding the keyword `DISTINCT`:

```
mysql> SELECT DISTINCT owner FROM pet;
+--------+
| owner  |
+--------+
| Benny  |
| Diane  |
| Gwen   |
| Harold |
+--------+
```

You can use a `WHERE` clause to combine row selection with column selection. For example, to get birth dates for dogs and cats only, use this query:

```
mysql> SELECT name, species, birth FROM pet
    -> WHERE species = 'dog' OR species = 'cat';
+--------+---------+------------+
| name   | species | birth      |
+--------+---------+------------+
| Fluffy | cat     | 1993-02-04 |
| Claws  | cat     | 1994-03-17 |
| Buffy  | dog     | 1989-05-13 |
| Fang   | dog     | 1990-08-27 |
| Bowser | dog     | 1989-08-31 |
+--------+---------+------------+
```

3.3.4.4 Sorting Rows

You may have noticed in the preceding examples that the result rows are displayed in no particular order. It is often easier to examine query output when the rows are sorted in some meaningful way. To sort a result, use an `ORDER BY` clause.

Here are animal birthdays, sorted by date:

```
mysql> SELECT name, birth FROM pet ORDER BY birth;
+----------+------------+
| name     | birth      |
+----------+------------+
| Buffy    | 1989-05-13 |
| Bowser   | 1989-08-31 |
| Fang     | 1990-08-27 |
| Fluffy   | 1993-02-04 |
| Claws    | 1994-03-17 |
| Slim     | 1996-04-29 |
| Whistler | 1997-12-09 |
| Chirpy   | 1998-09-11 |
| Puffball | 1999-03-30 |
+----------+------------+
```

On character type columns, sorting—like all other comparison operations—is normally performed in a case-insensitive fashion. This means that the order is undefined for columns that are identical except for their case. You can force a case-sensitive sort for a column by using `BINARY` like so: `ORDER BY BINARY col_name`.

The default sort order is ascending, with smallest values first. To sort in reverse (descending) order, add the `DESC` keyword to the name of the column you are sorting by:

```
mysql> SELECT name, birth FROM pet ORDER BY birth DESC;
+----------+------------+
| name     | birth      |
+----------+------------+
| Puffball | 1999-03-30 |
| Chirpy   | 1998-09-11 |
| Whistler | 1997-12-09 |
| Slim     | 1996-04-29 |
| Claws    | 1994-03-17 |
| Fluffy   | 1993-02-04 |
| Fang     | 1990-08-27 |
| Bowser   | 1989-08-31 |
| Buffy    | 1989-05-13 |
+----------+------------+
```

You can sort on multiple columns, and you can sort different columns in different directions. For example, to sort by type of animal in ascending order, then by birth date within animal type in descending order (youngest animals first), use the following query:

```
mysql> SELECT name, species, birth FROM pet
    -> ORDER BY species, birth DESC;
+----------+---------+------------+
| name     | species | birth      |
+----------+---------+------------+
| Chirpy   | bird    | 1998-09-11 |
| Whistler | bird    | 1997-12-09 |
| Claws    | cat     | 1994-03-17 |
| Fluffy   | cat     | 1993-02-04 |
| Fang     | dog     | 1990-08-27 |
| Bowser   | dog     | 1989-08-31 |
| Buffy    | dog     | 1989-05-13 |
| Puffball | hamster | 1999-03-30 |
| Slim     | snake   | 1996-04-29 |
+----------+---------+------------+
```

The `DESC` keyword applies only to the column name immediately preceding it (`birth`); it does not affect the `species` column sort order.

3.3.4.5 Date Calculations

MySQL provides several functions that you can use to perform calculations on dates, for example, to calculate ages or extract parts of dates.

To determine how many years old each of your pets is, use the `TIMESTAMPDIFF()` function. Its arguments are the unit in which you want the result expressed, and the two date for which to take the difference. The following query shows, for each pet, the birth date, the current date, and the age in years. An *alias* (`age`) is used to make the final output column label more meaningful.

```
mysql> SELECT name, birth, CURDATE(),
    -> TIMESTAMPDIFF(YEAR,birth,CURDATE()) AS age
    -> FROM pet;
+----------+------------+------------+------+
| name     | birth      | CURDATE()  | age  |
+----------+------------+------------+------+
| Fluffy   | 1993-02-04 | 2003-08-19 |   10 |
| Claws    | 1994-03-17 | 2003-08-19 |    9 |
| Buffy    | 1989-05-13 | 2003-08-19 |   14 |
| Fang     | 1990-08-27 | 2003-08-19 |   12 |
| Bowser   | 1989-08-31 | 2003-08-19 |   13 |
| Chirpy   | 1998-09-11 | 2003-08-19 |    4 |
| Whistler | 1997-12-09 | 2003-08-19 |    5 |
| Slim     | 1996-04-29 | 2003-08-19 |    7 |
| Puffball | 1999-03-30 | 2003-08-19 |    4 |
+----------+------------+------------+------+
```

The query works, but the result could be scanned more easily if the rows were presented in some order. This can be done by adding an `ORDER BY name` clause to sort the output by name:

```
mysql> SELECT name, birth, CURDATE(),
    -> TIMESTAMPDIFF(YEAR,birth,CURDATE()) AS age
    -> FROM pet ORDER BY name;
+----------+------------+------------+------+
| name     | birth      | CURDATE()  | age  |
+----------+------------+------------+------+
| Bowser   | 1989-08-31 | 2003-08-19 |   13 |
| Buffy    | 1989-05-13 | 2003-08-19 |   14 |
| Chirpy   | 1998-09-11 | 2003-08-19 |    4 |
| Claws    | 1994-03-17 | 2003-08-19 |    9 |
| Fang     | 1990-08-27 | 2003-08-19 |   12 |
| Fluffy   | 1993-02-04 | 2003-08-19 |   10 |
| Puffball | 1999-03-30 | 2003-08-19 |    4 |
| Slim     | 1996-04-29 | 2003-08-19 |    7 |
| Whistler | 1997-12-09 | 2003-08-19 |    5 |
+----------+------------+------------+------+
```

To sort the output by `age` rather than `name`, just use a different `ORDER BY` clause:

```
mysql> SELECT name, birth, CURDATE(),
    -> TIMESTAMPDIFF(YEAR,birth,CURDATE()) AS age
    -> FROM pet ORDER BY age;
+----------+------------+------------+------+
| name     | birth      | CURDATE()  | age  |
+----------+------------+------------+------+
| Chirpy   | 1998-09-11 | 2003-08-19 |    4 |
| Puffball | 1999-03-30 | 2003-08-19 |    4 |
| Whistler | 1997-12-09 | 2003-08-19 |    5 |
| Slim     | 1996-04-29 | 2003-08-19 |    7 |
| Claws    | 1994-03-17 | 2003-08-19 |    9 |
| Fluffy   | 1993-02-04 | 2003-08-19 |   10 |
| Fang     | 1990-08-27 | 2003-08-19 |   12 |
| Bowser   | 1989-08-31 | 2003-08-19 |   13 |
| Buffy    | 1989-05-13 | 2003-08-19 |   14 |
+----------+------------+------------+------+
```

A similar query can be used to determine age at death for animals that have died. You determine which animals these are by checking whether the `death` value is `NULL`. Then, for those with non-`NULL` values, compute the difference between the `death` and `birth` values:

```
mysql> SELECT name, birth, death,
    -> TIMESTAMPDIFF(YEAR,birth,death) AS age
    -> FROM pet WHERE death IS NOT NULL ORDER BY age;
```

```
+--------+------------+------------+------+
| name   | birth      | death      | age  |
+--------+------------+------------+------+
| Bowser | 1989-08-31 | 1995-07-29 |    5 |
+--------+------------+------------+------+
```

The query uses `death IS NOT NULL` rather than `death <> NULL` because `NULL` is a special value that cannot be compared using the usual comparison operators. This is discussed later. See Section 3.3.4.6, “Working with `NULL` Values”.

What if you want to know which animals have birthdays next month? For this type of calculation, year and day are irrelevant; you simply want to extract the month part of the `birth` column. MySQL provides several functions for extracting parts of dates, such as `YEAR()`, `MONTH()`, and `DAYOFMONTH()`. `MONTH()` is the appropriate function here. To see how it works, run a simple query that displays the value of both `birth` and `MONTH(birth)`:

```
mysql> SELECT name, birth, MONTH(birth) FROM pet;
+----------+------------+--------------+
| name     | birth      | MONTH(birth) |
+----------+------------+--------------+
| Fluffy   | 1993-02-04 |            2 |
| Claws    | 1994-03-17 |            3 |
| Buffy    | 1989-05-13 |            5 |
| Fang     | 1990-08-27 |            8 |
| Bowser   | 1989-08-31 |            8 |
| Chirpy   | 1998-09-11 |            9 |
| Whistler | 1997-12-09 |           12 |
| Slim     | 1996-04-29 |            4 |
| Puffball | 1999-03-30 |            3 |
+----------+------------+--------------+
```

Finding animals with birthdays in the upcoming month is also simple. Suppose that the current month is April. Then the month value is `4` and you can look for animals born in May (month `5`) like this:

```
mysql> SELECT name, birth FROM pet WHERE MONTH(birth) = 5;
+-------+------------+
| name  | birth      |
+-------+------------+
| Buffy | 1989-05-13 |
+-------+------------+
```

There is a small complication if the current month is December. You cannot merely add one to the month number (`12`) and look for animals born in month `13`, because there is no such month. Instead, you look for animals born in January (month `1`).

You can write the query so that it works no matter what the current month is, so that you do not have to use the number for a particular month. `DATE_ADD()` enables you to add a time interval to a given date. If you add a month to the value of `CURDATE()`, then extract the month part with `MONTH()`, the result produces the month in which to look for birthdays:

```
mysql> SELECT name, birth FROM pet
    -> WHERE MONTH(birth) = MONTH(DATE_ADD(CURDATE(),INTERVAL 1 MONTH));
```

A different way to accomplish the same task is to add `1` to get the next month after the current one after using the modulo function (`MOD`) to wrap the month value to `0` if it is currently `12`:

```
mysql> SELECT name, birth FROM pet
    -> WHERE MONTH(birth) = MOD(MONTH(CURDATE()), 12) + 1;
```

MONTH() returns a number between 1 and 12. And MOD(something,12) returns a number between 0 and 11. So the addition has to be after the MOD(), otherwise we would go from November (11) to January (1).

3.3.4.6 Working with NULL Values

The NULL value can be surprising until you get used to it. Conceptually, NULL means “a missing unknown value” and it is treated somewhat differently from other values.

To test for NULL, use the IS NULL and IS NOT NULL operators, as shown here:

```
mysql> SELECT 1 IS NULL, 1 IS NOT NULL;
+-----------+---------------+
| 1 IS NULL | 1 IS NOT NULL |
+-----------+---------------+
|         0 |             1 |
+-----------+---------------+
```

You cannot use arithmetic comparison operators such as =, <, or <> to test for NULL. To demonstrate this for yourself, try the following query:

```
mysql> SELECT 1 = NULL, 1 <> NULL, 1 < NULL, 1 > NULL;
+----------+-----------+----------+----------+
| 1 = NULL | 1 <> NULL | 1 < NULL | 1 > NULL |
+----------+-----------+----------+----------+
|     NULL |      NULL |     NULL |     NULL |
+----------+-----------+----------+----------+
```

Because the result of any arithmetic comparison with NULL is also NULL, you cannot obtain any meaningful results from such comparisons.

In MySQL, 0 or NULL means false and anything else means true. The default truth value from a boolean operation is 1.

This special treatment of NULL is why, in the previous section, it was necessary to determine which animals are no longer alive using death IS NOT NULL instead of death <> NULL.

Two NULL values are regarded as equal in a GROUP BY.

When doing an ORDER BY, NULL values are presented first if you do ORDER BY ... ASC and last if you do ORDER BY ... DESC.

A common error when working with NULL is to assume that it is not possible to insert a zero or an empty string into a column defined as NOT NULL, but this is not the case. These are in fact values, whereas NULL means “not having a value.” You can test this easily enough by using IS [NOT] NULL as shown:

```
mysql> SELECT 0 IS NULL, 0 IS NOT NULL, '' IS NULL, '' IS NOT NULL;
+-----------+---------------+------------+----------------+
| 0 IS NULL | 0 IS NOT NULL | '' IS NULL | '' IS NOT NULL |
+-----------+---------------+------------+----------------+
|         0 |             1 |          0 |              1 |
+-----------+---------------+------------+----------------+
```

Thus it is entirely possible to insert a zero or empty string into a NOT NULL column, as these are in fact NOT NULL. See Section C.5.5.3, “Problems with NULL Values”.

3.3.4.7 Pattern Matching

MySQL provides standard SQL pattern matching as well as a form of pattern matching based on extended regular expressions similar to those used by Unix utilities such as vi, grep, and sed.

SQL pattern matching enables you to use "_" to match any single character and "%" to match an arbitrary number of characters (including zero characters). In MySQL, SQL patterns are case-insensitive by default. Some examples are shown here. You do not use = or <> when you use SQL patterns; use the LIKE or NOT LIKE comparison operators instead.

To find names beginning with "b":

```
mysql> SELECT * FROM pet WHERE name LIKE 'b%';
+--------+--------+---------+------+------------+------------+
| name   | owner  | species | sex  | birth      | death      |
+--------+--------+---------+------+------------+------------+
| Buffy  | Harold | dog     | f    | 1989-05-13 | NULL       |
| Bowser | Diane  | dog     | m    | 1989-08-31 | 1995-07-29 |
+--------+--------+---------+------+------------+------------+
```

To find names ending with "fy":

```
mysql> SELECT * FROM pet WHERE name LIKE '%fy';
+--------+--------+---------+------+------------+-------+
| name   | owner  | species | sex  | birth      | death |
+--------+--------+---------+------+------------+-------+
| Fluffy | Harold | cat     | f    | 1993-02-04 | NULL  |
| Buffy  | Harold | dog     | f    | 1989-05-13 | NULL  |
+--------+--------+---------+------+------------+-------+
```

To find names containing a "w":

```
mysql> SELECT * FROM pet WHERE name LIKE '%w%';
+----------+-------+---------+------+------------+------------+
| name     | owner | species | sex  | birth      | death      |
+----------+-------+---------+------+------------+------------+
| Claws    | Gwen  | cat     | m    | 1994-03-17 | NULL       |
| Bowser   | Diane | dog     | m    | 1989-08-31 | 1995-07-29 |
| Whistler | Gwen  | bird    | NULL | 1997-12-09 | NULL       |
+----------+-------+---------+------+------------+------------+
```

To find names containing exactly five characters, use five instances of the "_" pattern character:

```
mysql> SELECT * FROM pet WHERE name LIKE '_____';
+-------+--------+---------+------+------------+-------+
| name  | owner  | species | sex  | birth      | death |
+-------+--------+---------+------+------------+-------+
| Claws | Gwen   | cat     | m    | 1994-03-17 | NULL  |
| Buffy | Harold | dog     | f    | 1989-05-13 | NULL  |
+-------+--------+---------+------+------------+-------+
```

The other type of pattern matching provided by MySQL uses extended regular expressions. When you test for a match for this type of pattern, use the REGEXP and NOT REGEXP operators (or RLIKE and NOT RLIKE, which are synonyms).

The following list describes some characteristics of extended regular expressions:

- "." matches any single character.
- A character class "[...]" matches any character within the brackets. For example, "[abc]" matches "a", "b", or "c". To name a range of characters, use a dash. "[a-z]" matches any letter, whereas "[0-9]" matches any digit.

- "*" matches zero or more instances of the thing preceding it. For example, "x*" matches any number of "x" characters, "[0-9]*" matches any number of digits, and ".*" matches any number of anything.

- A REGEXP pattern match succeeds if the pattern matches anywhere in the value being tested. (This differs from a LIKE pattern match, which succeeds only if the pattern matches the entire value.)

- To anchor a pattern so that it must match the beginning or end of the value being tested, use "^" at the beginning or "$" at the end of the pattern.

To demonstrate how extended regular expressions work, the LIKE queries shown previously are rewritten here to use REGEXP.

To find names beginning with "b", use "^" to match the beginning of the name:

```
mysql> SELECT * FROM pet WHERE name REGEXP '^b';
+--------+--------+---------+------+------------+------------+
| name   | owner  | species | sex  | birth      | death      |
+--------+--------+---------+------+------------+------------+
| Buffy  | Harold | dog     | f    | 1989-05-13 | NULL       |
| Bowser | Diane  | dog     | m    | 1989-08-31 | 1995-07-29 |
+--------+--------+---------+------+------------+------------+
```

If you really want to force a REGEXP comparison to be case sensitive, use the BINARY keyword to make one of the strings a binary string. This query matches only lowercase "b" at the beginning of a name:

```
mysql> SELECT * FROM pet WHERE name REGEXP BINARY '^b';
```

To find names ending with "fy", use "$" to match the end of the name:

```
mysql> SELECT * FROM pet WHERE name REGEXP 'fy$';
+--------+--------+---------+------+------------+-------+
| name   | owner  | species | sex  | birth      | death |
+--------+--------+---------+------+------------+-------+
| Fluffy | Harold | cat     | f    | 1993-02-04 | NULL  |
| Buffy  | Harold | dog     | f    | 1989-05-13 | NULL  |
+--------+--------+---------+------+------------+-------+
```

To find names containing a "w", use this query:

```
mysql> SELECT * FROM pet WHERE name REGEXP 'w';
+----------+-------+---------+------+------------+------------+
| name     | owner | species | sex  | birth      | death      |
+----------+-------+---------+------+------------+------------+
| Claws    | Gwen  | cat     | m    | 1994-03-17 | NULL       |
| Bowser   | Diane | dog     | m    | 1989-08-31 | 1995-07-29 |
| Whistler | Gwen  | bird    | NULL | 1997-12-09 | NULL       |
+----------+-------+---------+------+------------+------------+
```

Because a regular expression pattern matches if it occurs anywhere in the value, it is not necessary in the previous query to put a wildcard on either side of the pattern to get it to match the entire value like it would be if you used an SQL pattern.

To find names containing exactly five characters, use "^" and "$" to match the beginning and end of the name, and five instances of "." in between:

```
mysql> SELECT * FROM pet WHERE name REGEXP '^.....$';
+-------+--------+---------+------+------------+-------+
| name  | owner  | species | sex  | birth      | death |
+-------+--------+---------+------+------------+-------+
| Claws | Gwen   | cat     | m    | 1994-03-17 | NULL  |
| Buffy | Harold | dog     | f    | 1989-05-13 | NULL  |
+-------+--------+---------+------+------------+-------+
```

You could also write the previous query using the `{n}` (“repeat-*n*-times”) operator:

```
mysql> SELECT * FROM pet WHERE name REGEXP '^.{5}$';
+-------+--------+---------+------+------------+-------+
| name  | owner  | species | sex  | birth      | death |
+-------+--------+---------+------+------------+-------+
| Claws | Gwen   | cat     | m    | 1994-03-17 | NULL  |
| Buffy | Harold | dog     | f    | 1989-05-13 | NULL  |
+-------+--------+---------+------+------------+-------+
```

Section 12.5.2, “Regular Expressions”, provides more information about the syntax for regular expressions.

3.3.4.8 Counting Rows

Databases are often used to answer the question, “How often does a certain type of data occur in a table?” For example, you might want to know how many pets you have, or how many pets each owner has, or you might want to perform various kinds of census operations on your animals.

Counting the total number of animals you have is the same question as “How many rows are in the `pet` table?” because there is one record per pet. `COUNT(*)` counts the number of rows, so the query to count your animals looks like this:

```
mysql> SELECT COUNT(*) FROM pet;
+----------+
| COUNT(*) |
+----------+
|        9 |
+----------+
```

Earlier, you retrieved the names of the people who owned pets. You can use `COUNT()` if you want to find out how many pets each owner has:

```
mysql> SELECT owner, COUNT(*) FROM pet GROUP BY owner;
+--------+----------+
| owner  | COUNT(*) |
+--------+----------+
| Benny  |        2 |
| Diane  |        2 |
| Gwen   |        3 |
| Harold |        2 |
+--------+----------+
```

The preceding query uses `GROUP BY` to group all records for each `owner`. The use of `COUNT()` in conjunction with `GROUP BY` is useful for characterizing your data under various groupings. The following examples show different ways to perform animal census operations.

Number of animals per species:

```
mysql> SELECT species, COUNT(*) FROM pet GROUP BY species;
+---------+----------+
| species | COUNT(*) |
+---------+----------+
| bird    |        2 |
| cat     |        2 |
| dog     |        3 |
| hamster |        1 |
| snake   |        1 |
+---------+----------+
```

Number of animals per sex:

```
mysql> SELECT sex, COUNT(*) FROM pet GROUP BY sex;
+------+----------+
| sex  | COUNT(*) |
+------+----------+
| NULL |        1 |
| f    |        4 |
| m    |        4 |
+------+----------+
```

(In this output, `NULL` indicates that the sex is unknown.)

Number of animals per combination of species and sex:

```
mysql> SELECT species, sex, COUNT(*) FROM pet GROUP BY species, sex;
+---------+------+----------+
| species | sex  | COUNT(*) |
+---------+------+----------+
| bird    | NULL |        1 |
| bird    | f    |        1 |
| cat     | f    |        1 |
| cat     | m    |        1 |
| dog     | f    |        1 |
| dog     | m    |        2 |
| hamster | f    |        1 |
| snake   | m    |        1 |
+---------+------+----------+
```

You need not retrieve an entire table when you use `COUNT()`. For example, the previous query, when performed just on dogs and cats, looks like this:

```
mysql> SELECT species, sex, COUNT(*) FROM pet
    -> WHERE species = 'dog' OR species = 'cat'
    -> GROUP BY species, sex;
+---------+------+----------+
| species | sex  | COUNT(*) |
+---------+------+----------+
| cat     | f    |        1 |
| cat     | m    |        1 |
| dog     | f    |        1 |
| dog     | m    |        2 |
+---------+------+----------+
```

Or, if you wanted the number of animals per sex only for animals whose sex is known:

```
mysql> SELECT species, sex, COUNT(*) FROM pet
    -> WHERE sex IS NOT NULL
    -> GROUP BY species, sex;
+---------+------+----------+
| species | sex  | COUNT(*) |
+---------+------+----------+
| bird    | f    |        1 |
| cat     | f    |        1 |
| cat     | m    |        1 |
| dog     | f    |        1 |
| dog     | m    |        2 |
| hamster | f    |        1 |
| snake   | m    |        1 |
+---------+------+----------+
```

If you name columns to select in addition to the COUNT() value, a GROUP BY clause should be present that names those same columns. Otherwise, the following occurs:

- If the ONLY_FULL_GROUP_BY SQL mode is enabled, an error occurs:

```
mysql> SET sql_mode = 'ONLY_FULL_GROUP_BY';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT owner, COUNT(*) FROM pet;
ERROR 1140 (42000): Mixing of GROUP columns (MIN(),MAX(),COUNT()...)
with no GROUP columns is illegal if there is no GROUP BY clause
```

- If ONLY_FULL_GROUP_BY is not enabled, the query is processed by treating all rows as a single group, but the value selected for each named column is indeterminate. The server is free to select the value from any row:

```
mysql> SET sql_mode = '';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT owner, COUNT(*) FROM pet;
+--------+----------+
| owner  | COUNT(*) |
+--------+----------+
| Harold |        8 |
+--------+----------+
1 row in set (0.00 sec)
```

See also Section 12.17.3, “MySQL Extensions to GROUP BY”.

3.3.4.9 Using More Than one Table

The pet table keeps track of which pets you have. If you want to record other information about them, such as events in their lives like visits to the vet or when litters are born, you need another table. What should this table look like? It needs to contain the following information:

- The pet name so that you know which animal each event pertains to.
- A date so that you know when the event occurred.
- A field to describe the event.
- An event type field, if you want to be able to categorize events.

Given these considerations, the CREATE TABLE statement for the event table might look like this:

```
mysql> CREATE TABLE event (name VARCHAR(20), date DATE,
    -> type VARCHAR(15), remark VARCHAR(255));
```

As with the pet table, it is easiest to load the initial records by creating a tab-delimited text file containing the following information.

name	date	type	remark
Fluffy	1995-05-15	litter	4 kittens, 3 female, 1 male
Buffy	1993-06-23	litter	5 puppies, 2 female, 3 male

name	date	type	remark
Buffy	1994-06-19	litter	3 puppies, 3 female
Chirpy	1999-03-21	vet	needed beak straightened
Slim	1997-08-03	vet	broken rib
Bowser	1991-10-12	kennel	
Fang	1991-10-12	kennel	
Fang	1998-08-28	birthday	Gave him a new chew toy
Claws	1998-03-17	birthday	Gave him a new flea collar
Whistler	1998-12-09	birthday	First birthday

Load the records like this:

```
mysql> LOAD DATA LOCAL INFILE 'event.txt' INTO TABLE event;
```

Based on what you have learned from the queries that you have run on the `pet` table, you should be able to perform retrievals on the records in the `event` table; the principles are the same. But when is the `event` table by itself insufficient to answer questions you might ask?

Suppose that you want to find out the ages at which each pet had its litters. We saw earlier how to calculate ages from two dates. The litter date of the mother is in the `event` table, but to calculate her age on that date you need her birth date, which is stored in the `pet` table. This means the query requires both tables:

```
mysql> SELECT pet.name,
    -> (YEAR(date)-YEAR(birth)) - (RIGHT(date,5)<RIGHT(birth,5)) AS age,
    -> remark
    -> FROM pet INNER JOIN event
    ->   ON pet.name = event.name
    -> WHERE event.type = 'litter';
+--------+------+-----------------------------+
| name   | age  | remark                      |
+--------+------+-----------------------------+
| Fluffy |    2 | 4 kittens, 3 female, 1 male |
| Buffy  |    4 | 5 puppies, 2 female, 3 male |
| Buffy  |    5 | 3 puppies, 3 female         |
+--------+------+-----------------------------+
```

There are several things to note about this query:

- The `FROM` clause joins two tables because the query needs to pull information from both of them.
- When combining (joining) information from multiple tables, you need to specify how records in one table can be matched to records in the other. This is easy because they both have a `name` column. The query uses an `ON` clause to match up records in the two tables based on the `name` values.

  The query uses an `INNER JOIN` to combine the tables. An `INNER JOIN` permits rows from either table to appear in the result if and only if both tables meet the conditions specified in the `ON` clause. In this example, the `ON` clause specifies that the `name` column in the `pet` table must match the `name` column in the `event` table. If a name appears in one table but not the other, the row will not appear in the result because the condition in the `ON` clause fails.
- Because the `name` column occurs in both tables, you must be specific about which table you mean when referring to the column. This is done by prepending the table name to the column name.

You need not have two different tables to perform a join. Sometimes it is useful to join a table to itself, if you want to compare records in a table to other records in that same table. For example, to find breeding pairs among your pets, you can join the pet table with itself to produce candidate pairs of males and females of like species:

```
mysql> SELECT p1.name, p1.sex, p2.name, p2.sex, p1.species
    -> FROM pet AS p1 INNER JOIN pet AS p2
    ->   ON p1.species = p2.species AND p1.sex = 'f' AND p2.sex = 'm';
+--------+------+--------+------+---------+
| name   | sex  | name   | sex  | species |
+--------+------+--------+------+---------+
| Fluffy | f    | Claws  | m    | cat     |
| Buffy  | f    | Fang   | m    | dog     |
| Buffy  | f    | Bowser | m    | dog     |
+--------+------+--------+------+---------+
```

In this query, we specify aliases for the table name to refer to the columns and keep straight which instance of the table each column reference is associated with.

3.4 Getting Information About Databases and Tables

What if you forget the name of a database or table, or what the structure of a given table is (for example, what its columns are called)? MySQL addresses this problem through several statements that provide information about the databases and tables it supports.

You have previously seen SHOW DATABASES, which lists the databases managed by the server. To find out which database is currently selected, use the DATABASE() function:

```
mysql> SELECT DATABASE();
+------------+
| DATABASE() |
+------------+
| menagerie  |
+------------+
```

If you have not yet selected any database, the result is NULL.

To find out what tables the default database contains (for example, when you are not sure about the name of a table), use this command:

```
mysql> SHOW TABLES;
+---------------------+
| Tables_in_menagerie |
+---------------------+
| event               |
| pet                 |
+---------------------+
```

The name of the column in the output produced by this statement is always Tables_in_*db_name*, where *db_name* is the name of the database. See Section 13.7.5.36, “SHOW TABLES Syntax”, for more information.

If you want to find out about the structure of a table, the DESCRIBE statement is useful; it displays information about each of a table's columns:

```
mysql> DESCRIBE pet;
+---------+-------------+------+-----+---------+-------+
| Field   | Type        | Null | Key | Default | Extra |
+---------+-------------+------+-----+---------+-------+
| name    | varchar(20) | YES  |     | NULL    |       |
| owner   | varchar(20) | YES  |     | NULL    |       |
| species | varchar(20) | YES  |     | NULL    |       |
| sex     | char(1)     | YES  |     | NULL    |       |
| birth   | date        | YES  |     | NULL    |       |
| death   | date        | YES  |     | NULL    |       |
+---------+-------------+------+-----+---------+-------+
```

Field indicates the column name, Type is the data type for the column, NULL indicates whether the column can contain NULL values, Key indicates whether the column is indexed, and Default specifies the column's default value. Extra displays special information about columns: If a column was created with the AUTO_INCREMENT option, the value will be auto_increment rather than empty.

DESC is a short form of DESCRIBE. See Section 13.8.1, “DESCRIBE Syntax”, for more information.

You can obtain the CREATE TABLE statement necessary to create an existing table using the SHOW CREATE TABLE statement. See Section 13.7.5.10, “SHOW CREATE TABLE Syntax”.

If you have indexes on a table, SHOW INDEX FROM *tbl_name* produces information about them. See Section 13.7.5.21, “SHOW INDEX Syntax”, for more about this statement.

3.5 Using mysql in Batch Mode

In the previous sections, you used mysql interactively to enter queries and view the results. You can also run mysql in batch mode. To do this, put the commands you want to run in a file, then tell mysql to read its input from the file:

```
shell> mysql < batch-file
```

If you are running mysql under Windows and have some special characters in the file that cause problems, you can do this:

```
C:\> mysql -e "source batch-file"
```

If you need to specify connection parameters on the command line, the command might look like this:

```
shell> mysql -h host -u user -p < batch-file
Enter password: ********
```

When you use mysql this way, you are creating a script file, then executing the script.

If you want the script to continue even if some of the statements in it produce errors, you should use the --force command-line option.

Why use a script? Here are a few reasons:

- If you run a query repeatedly (say, every day or every week), making it a script enables you to avoid retyping it each time you execute it.
- You can generate new queries from existing ones that are similar by copying and editing script files.
- Batch mode can also be useful while you're developing a query, particularly for multiple-line commands or multiple-statement sequences of commands. If you make a mistake, you don't have to retype everything. Just edit your script to correct the error, then tell mysql to execute it again.

- If you have a query that produces a lot of output, you can run the output through a pager rather than watching it scroll off the top of your screen:

  ```
  shell> mysql < batch-file | more
  ```

- You can catch the output in a file for further processing:

  ```
  shell> mysql < batch-file > mysql.out
  ```

- You can distribute your script to other people so that they can also run the commands.

- Some situations do not allow for interactive use, for example, when you run a query from a `cron` job. In this case, you must use batch mode.

The default output format is different (more concise) when you run `mysql` in batch mode than when you use it interactively. For example, the output of `SELECT DISTINCT species FROM pet` looks like this when `mysql` is run interactively:

```
+---------+
| species |
+---------+
| bird    |
| cat     |
| dog     |
| hamster |
| snake   |
+---------+
```

In batch mode, the output looks like this instead:

```
species
bird
cat
dog
hamster
snake
```

If you want to get the interactive output format in batch mode, use `mysql -t`. To echo to the output the commands that are executed, use `mysql -vvv`.

You can also use scripts from the `mysql` prompt by using the `source` command or `\.` command:

```
mysql> source filename;
mysql> \. filename
```

See Section 4.5.1.5, “Executing SQL Statements from a Text File”, for more information.

3.6 Examples of Common Queries

Here are examples of how to solve some common problems with MySQL.

Some of the examples use the table `shop` to hold the price of each article (item number) for certain traders (dealers). Supposing that each trader has a single fixed price per article, then (`article`, `dealer`) is a primary key for the records.

Start the command-line tool `mysql` and select a database:

```
shell> mysql your-database-name
```

(In most MySQL installations, you can use the database named `test`).

You can create and populate the example table with these statements:

```
CREATE TABLE shop (
    article INT(4) UNSIGNED ZEROFILL DEFAULT '0000' NOT NULL,
    dealer  CHAR(20)                 DEFAULT ''     NOT NULL,
    price   DOUBLE(16,2)             DEFAULT '0.00' NOT NULL,
    PRIMARY KEY(article, dealer));
INSERT INTO shop VALUES
    (1,'A',3.45),(1,'B',3.99),(2,'A',10.99),(3,'B',1.45),
    (3,'C',1.69),(3,'D',1.25),(4,'D',19.95);
```

After issuing the statements, the table should have the following contents:

```
SELECT * FROM shop;

+---------+--------+-------+
| article | dealer | price |
+---------+--------+-------+
|    0001 | A      |  3.45 |
|    0001 | B      |  3.99 |
|    0002 | A      | 10.99 |
|    0003 | B      |  1.45 |
|    0003 | C      |  1.69 |
|    0003 | D      |  1.25 |
|    0004 | D      | 19.95 |
+---------+--------+-------+
```

3.6.1 The Maximum Value for a Column

“What is the highest item number?”

```
SELECT MAX(article) AS article FROM shop;

+---------+
| article |
+---------+
|       4 |
+---------+
```

3.6.2 The Row Holding the Maximum of a Certain Column

Task: Find the number, dealer, and price of the most expensive article.

This is easily done with a subquery:

```
SELECT article, dealer, price
FROM   shop
WHERE  price=(SELECT MAX(price) FROM shop);

+---------+--------+-------+
| article | dealer | price |
+---------+--------+-------+
|    0004 | D      | 19.95 |
+---------+--------+-------+
```

Other solutions are to use a `LEFT JOIN` or to sort all rows descending by price and get only the first row using the MySQL-specific `LIMIT` clause:

```
SELECT s1.article, s1.dealer, s1.price
FROM shop s1
LEFT JOIN shop s2 ON s1.price < s2.price
WHERE s2.article IS NULL;

SELECT article, dealer, price
FROM shop
ORDER BY price DESC
LIMIT 1;
```

Note

If there were several most expensive articles, each with a price of 19.95, the `LIMIT` solution would show only one of them.

3.6.3 Maximum of Column per Group

Task: Find the highest price per article.

```
SELECT article, MAX(price) AS price
FROM   shop
GROUP BY article;

+---------+-------+
| article | price |
+---------+-------+
|    0001 |  3.99 |
|    0002 | 10.99 |
|    0003 |  1.69 |
|    0004 | 19.95 |
+---------+-------+
```

3.6.4 The Rows Holding the Group-wise Maximum of a Certain Column

Task: For each article, find the dealer or dealers with the most expensive price.

This problem can be solved with a subquery like this one:

```
SELECT article, dealer, price
FROM   shop s1
WHERE  price=(SELECT MAX(s2.price)
              FROM shop s2
              WHERE s1.article = s2.article);

+---------+--------+-------+
| article | dealer | price |
+---------+--------+-------+
|    0001 | B      |  3.99 |
|    0002 | A      | 10.99 |
|    0003 | C      |  1.69 |
|    0004 | D      | 19.95 |
+---------+--------+-------+
```

The preceding example uses a correlated subquery, which can be inefficient (see Section 13.2.10.7, “Correlated Subqueries”). Other possibilities for solving the problem are to use an uncorrelated subquery in the `FROM` clause or a `LEFT JOIN`.

Uncorrelated subquery:

```
SELECT s1.article, dealer, s1.price
FROM shop s1
JOIN (
  SELECT article, MAX(price) AS price
  FROM shop
  GROUP BY article) AS s2
  ON s1.article = s2.article AND s1.price = s2.price;
```

`LEFT JOIN`:

```
SELECT s1.article, s1.dealer, s1.price
FROM shop s1
LEFT JOIN shop s2 ON s1.article = s2.article AND s1.price < s2.price
WHERE s2.article IS NULL;
```

The `LEFT JOIN` works on the basis that when `s1.price` is at its maximum value, there is no `s2.price` with a greater value and the `s2` rows values will be `NULL`. See Section 13.2.9.2, "`JOIN` Syntax".

3.6.5 Using User-Defined Variables

You can employ MySQL user variables to remember results without having to store them in temporary variables in the client. (See Section 9.4, "User-Defined Variables".)

For example, to find the articles with the highest and lowest price you can do this:

```
mysql> SELECT @min_price:=MIN(price),@max_price:=MAX(price) FROM shop;
mysql> SELECT * FROM shop WHERE price=@min_price OR price=@max_price;
+---------+--------+-------+
| article | dealer | price |
+---------+--------+-------+
|    0003 | D      |  1.25 |
|    0004 | D      | 19.95 |
+---------+--------+-------+
```

Note

It is also possible to store the name of a database object such as a table or a column in a user variable and then to use this variable in an SQL statement; however, this requires the use of a prepared statement. See Section 13.5, "SQL Syntax for Prepared Statements", for more information.

3.6.6 Using Foreign Keys

In MySQL, `InnoDB` tables support checking of foreign key constraints. See Section 14.2, "The `InnoDB` Storage Engine", and Section 1.8.2.4, "Foreign Key Differences".

A foreign key constraint is not required merely to join two tables. For storage engines other than `InnoDB`, it is possible when defining a column to use a `REFERENCES` *`tbl_name`*`(`*`col_name`*`)` clause, which has no actual effect, and *serves only as a memo or comment to you that the column which you are currently defining is intended to refer to a column in another table*. It is extremely important to realize when using this syntax that:

- MySQL does not perform any sort of `CHECK` to make sure that *`col_name`* actually exists in *`tbl_name`* (or even that *`tbl_name`* itself exists).

- MySQL does not perform any sort of action on *`tbl_name`* such as deleting rows in response to actions taken on rows in the table which you are defining; in other words, this syntax induces no `ON DELETE` or `ON UPDATE` behavior whatsoever. (Although you can write an `ON DELETE` or `ON UPDATE` clause as part of the `REFERENCES` clause, it is also ignored.)

- This syntax creates a *column*; it does **not** create any sort of index or key.

You can use a column so created as a join column, as shown here:

```
CREATE TABLE person (
    id SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,
    name CHAR(60) NOT NULL,
    PRIMARY KEY (id)
);

CREATE TABLE shirt (
    id SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,
    style ENUM('t-shirt', 'polo', 'dress') NOT NULL,
    color ENUM('red', 'blue', 'orange', 'white', 'black') NOT NULL,
    owner SMALLINT UNSIGNED NOT NULL REFERENCES person(id),
    PRIMARY KEY (id)
);

INSERT INTO person VALUES (NULL, 'Antonio Paz');

SELECT @last := LAST_INSERT_ID();

INSERT INTO shirt VALUES
(NULL, 'polo', 'blue', @last),
(NULL, 'dress', 'white', @last),
(NULL, 't-shirt', 'blue', @last);

INSERT INTO person VALUES (NULL, 'Lilliana Angelovska');

SELECT @last := LAST_INSERT_ID();

INSERT INTO shirt VALUES
(NULL, 'dress', 'orange', @last),
(NULL, 'polo', 'red', @last),
(NULL, 'dress', 'blue', @last),
(NULL, 't-shirt', 'white', @last);

SELECT * FROM person;
+----+---------------------+
| id | name                |
+----+---------------------+
|  1 | Antonio Paz         |
|  2 | Lilliana Angelovska |
+----+---------------------+

SELECT * FROM shirt;
+----+---------+--------+-------+
| id | style   | color  | owner |
+----+---------+--------+-------+
|  1 | polo    | blue   |     1 |
|  2 | dress   | white  |     1 |
|  3 | t-shirt | blue   |     1 |
|  4 | dress   | orange |     2 |
|  5 | polo    | red    |     2 |
|  6 | dress   | blue   |     2 |
|  7 | t-shirt | white  |     2 |
+----+---------+--------+-------+


SELECT s.* FROM person p INNER JOIN shirt s
```

```
   ON s.owner = p.id
 WHERE p.name LIKE 'Lilliana%'
   AND s.color <> 'white';

+----+-------+--------+-------+
| id | style | color  | owner |
+----+-------+--------+-------+
|  4 | dress | orange |     2 |
|  5 | polo  | red    |     2 |
|  6 | dress | blue   |     2 |
+----+-------+--------+-------+
```

When used in this fashion, the `REFERENCES` clause is not displayed in the output of `SHOW CREATE TABLE` or `DESCRIBE`:

```
SHOW CREATE TABLE shirt\G
*************************** 1. row ***************************
Table: shirt
Create Table: CREATE TABLE `shirt` (
`id` smallint(5) unsigned NOT NULL auto_increment,
`style` enum('t-shirt','polo','dress') NOT NULL,
`color` enum('red','blue','orange','white','black') NOT NULL,
`owner` smallint(5) unsigned NOT NULL,
PRIMARY KEY  (`id`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1
```

The use of `REFERENCES` in this way as a comment or “reminder” in a column definition works with `MyISAM` tables.

3.6.7 Searching on Two Keys

An `OR` using a single key is well optimized, as is the handling of `AND`.

The one tricky case is that of searching on two different keys combined with `OR`:

```
SELECT field1_index, field2_index FROM test_table
WHERE field1_index = '1' OR  field2_index = '1'
```

This case is optimized. See Section 8.2.1.4, “Index Merge Optimization”.

You can also solve the problem efficiently by using a `UNION` that combines the output of two separate `SELECT` statements. See Section 13.2.9.4, “`UNION` Syntax”.

Each `SELECT` searches only one key and can be optimized:

```
SELECT field1_index, field2_index
    FROM test_table WHERE field1_index = '1'
UNION
SELECT field1_index, field2_index
    FROM test_table WHERE field2_index = '1';
```

3.6.8 Calculating Visits Per Day

The following example shows how you can use the bit group functions to calculate the number of days per month a user has visited a Web page.

```
CREATE TABLE t1 (year YEAR(4), month INT(2) UNSIGNED ZEROFILL,
             day INT(2) UNSIGNED ZEROFILL);
INSERT INTO t1 VALUES(2000,1,1),(2000,1,20),(2000,1,30),(2000,2,2),
            (2000,2,23),(2000,2,23);
```

The example table contains year-month-day values representing visits by users to the page. To determine how many different days in each month these visits occur, use this query:

```
SELECT year,month,BIT_COUNT(BIT_OR(1<<day)) AS days FROM t1
       GROUP BY year,month;
```

Which returns:

```
+------+-------+------+
| year | month | days |
+------+-------+------+
| 2000 |    01 |    3 |
| 2000 |    02 |    2 |
+------+-------+------+
```

The query calculates how many different days appear in the table for each year/month combination, with automatic removal of duplicate entries.

3.6.9 Using AUTO_INCREMENT

The AUTO_INCREMENT attribute can be used to generate a unique identity for new rows:

```
CREATE TABLE animals (
     id MEDIUMINT NOT NULL AUTO_INCREMENT,
     name CHAR(30) NOT NULL,
     PRIMARY KEY (id)
);

INSERT INTO animals (name) VALUES
    ('dog'),('cat'),('penguin'),
    ('lax'),('whale'),('ostrich');

SELECT * FROM animals;
```

Which returns:

```
+----+---------+
| id | name    |
+----+---------+
|  1 | dog     |
|  2 | cat     |
|  3 | penguin |
|  4 | lax     |
|  5 | whale   |
|  6 | ostrich |
+----+---------+
```

No value was specified for the AUTO_INCREMENT column, so MySQL assigned sequence numbers automatically. You can also explicitly assign 0 to the column to generate sequence numbers. If the column is declared NOT NULL, it is also possible to assign NULL to the column to generate sequence numbers.

You can retrieve the most recent AUTO_INCREMENT value with the LAST_INSERT_ID() SQL function or the mysql_insert_id() C API function. These functions are connection-specific, so their return values are not affected by another connection which is also performing inserts.

Use the smallest integer data type for the AUTO_INCREMENT column that is large enough to hold the maximum sequence value you will need. When the column reaches the upper limit of the data type, the next attempt to generate a sequence number fails. Use the UNSIGNED attribute if possible to allow a greater range. For example, if you use TINYINT, the maximum permissible sequence number is 127. For TINYINT UNSIGNED, the maximum is 255. See Section 11.2.1, “Integer Types (Exact Value) - INTEGER, INT, SMALLINT, TINYINT, MEDIUMINT, BIGINT” for the ranges of all the integer types.

Note

For a multiple-row insert, LAST_INSERT_ID() and mysql_insert_id() actually return the AUTO_INCREMENT key from the *first* of the inserted rows. This enables multiple-row inserts to be reproduced correctly on other servers in a replication setup.

To start with an AUTO_INCREMENT value other than 1, set that value with CREATE TABLE or ALTER TABLE, like this:

```
mysql> ALTER TABLE tbl AUTO_INCREMENT = 100;
```

InnoDB Notes

For InnoDB tables, be careful if you modify the column containing the auto-increment value in the middle of a sequence of INSERT statements. For example, if you use an UPDATE statement to put a new, larger value in the auto-increment column, a subsequent INSERT could encounter a “Duplicate entry” error. The test whether an auto-increment value is already present occurs if you do a DELETE followed by more INSERT statements, or when you COMMIT the transaction, but not after an UPDATE statement.

MyISAM Notes

- For MyISAM tables, you can specify AUTO_INCREMENT on a secondary column in a multiple-column index. In this case, the generated value for the AUTO_INCREMENT column is calculated as MAX(*auto_increment_column*) + 1 WHERE prefix=*given-prefix*. This is useful when you want to put data into ordered groups.

  ```
  CREATE TABLE animals (
      grp ENUM('fish','mammal','bird') NOT NULL,
      id MEDIUMINT NOT NULL AUTO_INCREMENT,
      name CHAR(30) NOT NULL,
      PRIMARY KEY (grp,id)
  ) ENGINE=MyISAM;

  INSERT INTO animals (grp,name) VALUES
      ('mammal','dog'),('mammal','cat'),
      ('bird','penguin'),('fish','lax'),('mammal','whale'),
      ('bird','ostrich');

  SELECT * FROM animals ORDER BY grp,id;
  ```

  Which returns:

  ```
  +--------+----+---------+
  | grp    | id | name    |
  +--------+----+---------+
  | fish   |  1 | lax     |
  | mammal |  1 | dog     |
  | mammal |  2 | cat     |
  | mammal |  3 | whale   |
  | bird   |  1 | penguin |
  | bird   |  2 | ostrich |
  +--------+----+---------+
  ```

In this case (when the AUTO_INCREMENT column is part of a multiple-column index), AUTO_INCREMENT values are reused if you delete the row with the biggest AUTO_INCREMENT value in any group. This happens even for MyISAM tables, for which AUTO_INCREMENT values normally are not reused.

- If the AUTO_INCREMENT column is part of multiple indexes, MySQL generates sequence values using the index that begins with the AUTO_INCREMENT column, if there is one. For example, if the animals table contained indexes PRIMARY KEY (grp, id) and INDEX (id), MySQL would ignore the PRIMARY KEY for generating sequence values. As a result, the table would contain a single sequence, not a sequence per grp value.

Further Reading

More information about AUTO_INCREMENT is available here:

- How to assign the AUTO_INCREMENT attribute to a column: Section 13.1.14, "CREATE TABLE Syntax", and Section 13.1.6, "ALTER TABLE Syntax".
- How AUTO_INCREMENT behaves depending on the NO_AUTO_VALUE_ON_ZERO SQL mode: Section 5.1.7, "Server SQL Modes".
- How to use the LAST_INSERT_ID() function to find the row that contains the most recent AUTO_INCREMENT value: Section 12.14, "Information Functions".
- Setting the AUTO_INCREMENT value to be used: Section 5.1.4, "Server System Variables".
- AUTO_INCREMENT and replication: Section 16.4.1.1, "Replication and AUTO_INCREMENT".
- Server-system variables related to AUTO_INCREMENT (auto_increment_increment and auto_increment_offset) that can be used for replication: Section 5.1.4, "Server System Variables".

3.7 Using MySQL with Apache

There are programs that let you authenticate your users from a MySQL database and also let you write your log files into a MySQL table.

You can change the Apache logging format to be easily readable by MySQL by putting the following into the Apache configuration file:

```
LogFormat \
        "\"%h\",%{%Y%m%d%H%M%S}t,%>s,\"%b\",\"%{Content-Type}o\",  \
        \"%U\",\"%{Referer}i\",\"%{User-Agent}i\""
```

To load a log file in that format into MySQL, you can use a statement something like this:

```
LOAD DATA INFILE '/local/access_log' INTO TABLE tbl_name
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"' ESCAPED BY '\\'
```

The named table should be created to have columns that correspond to those that the LogFormat line writes to the log file.

Chapter 4 MySQL Programs

Table of Contents

This chapter provides a brief overview of the MySQL command-line programs provided by Oracle Corporation. It also discusses the general syntax for specifying options when you run these programs. Most programs have options that are specific to their own operation, but the option syntax is similar for all of them. Finally, the chapter provides more detailed descriptions of individual programs, including which options they recognize.

4.1 Overview of MySQL Programs

There are many different programs in a MySQL installation. This section provides a brief overview of them. Later sections provide a more detailed description of each one. Each program's description indicates its invocation syntax and the options that it supports.

Most MySQL distributions include all of these programs, except for those programs that are platform-specific. (For example, the server startup scripts are not used on Windows.) The exception is that RPM distributions are more specialized. There is one RPM for the server, another for client programs, and so forth. If you appear to be missing one or more programs, see Chapter 2, *Installing and Upgrading MySQL*, for information on types of distributions and what they contain. It may be that you have a distribution that does not include all programs and you need to install an additional package.

Each MySQL program takes many different options. Most programs provide a `--help` option that you can use to get a description of the program's different options. For example, try `mysql --help`.

You can override default option values for MySQL programs by specifying options on the command line or in an option file. See Section 4.2, “Using MySQL Programs”, for general information on invoking programs and specifying program options.

The MySQL server, `mysqld`, is the main program that does most of the work in a MySQL installation. The server is accompanied by several related scripts that assist you in starting and stopping the server:

- `mysqld`

  The SQL daemon (that is, the MySQL server). To use client programs, `mysqld` must be running, because clients gain access to databases by connecting to the server. See Section 4.3.1, “`mysqld` — The MySQL Server”.

- `mysqld_safe`

  A server startup script. `mysqld_safe` attempts to start `mysqld`. See Section 4.3.2, “`mysqld_safe` — MySQL Server Startup Script”.

- `mysql.server`

  A server startup script. This script is used on systems that use System V-style run directories containing scripts that start system services for particular run levels. It invokes `mysqld_safe` to start the MySQL server. See Section 4.3.3, “`mysql.server` — MySQL Server Startup Script”.

- `mysqld_multi`

  A server startup script that can start or stop multiple servers installed on the system. See Section 4.3.4, “`mysqld_multi` — Manage Multiple MySQL Servers”.

Several programs perform setup operations during MySQL installation or upgrading:

- `comp_err`

  This program is used during the MySQL build/installation process. It compiles error message files from the error source files. See Section 4.4.1, “`comp_err` — Compile MySQL Error Message File”.

- `mysql_install_db`

  This script creates the MySQL database, initializes the grant tables with default privileges, and sets up the `InnoDB` system tablespace. It is usually executed only once, when first installing MySQL on a system. See Section 4.4.3, "`mysql_install_db` — Initialize MySQL Data Directory", Section 2.9.1, "Postinstallation Procedures for Unix-like Systems", and Section 4.4.3, "`mysql_install_db` — Initialize MySQL Data Directory".

- `mysql_plugin`

  This program configures MySQL server plugins. See Section 4.4.4, "`mysql_plugin` — Configure MySQL Server Plugins".

- `mysql_secure_installation`

  This program enables you to improve the security of your MySQL installation. SQL. See Section 4.4.5, "`mysql_secure_installation` — Improve MySQL Installation Security".

- `mysql_tzinfo_to_sql`

  This program loads the time zone tables in the `mysql` database using the contents of the host system *zoneinfo* database (the set of files describing time zones). SQL. See Section 4.4.6, "`mysql_tzinfo_to_sql` — Load the Time Zone Tables".

- `mysql_upgrade`

  This program is used after a MySQL upgrade operation. It checks tables for incompatibilities and repairs them if necessary, and updates the grant tables with any changes that have been made in newer versions of MySQL. See Section 4.4.7, "`mysql_upgrade` — Check and Upgrade MySQL Tables".

MySQL client programs that connect to the MySQL server:

- `mysql`

  The command-line tool for interactively entering SQL statements or executing them from a file in batch mode. See Section 4.5.1, "`mysql` — The MySQL Command-Line Tool".

- `mysqladmin`

  A client that performs administrative operations, such as creating or dropping databases, reloading the grant tables, flushing tables to disk, and reopening log files. `mysqladmin` can also be used to retrieve version, process, and status information from the server. See Section 4.5.2, "`mysqladmin` — Client for Administering a MySQL Server".

- `mysqlcheck`

  A table-maintenance client that checks, repairs, analyzes, and optimizes tables. See Section 4.5.3, "`mysqlcheck` — A Table Maintenance Program".

- `mysqldump`

  A client that dumps a MySQL database into a file as SQL, text, or XML. See Section 4.5.4, "`mysqldump` — A Database Backup Program".

- `mysqlimport`

  A client that imports text files into their respective tables using `LOAD DATA INFILE`. See Section 4.5.5, "`mysqlimport` — A Data Import Program".

- mysqlshow

  A client that displays information about databases, tables, columns, and indexes. See Section 4.5.6, “mysqlshow — Display Database, Table, and Column Information”.

- mysqlslap

  A client that is designed to emulate client load for a MySQL server and report the timing of each stage. It works as if multiple clients are accessing the server. See Section 4.5.7, “mysqlslap — Load Emulation Client”.

MySQL administrative and utility programs:

- innochecksum

  An offline InnoDB offline file checksum utility. See Section 4.6.1, “innochecksum — Offline InnoDB File Checksum Utility”.

- myisam_ftdump

  A utility that displays information about full-text indexes in MyISAM tables. See Section 4.6.2, “myisam_ftdump — Display Full-Text Index information”.

- myisamchk

  A utility to describe, check, optimize, and repair MyISAM tables. See Section 4.6.3, “myisamchk — MyISAM Table-Maintenance Utility”.

- myisamlog

  A utility that processes the contents of a MyISAM log file. See Section 4.6.4, “myisamlog — Display MyISAM Log File Contents”.

- myisampack

  A utility that compresses MyISAM tables to produce smaller read-only tables. See Section 4.6.5, “myisampack — Generate Compressed, Read-Only MyISAM Tables”.

- mysql_config_editor

  A utility that enables you to store authentication credentials in a secure, encrypted login file named .mylogin.cnf. See Section 4.6.6, “mysql_config_editor — MySQL Configuration Utility”.

- mysqlbinlog

  A utility for reading statements from a binary log. The log of executed statements contained in the binary log files can be used to help recover from a crash. See Section 4.6.7, “mysqlbinlog — Utility for Processing Binary Log Files”.

- mysqldumpslow

  A utility to read and summarize the contents of a slow query log. See Section 4.6.8, “mysqldumpslow — Summarize Slow Query Log Files”.

- mysqlhotcopy

  A utility that quickly makes backups of MyISAM tables while the server is running. See Section 4.6.9, “mysqlhotcopy — A Database Backup Program”.

- mysql_waitpid

  A utility that kills the process with a given process ID. See Section 4.6.10, “mysql_waitpid — Kill Process and Wait for Its Termination”.

- mysql_zap

  A utility that kills processes that match a pattern. See Section 4.6.11, “mysql_zap — Kill Processes That Match a Pattern”.

MySQL program-development utilities:

- mysql_config

  A shell script that produces the option values needed when compiling MySQL programs. See Section 4.7.1, “mysql_config — Display Options for Compiling Clients”.

- my_print_defaults

  A utility that shows which options are present in option groups of option files. See Section 4.7.2, “my_print_defaults — Display Options from Option Files”.

- resolve_stack_dump

  A utility program that resolves a numeric stack trace dump to symbols. See Section 4.7.3, “resolve_stack_dump — Resolve Numeric Stack Trace Dump to Symbols”.

Miscellaneous utilities:

- perror

  A utility that displays the meaning of system or MySQL error codes. See Section 4.8.1, “perror — Explain Error Codes”.

- replace

  A utility program that performs string replacement in the input text. See Section 4.8.2, “replace — A String-Replacement Utility”.

- resolveip

  A utility program that resolves a host name to an IP address or vice versa. See Section 4.8.3, “resolveip — Resolve Host name to IP Address or Vice Versa”.

Oracle Corporation also provides the MySQL Workbench GUI tool, which is used to administer MySQL servers and databases, to create, execute, and evaluate queries, and to migrate schemas and data from other relational database management systems for use with MySQL. Additional GUI tools include MySQL Notifier for Microsoft Windows and MySQL for Excel.

MySQL client programs that communicate with the server using the MySQL client/server library use the following environment variables.

Environment Variable	Meaning
MYSQL_UNIX_PORT	The default Unix socket file; used for connections to localhost
MYSQL_TCP_PORT	The default port number; used for TCP/IP connections
MYSQL_PWD	The default password

Environment Variable	Meaning
MYSQL_DEBUG	Debug trace options when debugging
TMPDIR	The directory where temporary tables and files are created

For a full list of environment variables used by MySQL programs, see Section 2.11, "Environment Variables".

Use of MYSQL_PWD is insecure. See Section 6.1.2.1, "End-User Guidelines for Password Security".

4.2 Using MySQL Programs

4.2.1 Invoking MySQL Programs

To invoke a MySQL program from the command line (that is, from your shell or command prompt), enter the program name followed by any options or other arguments needed to instruct the program what you want it to do. The following commands show some sample program invocations. "shell>" represents the prompt for your command interpreter; it is not part of what you type. The particular prompt you see depends on your command interpreter. Typical prompts are $ for sh, ksh, or bash, % for csh or tcsh, and C:\> for the Windows command.com or cmd.exe command interpreters.

```
shell> mysql --user=root test
shell> mysqladmin extended-status variables
shell> mysqlshow --help
shell> mysqldump -u root personnel
```

Arguments that begin with a single or double dash ("-", "--") specify program options. Options typically indicate the type of connection a program should make to the server or affect its operational mode. Option syntax is described in Section 4.2.3, "Specifying Program Options".

Nonoption arguments (arguments with no leading dash) provide additional information to the program. For example, the mysql program interprets the first nonoption argument as a database name, so the command mysql --user=root test indicates that you want to use the test database.

Later sections that describe individual programs indicate which options a program supports and describe the meaning of any additional nonoption arguments.

Some options are common to a number of programs. The most frequently used of these are the --host (or -h), --user (or -u), and --password (or -p) options that specify connection parameters. They indicate the host where the MySQL server is running, and the user name and password of your MySQL account. All MySQL client programs understand these options; they enable you to specify which server to connect to and the account to use on that server. Other connection options are --port (or -P) to specify a TCP/IP port number and --socket (or -S) to specify a Unix socket file on Unix (or named pipe name on Windows). For more information on options that specify connection options, see Section 4.2.2, "Connecting to the MySQL Server".

You may find it necessary to invoke MySQL programs using the path name to the bin directory in which they are installed. This is likely to be the case if you get a "program not found" error whenever you attempt to run a MySQL program from any directory other than the bin directory. To make it more convenient to use MySQL, you can add the path name of the bin directory to your PATH environment variable setting. That enables you to run a program by typing only its name, not its entire path name. For example, if mysql is installed in /usr/local/mysql/bin, you can run the program by invoking it as mysql, and it is not necessary to invoke it as /usr/local/mysql/bin/mysql.

Consult the documentation for your command interpreter for instructions on setting your PATH variable. The syntax for setting environment variables is interpreter-specific. (Some information is given in

Section 4.2.4, “Setting Environment Variables”.) After modifying your PATH setting, open a new console window on Windows or log in again on Unix so that the setting goes into effect.

4.2.2 Connecting to the MySQL Server

For a client program to be able to connect to the MySQL server, it must use the proper connection parameters, such as the name of the host where the server is running and the user name and password of your MySQL account. Each connection parameter has a default value, but you can override them as necessary using program options specified either on the command line or in an option file.

The examples here use the mysql client program, but the principles apply to other clients such as mysqldump, mysqladmin, or mysqlshow.

This command invokes mysql without specifying any connection parameters explicitly:

```
shell> mysql
```

Because there are no parameter options, the default values apply:

- The default host name is localhost. On Unix, this has a special meaning, as described later.
- The default user name is ODBC on Windows or your Unix login name on Unix.
- No password is sent if neither -p nor --password is given.
- For mysql, the first nonoption argument is taken as the name of the default database. If there is no such option, mysql does not select a default database.

To specify the host name and user name explicitly, as well as a password, supply appropriate options on the command line:

```
shell> mysql --host=localhost --user=myname --password=mypass mydb
shell> mysql -h localhost -u myname -pmypass mydb
```

For password options, the password value is optional:

- If you use a -p or --password option and specify the password value, there must be *no space* between -p or --password= and the password following it.
- If you use a -p or --password option but do not specify the password value, the client program prompts you to enter the password. The password is not displayed as you enter it. This is more secure than giving the password on the command line. Other users on your system may be able to see a password specified on the command line by executing a command such as ps auxw. See Section 6.1.2.1, “End-User Guidelines for Password Security”.

As just mentioned, including the password value on the command line can be a security risk. To avoid this problem, specify the --password or -p option without any following password value:

```
shell> mysql --host=localhost --user=myname --password mydb
shell> mysql -h localhost -u myname -p mydb
```

When the password option has no password value, the client program prints a prompt and waits for you to enter the password. (In these examples, mydb is *not* interpreted as a password because it is separated from the preceding password option by a space.)

On some systems, the library routine that MySQL uses to prompt for a password automatically limits the password to eight characters. That is a problem with the system library, not with MySQL. Internally, MySQL does not have any limit for the length of the password. To work around the problem, change your MySQL password to a value that is eight or fewer characters long, or put your password in an option file.

On Unix, MySQL programs treat the host name `localhost` specially, in a way that is likely different from what you expect compared to other network-based programs. For connections to `localhost`, MySQL programs attempt to connect to the local server by using a Unix socket file. This occurs even if a `--port` or `-P` option is given to specify a port number. To ensure that the client makes a TCP/IP connection to the local server, use `--host` or `-h` to specify a host name value of `127.0.0.1`, or the IP address or name of the local server. You can also specify the connection protocol explicitly, even for `localhost`, by using the `--protocol=TCP` option. For example:

```
shell> mysql --host=127.0.0.1
shell> mysql --protocol=TCP
```

The `--protocol` option enables you to establish a particular type of connection even when the other options would normally default to some other protocol.

If the server is configured to accept IPv6 connections, client can connect over IPv6 using `--host=::1`. See Section 5.1.9, “IPv6 Support”.

On Windows, you can force a MySQL client to use a named-pipe connection by specifying the `--pipe` or `--protocol=PIPE` option, or by specifying `.` (period) as the host name. If named-pipe connections are not enabled, an error occurs. Use the `--socket` option to specify the name of the pipe if you do not want to use the default pipe name.

Connections to remote servers always use TCP/IP. This command connects to the server running on `remote.example.com` using the default port number (3306):

```
shell> mysql --host=remote.example.com
```

To specify a port number explicitly, use the `--port` or `-P` option:

```
shell> mysql --host=remote.example.com --port=13306
```

You can specify a port number for connections to a local server, too. However, as indicated previously, connections to `localhost` on Unix will use a socket file by default. You will need to force a TCP/IP connection as already described or any option that specifies a port number will be ignored.

For this command, the program uses a socket file on Unix and the `--port` option is ignored:

```
shell> mysql --port=13306 --host=localhost
```

To cause the port number to be used, invoke the program in either of these ways:

```
shell> mysql --port=13306 --host=127.0.0.1
shell> mysql --port=13306 --protocol=TCP
```

The following list summarizes the options that can be used to control how client programs connect to the server:

- `--host=`*`host_name`*, `-h` *`host_name`*

The host where the server is running. The default value is `localhost`.

- `--password[=pass_val]`, `-p[pass_val]`

  The password of the MySQL account. As described earlier, the password value is optional, but if given, there must be *no space* between `-p` or `--password=` and the password following it. The default is to send no password.

- `--pipe`, `-W`

  On Windows, connect to the server using a named pipe. The server must be started with the `--enable-named-pipe` option to enable named-pipe connections.

- `--port=port_num`, `-P port_num`

  The port number to use for the connection, for connections made using TCP/IP. The default port number is 3306.

- `--protocol={TCP|SOCKET|PIPE|MEMORY}`

  This option explicitly specifies a protocol to use for connecting to the server. It is useful when the other connection parameters normally would cause a protocol to be used other than the one you want. For example, connections on Unix to `localhost` are made using a Unix socket file by default:

  ```
  shell> mysql --host=localhost
  ```

  To force a TCP/IP connection to be used instead, specify a `--protocol` option:

  ```
  shell> mysql --host=localhost --protocol=TCP
  ```

  The following table shows the permissible `--protocol` option values and indicates the platforms on which each value may be used. The values are not case sensitive.

`--protocol` Value	Connection Protocol	Permissible Operating Systems
`TCP`	TCP/IP connection to local or remote server	All
`SOCKET`	Unix socket file connection to local server	Unix only
`PIPE`	Named-pipe connection to local or remote server	Windows only
`MEMORY`	Shared-memory connection to local server	Windows only

- `--shared-memory-base-name=name`

  On Windows, the shared-memory name to use, for connections made using shared memory to a local server. The default value is `MYSQL`. The shared-memory name is case sensitive.

  The server must be started with the `--shared-memory` option to enable shared-memory connections.

- `--socket=file_name`, `-S file_name`

  On Unix, the name of the Unix socket file to use, for connections made using a named pipe to a local server. The default Unix socket file name is `/tmp/mysql.sock`.

  On Windows, the name of the named pipe to use, for connections to a local server. The default Windows pipe name is `MySQL`. The pipe name is not case sensitive.

The server must be started with the --enable-named-pipe option to enable named-pipe connections.

- --ssl*

  Options that begin with --ssl are used for establishing a secure connection to the server using SSL, if the server is configured with SSL support. For details, see Section 6.3.11.4, "SSL Command Options".

- --user=*user_name*, -u *user_name*

  The user name of the MySQL account you want to use. The default user name is ODBC on Windows or your Unix login name on Unix.

It is possible to specify different default values to be used when you make a connection so that you need not enter them on the command line each time you invoke a client program. This can be done in a couple of ways:

- You can specify connection parameters in the [client] section of an option file. The relevant section of the file might look like this:

  ```
  [client]
  host=host_name
  user=user_name
  password=your_pass
  ```

  Section 4.2.3.3, "Using Option Files", discusses option files further.

- You can specify some connection parameters using environment variables. The host can be specified for mysql using MYSQL_HOST. The MySQL user name can be specified using USER (this is for Windows only). The password can be specified using MYSQL_PWD, although this is insecure; see Section 6.1.2.1, "End-User Guidelines for Password Security". For a list of variables, see Section 2.11, "Environment Variables".

4.2.3 Specifying Program Options

There are several ways to specify options for MySQL programs:

- List the options on the command line following the program name. This is common for options that apply to a specific invocation of the program.

- List the options in an option file that the program reads when it starts. This is common for options that you want the program to use each time it runs.

- List the options in environment variables (see Section 4.2.4, "Setting Environment Variables"). This method is useful for options that you want to apply each time the program runs. In practice, option files are used more commonly for this purpose, but Section 5.3.3, "Running Multiple MySQL Instances on Unix", discusses one situation in which environment variables can be very helpful. It describes a handy technique that uses such variables to specify the TCP/IP port number and Unix socket file for the server and for client programs.

Options are processed in order, so if an option is specified multiple times, the last occurrence takes precedence. The following command causes mysql to connect to the server running on localhost:

```
shell> mysql -h example.com -h localhost
```

If conflicting or related options are given, later options take precedence over earlier options. The following command runs mysql in "no column names" mode:

```
shell> mysql --column-names --skip-column-names
```

MySQL programs determine which options are given first by examining environment variables, then by reading option files, and then by checking the command line. This means that environment variables have the lowest precedence and command-line options the highest.

You can take advantage of the way that MySQL programs process options by specifying default option values for a program in an option file. That enables you to avoid typing them each time you run the program while enabling you to override the defaults if necessary by using command-line options.

> **Note**
>
> Prior to MySQL 5.7.2, program options could be specified in full or as any unambiguous prefix. For example, the `--compress` option could be given to `mysqldump` as `--compr`, but not as `--comp` because the latter is ambiguous. As of MySQL 5.7.2, option prefixes are no longer supported; only full options are accepted. This is because prefixes can cause problems when new options are implemented for programs and a prefix that is currently unambiguous might become ambiguous in the future.

4.2.3.1 Using Options on the Command Line

Program options specified on the command line follow these rules:

- Options are given after the command name.

- An option argument begins with one dash or two dashes, depending on whether it is a short form or long form of the option name. Many options have both short and long forms. For example, `-?` and `--help` are the short and long forms of the option that instructs a MySQL program to display its help message.

- Option names are case sensitive. `-v` and `-V` are both legal and have different meanings. (They are the corresponding short forms of the `--verbose` and `--version` options.)

- Some options take a value following the option name. For example, `-h localhost` or `--host=localhost` indicate the MySQL server host to a client program. The option value tells the program the name of the host where the MySQL server is running.

- For a long option that takes a value, separate the option name and the value by an “`=`” sign. For a short option that takes a value, the option value can immediately follow the option letter, or there can be a space between: `-hlocalhost` and `-h localhost` are equivalent. An exception to this rule is the option for specifying your MySQL password. This option can be given in long form as `--password=`*`pass_val`* or as `--password`. In the latter case (with no password value given), the program prompts you for the password. The password option also may be given in short form as `-p`*`pass_val`* or as `-p`. However, for the short form, if the password value is given, it must follow the option letter with *no intervening space*. The reason for this is that if a space follows the option letter, the program has no way to tell whether a following argument is supposed to be the password value or some other kind of argument. Consequently, the following two commands have two completely different meanings:

  ```
  shell> mysql -ptest
  shell> mysql -p test
  ```

  The first command instructs `mysql` to use a password value of `test`, but specifies no default database. The second instructs `mysql` to prompt for the password value and to use `test` as the default database.

- Within option names, dash (“-”) and underscore (“_”) may be used interchangeably. For example, --skip-grant-tables and --skip_grant_tables are equivalent. (However, the leading dashes cannot be given as underscores.)

- For options that take a numeric value, the value can be given with a suffix of K, M, or G (either uppercase or lowercase) to indicate a multiplier of 1024, 1024^2 or 1024^3. For example, the following command tells mysqladmin to ping the server 1024 times, sleeping 10 seconds between each ping:

  ```
  mysql> mysqladmin --count=1K --sleep=10 ping
  ```

Option values that contain spaces must be quoted when given on the command line. For example, the --execute (or -e) option can be used with mysql to pass SQL statements to the server. When this option is used, mysql executes the statements in the option value and exits. The statements must be enclosed by quotation marks. For example, you can use the following command to obtain a list of user accounts:

```
mysql> mysql -u root -p --execute="SELECT User, Host FROM mysql.user"
Enter password: ******
+------+-----------+
| User | Host      |
+------+-----------+
|      | gigan     |
| root | gigan     |
|      | localhost |
| jon  | localhost |
| root | localhost |
+------+-----------+
shell>
```

Note that the long form (--execute) is followed by an equals sign (=).

If you wish to use quoted values within a statement, you will either need to escape the inner quotation marks, or use a different type of quotation marks within the statement from those used to quote the statement itself. The capabilities of your command processor dictate your choices for whether you can use single or double quotation marks and the syntax for escaping quote characters. For example, if your command processor supports quoting with single or double quotation marks, you can use double quotation marks around the statement, and single quotation marks for any quoted values within the statement.

Multiple SQL statements may be passed in the option value on the command line, separated by semicolons:

```
shell> mysql -u root -p -e "SELECT VERSION();SELECT NOW()"
Enter password: ******
+-----------------+
| VERSION()       |
+-----------------+
| 5.1.5-alpha-log |
+-----------------+
+---------------------+
| NOW()               |
+---------------------+
| 2006-01-05 21:19:04 |
+---------------------+
```

4.2.3.2 Program Option Modifiers

Some options are “boolean” and control behavior that can be turned on or off. For example, the mysql client supports a --column-names option that determines whether or not to display a row of column names at the beginning of query results. By default, this option is enabled. However, you may want to disable it in some instances, such as when sending the output of `mysql` into another program that expects to see only data and not an initial header line.

To disable column names, you can specify the option using any of these forms:

```
--disable-column-names
--skip-column-names
--column-names=0
```

The `--disable` and `--skip` prefixes and the `=0` suffix all have the same effect: They turn the option off.

The “enabled” form of the option may be specified in any of these ways:

```
--column-names
--enable-column-names
--column-names=1
```

The values `ON`, `TRUE`, `OFF`, and `FALSE` are also recognized for boolean options (not case sensitive).

If an option is prefixed by `--loose`, a program does not exit with an error if it does not recognize the option, but instead issues only a warning:

```
shell> mysql --loose-no-such-option
mysql: WARNING: unknown option '--no-such-option'
```

The `--loose` prefix can be useful when you run programs from multiple installations of MySQL on the same machine and list options in an option file, An option that may not be recognized by all versions of a program can be given using the `--loose` prefix (or `loose` in an option file). Versions of the program that recognize the option process it normally, and versions that do not recognize it issue a warning and ignore it.

`mysqld` enables a limit to be placed on how large client programs can set dynamic system variables. To do this, use a `--maximum` prefix with the variable name. For example, `--maximum-query_cache_size=4M` prevents any client from making the query cache size larger than 4MB.

4.2.3.3 Using Option Files

Most MySQL programs can read startup options from option files (also sometimes called configuration files). Option files provide a convenient way to specify commonly used options so that they need not be entered on the command line each time you run a program. For the MySQL server, MySQL provides a number of preconfigured option files.

To determine whether a program reads option files, invoke it with the `--help` option. (For `mysqld`, use `--verbose` and `--help`.) If the program reads option files, the help message indicates which files it looks for and which option groups it recognizes.

The `.mylogin.cnf` file that contains login path options is created by the `mysql_config_editor` utility. See Section 4.6.6, “`mysql_config_editor` — MySQL Configuration Utility”. A “login path” is an option group that permits only a limited set of options: `host`, `user`, and `password`. Client programs specify which login path to read from `.mylogin.cnf` using the `--login-path` option.

To specify an alternate file name, set the `MYSQL_TEST_LOGIN_FILE` environment variable. This variable is used by the `mysql-test-run.pl` testing utility, but also is recognized by `mysql_config_editor` and by MySQL clients such as `mysql`, `mysqladmin`, and so forth.

On Windows, MySQL programs read startup options from the following files, in the specified order (top items are used first).

File Name	Purpose
`%PROGRAMDATA%\MySQL\MySQL Server 5.7\my.ini`, `%PROGRAMDATA%\MySQL\MySQL Server 5.7\my.cnf`	Global options
`%WINDIR%\my.ini`, `%WINDIR%\my.cnf`	Global options
`C:\my.ini`, `C:\my.cnf`	Global options
`INSTALLDIR``\my.ini`, *`INSTALLDIR`*`\my.cnf`	Global options
`defaults-extra-file`	The file specified with `--defaults-extra-file=`*`path`*, if any
`%APPDATA%\MySQL\.mylogin.cnf`	Login path options

`%PROGRAMDATA%` represents the file system directory that contains application data for all users on the host. This path defaults to `C:\ProgramData` on Microsoft Windows Vista and greater, and `C:\Documents and Settings\All Users\Application Data` on older versions of Microsoft Windows.

`%WINDIR%` represents the location of your Windows directory. This is commonly `C:\WINDOWS`. You can determine its exact location from the value of the `WINDIR` environment variable using the following command:

```
C:\> echo %WINDIR%
```

`INSTALLDIR` represents the MySQL installation directory. This is typically `C:\`*`PROGRAMDIR`*`\MySQL\MySQL 5.7 Server` where *`PROGRAMDIR`* represents the programs directory (usually `Program Files` on English-language versions of Windows), when MySQL 5.7 has been installed using the installation and configuration wizards. See Section 2.3.3, “Installing MySQL on Microsoft Windows Using MySQL Installer”.

`%APPDATA%` represents the value of the Windows application data directory. You can determine its exact location from the value of the `APPDATA` environment variable using the following command:

```
C:\> echo %APPDATA%
```

On Unix, Linux and Mac OS X, MySQL programs read startup options from the following files, in the specified order (top items are used first).

File Name	Purpose
`/etc/my.cnf`	Global options
`/etc/mysql/my.cnf`	Global options
`SYSCONFDIR``/my.cnf`	Global options
`$MYSQL_HOME/my.cnf`	Server-specific options
`defaults-extra-file`	The file specified with `--defaults-extra-file=`*`path`*, if any
`~/.my.cnf`	User-specific options

File Name	Purpose
`~/.mylogin.cnf`	Login path options

`~` represents the current user's home directory (the value of `$HOME`).

`SYSCONFDIR` represents the directory specified with the `SYSCONFDIR` option to `CMake` when MySQL was built. By default, this is the `etc` directory located under the compiled-in installation directory.

`MYSQL_HOME` is an environment variable containing the path to the directory in which the server-specific `my.cnf` file resides. If `MYSQL_HOME` is not set and you start the server using the `mysqld_safe` program, `mysqld_safe` attempts to set `MYSQL_HOME` as follows:

- Let *`BASEDIR`* and *`DATADIR`* represent the path names of the MySQL base directory and data directory, respectively.

- If there is a `my.cnf` file in *`DATADIR`* but not in *`BASEDIR`*, `mysqld_safe` sets `MYSQL_HOME` to *`DATADIR`*.

- Otherwise, if `MYSQL_HOME` is not set and there is no `my.cnf` file in *`DATADIR`*, `mysqld_safe` sets `MYSQL_HOME` to *`BASEDIR`*.

In MySQL 5.7, use of *`DATADIR`* as the location for `my.cnf` is deprecated.

Typically, *`DATADIR`* is `/usr/local/mysql/data` for a binary installation or `/usr/local/var` for a source installation. Note that this is the data directory location that was specified at configuration time, not the one specified with the `--datadir` option when `mysqld` starts. Use of `--datadir` at runtime has no effect on where the server looks for option files, because it looks for them before processing any options.

MySQL looks for option files in the order just described and reads any that exist. If an option file that you want to use does not exist, create it with a plain text editor.

If multiple instances of a given option are found, the last instance takes precedence. There is one exception: For `mysqld`, the *first* instance of the `--user` option is used as a security precaution, to prevent a user specified in an option file from being overridden on the command line.

Note

On Unix platforms, MySQL ignores configuration files that are world-writable. This is intentional as a security measure.

Any long option that may be given on the command line when running a MySQL program can be given in an option file as well. To get the list of available options for a program, run it with the `--help` option.

The syntax for specifying options in an option file is similar to command-line syntax (see Section 4.2.3.1, “Using Options on the Command Line”). However, in an option file, you omit the leading two dashes from the option name and you specify only one option per line. For example, `--quick` and `--host=localhost` on the command line should be specified as `quick` and `host=localhost` on separate lines in an option file. To specify an option of the form `--loose-`*`opt_name`* in an option file, write it as `loose-`*`opt_name`*.

Empty lines in option files are ignored. Nonempty lines can take any of the following forms:

- *`#comment`*, *`;comment`*

  Comment lines start with “`#`” or “`;`”. A “`#`” comment can start in the middle of a line as well.

- `[`*`group`*`]`

group is the name of the program or group for which you want to set options. After a group line, any option-setting lines apply to the named group until the end of the option file or another group line is given. Option group names are not case sensitive.

- *opt_name*

  This is equivalent to --*opt_name* on the command line.

- *opt_name*=*value*

  This is equivalent to --*opt_name*=*value* on the command line. In an option file, you can have spaces around the “=” character, something that is not true on the command line. You can optionally enclose the value within single quotation marks or double quotation marks, which is useful if the value contains a “#” comment character.

Leading and trailing spaces are automatically deleted from option names and values.

You can use the escape sequences “\b”, “\t”, “\n”, “\r”, “\\”, and “\s” in option values to represent the backspace, tab, newline, carriage return, backslash, and space characters. The escaping rules in option files are:

- If a backslash is followed by a valid escape sequence character, the sequence is converted to the character represented by the sequence. For example, “\s” is converted to a space.
- If a backslash is not followed by a valid escape sequence character, it remains unchanged. For example, “\S” is retained as is.

The preceding rules mean that a literal backslash can be given as “\\”, or as “\” if it is not followed by a valid escape sequence character.

The rules for escape sequences in option files differ slightly from the rules for escape sequences in string literals in SQL statements. In the latter context, if “*x*” is not a valid escape sequence character, “*x*” becomes “*x*” rather than “*x*”. See Section 9.1.1, “String Literals”.

The escaping rules for option file values are especially pertinent for Windows path names, which use “\” as a path name separator. A separator in a Windows path name must be written as “\\” if it is followed by an escape sequence character. It can be written as “\\” or “\” if it is not. Alternatively, “/” may be used in Windows path names and will be treated as “\”. Suppose that you want to specify a base directory of C:\Program Files\MySQL\MySQL Server 5.7 in an option file. This can be done several ways. Some examples:

```
basedir="C:\Program Files\MySQL\MySQL Server 5.7"
basedir="C:\\Program Files\\MySQL\\MySQL Server 5.7"
basedir="C:/Program Files/MySQL/MySQL Server 5.7"
basedir=C:\\Program\sFiles\\MySQL\\MySQL\sServer\s5.7
```

If an option group name is the same as a program name, options in the group apply specifically to that program. For example, the [mysqld] and [mysql] groups apply to the mysqld server and the mysql client program, respectively.

The [client] option group is read by all client programs (but *not* by mysqld). This enables you to specify options that apply to all clients. For example, [client] is the perfect group to use to specify the password that you use to connect to the server. (But make sure that the option file is readable and writable only by yourself, so that other people cannot find out your password.) Be sure not to put an option in the [client] group unless it is recognized by *all* client programs that you use. Programs that do not understand the option quit after displaying an error message if you try to run them.

Here is a typical global option file:

```
[client]
port=3306
socket=/tmp/mysql.sock

[mysqld]
port=3306
socket=/tmp/mysql.sock
key_buffer_size=16M
max_allowed_packet=8M

[mysqldump]
quick
```

The preceding option file uses *var_name=value* syntax for the lines that set the key_buffer_size and max_allowed_packet variables.

Here is a typical user option file:

```
[client]
# The following password will be sent to all standard MySQL clients
password="my_password"

[mysql]
no-auto-rehash
connect_timeout=2

[mysqlhotcopy]
interactive-timeout
```

If you want to create option groups that should be read by mysqld servers from a specific MySQL release series only, you can do this by using groups with names of [mysqld-5.6], [mysqld-5.7], and so forth. The following group indicates that the --new option should be used only by MySQL servers with 5.7.x version numbers:

```
[mysqld-5.7]
new
```

It is possible to use !include directives in option files to include other option files and !includedir to search specific directories for option files. For example, to include the /home/mydir/myopt.cnf file, use the following directive:

```
!include /home/mydir/myopt.cnf
```

To search the /home/mydir directory and read option files found there, use this directive:

```
!includedir /home/mydir
```

There is no guarantee about the order in which the option files in the directory will be read.

Note

Currently, any files to be found and included using the !includedir directive on Unix operating systems *must* have file names ending in .cnf. On Windows, this directive checks for files with the .ini or .cnf extension.

Write the contents of an included option file like any other option file. That is, it should contain groups of options, each preceded by a `[group]` line that indicates the program to which the options apply.

While an included file is being processed, only those options in groups that the current program is looking for are used. Other groups are ignored. Suppose that a `my.cnf` file contains this line:

```
!include /home/mydir/myopt.cnf
```

And suppose that `/home/mydir/myopt.cnf` looks like this:

```
[mysqladmin]
force

[mysqld]
key_buffer_size=16M
```

If `my.cnf` is processed by `mysqld`, only the `[mysqld]` group in `/home/mydir/myopt.cnf` is used. If the file is processed by `mysqladmin`, only the `[mysqladmin]` group is used. If the file is processed by any other program, no options in `/home/mydir/myopt.cnf` are used.

The `!includedir` directive is processed similarly except that all option files in the named directory are read.

4.2.3.4 Command-Line Options that Affect Option-File Handling

Most MySQL programs that support option files handle the following options. They affect option-file handling, so they must be given on the command line and not in an option file. To work properly, each of these options must be given before other options, with these exceptions:

- `--print-defaults` may be used immediately after `--defaults-file` or `--defaults-extra-file`.
- On Windows, if the server is started with the `--defaults-file` and `--install` options, `--install` must be first. See Section 2.3.5.7, “Starting MySQL as a Windows Service”.

When specifying file names, you should avoid the use of the “`~`” shell metacharacter because it might not be interpreted as you expect.

- `--defaults-extra-file=file_name`

  Read this option file after the global option file but (on Unix) before the user option file. If the file does not exist or is otherwise inaccessible, an error occurs. *`file_name`* is interpreted relative to the current directory if given as a relative path name rather than a full path name.

- `--defaults-file=file_name`

  Use only the given option file. If the file does not exist or is otherwise inaccessible, an error occurs. *`file_name`* is interpreted relative to the current directory if given as a relative path name rather than a full path name.

- `--defaults-group-suffix=str`

  Read not only the usual option groups, but also groups with the usual names and a suffix of *`str`*. For example, the `mysql` client normally reads the `[client]` and `[mysql]` groups. If the `--defaults-group-suffix=_other` option is given, `mysql` also reads the `[client_other]` and `[mysql_other]` groups.

- --login-path=*name*

  Read options from the named login path in the .mylogin.cnf login file. A “login path” is an option group that permits only a limited set of options: host, user, and password. Think of a login path as a set of values that indicate the server host and the credentials for authenticating with the server. To create the login file, use the mysql_config_editor utility. See Section 4.6.6, “mysql_config_editor — MySQL Configuration Utility”.

- --no-defaults

  Do not read any option files. If program startup fails due to reading unknown options from an option file, --no-defaults can be used to prevent them from being read.

  The exception is that the .mylogin.cnf file, if it exists, is read in all cases. This permits passwords to be specified in a safer way than on the command line even when --no-defaults is used. (.mylogin.cnf is created by the mysql_config_editor utility. See Section 4.6.6, “mysql_config_editor — MySQL Configuration Utility”.)

- --print-defaults

  Print the program name and all options that it gets from option files.

4.2.3.5 Using Options to Set Program Variables

Many MySQL programs have internal variables that can be set at runtime using the SET statement. See Section 13.7.4, “SET Syntax”, and Section 5.1.5, “Using System Variables”.

Most of these program variables also can be set at server startup by using the same syntax that applies to specifying program options. For example, mysql has a max_allowed_packet variable that controls the maximum size of its communication buffer. To set the max_allowed_packet variable for mysql to a value of 16MB, use either of the following commands:

```
shell> mysql --max_allowed_packet=16777216
shell> mysql --max_allowed_packet=16M
```

The first command specifies the value in bytes. The second specifies the value in megabytes. For variables that take a numeric value, the value can be given with a suffix of K, M, or G (either uppercase or lowercase) to indicate a multiplier of 1024, 1024^2 or 1024^3. (For example, when used to set max_allowed_packet, the suffixes indicate units of kilobytes, megabytes, or gigabytes.)

In an option file, variable settings are given without the leading dashes:

```
[mysql]
max_allowed_packet=16777216
```

Or:

```
[mysql]
max_allowed_packet=16M
```

If you like, underscores in a variable name can be specified as dashes. The following option groups are equivalent. Both set the size of the server's key buffer to 512MB:

```
[mysqld]
key_buffer_size=512M
```

```
[mysqld]
key-buffer-size=512M
```

A variable can be specified by writing it in full or as any unambiguous prefix. For example, the `max_allowed_packet` variable can be set for `mysql` as `--max_a`, but not as `--max` because the latter is ambiguous:

```
shell> mysql --max=1000000
mysql: ambiguous option '--max=1000000' (max_allowed_packet, max_join_size)
```

Be aware that the use of variable prefixes can cause problems in the event that new variables are implemented for a program. A prefix that is unambiguous now might become ambiguous in the future.

Suffixes for specifying a value multiplier can be used when setting a variable at server startup, but not to set the value with `SET` at runtime. On the other hand, with `SET` you can assign a variable's value using an expression, which is not true when you set a variable at server startup. For example, the first of the following lines is legal at server startup, but the second is not:

```
shell> mysql --max_allowed_packet=16M
shell> mysql --max_allowed_packet=16*1024*1024
```

Conversely, the second of the following lines is legal at runtime, but the first is not:

```
mysql> SET GLOBAL max_allowed_packet=16M;
mysql> SET GLOBAL max_allowed_packet=16*1024*1024;
```

4.2.3.6 Option Defaults, Options Expecting Values, and the = Sign

By convention, long forms of options that assign a value are written with an equals (`=`) sign, like this:

```
shell> mysql --host=tonfisk --user=jon
```

For options that require a value (that is, not having a default value), the equals sign is not required, and so the following is also valid:

```
shell> mysql --host tonfisk --user jon
```

In both cases, the `mysql` client attempts to connect to a MySQL server running on the host named “tonfisk” using an account with the user name “jon”.

Due to this behavior, problems can occasionally arise when no value is provided for an option that expects one. Consider the following example, where a user connects to a MySQL server running on host `tonfisk` as user `jon`:

```
shell> mysql --host 85.224.35.45 --user jon
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 3
Server version: 5.7.5 Source distribution

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> SELECT CURRENT_USER();
+----------------+
| CURRENT_USER() |
+----------------+
| jon@%          |
+----------------+
1 row in set (0.00 sec)
```

Omitting the required value for one of these option yields an error, such as the one shown here:

```
shell> mysql --host 85.224.35.45 --user
mysql: option '--user' requires an argument
```

In this case, `mysql` was unable to find a value following the `--user` option because nothing came after it on the command line. However, if you omit the value for an option that is *not* the last option to be used, you obtain a different error that you may not be expecting:

```
shell> mysql --host --user jon
ERROR 2005 (HY000): Unknown MySQL server host '--user' (1)
```

Because `mysql` assumes that any string following `--host` on the command line is a host name, `--host --user` is interpreted as `--host=--user`, and the client attempts to connect to a MySQL server running on a host named “--user”.

Options having default values always require an equals sign when assigning a value; failing to do so causes an error. For example, the MySQL server `--log-error` option has the default value *`host_name`*`.err`, where *`host_name`* is the name of the host on which MySQL is running. Assume that you are running MySQL on a computer whose host name is “tonfisk”, and consider the following invocation of `mysqld_safe`:

```
shell> mysqld_safe &
[1] 11699
shell> 080112 12:53:40 mysqld_safe Logging to '/usr/local/mysql/var/tonfisk.err'.
080112 12:53:40 mysqld_safe Starting mysqld daemon with databases from /usr/local/mysql/var
shell>
```

After shutting down the server, restart it as follows:

```
shell> mysqld_safe --log-error &
[1] 11699
shell> 080112 12:53:40 mysqld_safe Logging to '/usr/local/mysql/var/tonfisk.err'.
080112 12:53:40 mysqld_safe Starting mysqld daemon with databases from /usr/local/mysql/var
shell>
```

The result is the same, since `--log-error` is not followed by anything else on the command line, and it supplies its own default value. (The `&` character tells the operating system to run MySQL in the background; it is ignored by MySQL itself.) Now suppose that you wish to log errors to a file named `my-errors.err`. You might try starting the server with `--log-error my-errors`, but this does not have the intended effect, as shown here:

```
shell> mysqld_safe --log-error my-errors &
[1] 31357
shell> 080111 22:53:31 mysqld_safe Logging to '/usr/local/mysql/var/tonfisk.err'.
080111 22:53:32 mysqld_safe Starting mysqld daemon with databases from /usr/local/mysql/var
080111 22:53:34 mysqld_safe mysqld from pid file /usr/local/mysql/var/tonfisk.pid ended

[1]+  Done                    ./mysqld_safe --log-error my-errors
```

The server attempted to start using `/usr/local/mysql/var/tonfisk.err` as the error log, but then shut down. Examining the last few lines of this file shows the reason:

```
shell> tail /usr/local/mysql/var/tonfisk.err
2013-09-24T15:36:22.278034Z 0 [ERROR] Too many arguments (first extra is 'my-errors').
2013-09-24T15:36:22.278059Z 0 [Note] Use --verbose --help to get a list of available options!
2013-09-24T15:36:22.278076Z 0 [ERROR] Aborting
2013-09-24T15:36:22.279704Z 0 [Note] InnoDB: Starting shutdown...
2013-09-24T15:36:23.777471Z 0 [Note] InnoDB: Shutdown completed; log sequence number 2319086
2013-09-24T15:36:23.780134Z 0 [Note] mysqld: Shutdown complete
```

Because the `--log-error` option supplies a default value, you must use an equals sign to assign a different value to it, as shown here:

```
shell> mysqld_safe --log-error=my-errors &
[1] 31437
shell> 080111 22:54:15 mysqld_safe Logging to '/usr/local/mysql/var/my-errors.err'.
080111 22:54:15 mysqld_safe Starting mysqld daemon with databases from /usr/local/mysql/var

shell>
```

Now the server has been started successfully, and is logging errors to the file `/usr/local/mysql/var/my-errors.err`.

Similar issues can arise when specifying option values in option files. For example, consider a `my.cnf` file that contains the following:

```
[mysql]

host
user
```

When the `mysql` client reads this file, these entries are parsed as `--host --user` or `--host=--user`, with the result shown here:

```
shell> mysql
ERROR 2005 (HY000): Unknown MySQL server host '--user' (1)
```

However, in option files, an equals sign is not assumed. Suppose the `my.cnf` file is as shown here:

```
[mysql]

user jon
```

Trying to start `mysql` in this case causes a different error:

```
shell> mysql
mysql: unknown option '--user jon'
```

A similar error would occur if you were to write `host tonfisk` in the option file rather than `host=tonfisk`. Instead, you must use the equals sign:

```
[mysql]

user=jon
```

Now the login attempt succeeds:

```
shell> mysql
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 5
Server version: 5.7.5 Source distribution

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> SELECT USER();
+---------------+
| USER()        |
+---------------+
| jon@localhost |
+---------------+
1 row in set (0.00 sec)
```

This is not the same behavior as with the command line, where the equals sign is not required:

```
shell> mysql --user jon --host tonfisk
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 6
Server version: 5.7.5 Source distribution

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> SELECT USER();
+---------------+
| USER()        |
+---------------+
| jon@tonfisk   |
+---------------+
1 row in set (0.00 sec)
```

In MySQL 5.7, specifying an option requiring a value without a value in an option file causes the server to abort with an error. Suppose that `my.cnf` contains the following:

```
[mysqld]
log_error
relay_log
relay_log_index
```

This causes the server to fail on startup, as shown here:

```
shell> mysqld_safe &

130924 10:41:46 mysqld_safe Logging to '/home/jon/bin/mysql/var/tonfisk.err'.
130924 10:41:46 mysqld_safe Starting mysqld daemon with databases from /home/jon/bin/mysql/var
130924 10:41:47 mysqld_safe mysqld from pid file /home/jon/bin/mysql/var/tonfisk.pid ended
```

The `--log-error` option does not require an argument; however, the `--relay-log` option requires one, as shown in the error log (which in the absence of a specified value, defaults to *`datadir/hostname`*`.err`):

```
shell> tail -n 3 ../var/tonfisk.err

130924 10:41:46 mysqld_safe Starting mysqld daemon with databases from /home/jon/bin/mysql/var
2013-09-24T15:41:47.217180Z 0 [ERROR] /home/jon/bin/mysql/libexec/mysqld: option '--relay-log' requires an
2013-09-24T15:41:47.217479Z 0 [ERROR] Aborting
```

This is a change from previous behavior, where the server would have interpreted the last two lines in the example `my.cnf` file as `--relay-log=relay_log_index` and created a relay log file using “relay_log_index” as the basename. (Bug #25192)

4.2.4 Setting Environment Variables

Environment variables can be set at the command prompt to affect the current invocation of your command processor, or set permanently to affect future invocations. To set a variable permanently, you can set it in a startup file or by using the interface provided by your system for this purpose. Consult the documentation for your command interpreter for specific details. Section 2.11, "Environment Variables", lists all environment variables that affect MySQL program operation.

To specify a value for an environment variable, use the syntax appropriate for your command processor. For example, on Windows, you can set the USER variable to specify your MySQL account name. To do so, use this syntax:

```
SET USER=your_name
```

The syntax on Unix depends on your shell. Suppose that you want to specify the TCP/IP port number using the MYSQL_TCP_PORT variable. Typical syntax (such as for sh, ksh, bash, zsh, and so on) is as follows:

```
MYSQL_TCP_PORT=3306
export MYSQL_TCP_PORT
```

The first command sets the variable, and the export command exports the variable to the shell environment so that its value becomes accessible to MySQL and other processes.

For csh and tcsh, use setenv to make the shell variable available to the environment:

```
setenv MYSQL_TCP_PORT 3306
```

The commands to set environment variables can be executed at your command prompt to take effect immediately, but the settings persist only until you log out. To have the settings take effect each time you log in, use the interface provided by your system or place the appropriate command or commands in a startup file that your command interpreter reads each time it starts.

On Windows, you can set environment variables using the System Control Panel (under Advanced).

On Unix, typical shell startup files are .bashrc or .bash_profile for bash, or .tcshrc for tcsh.

Suppose that your MySQL programs are installed in /usr/local/mysql/bin and that you want to make it easy to invoke these programs. To do this, set the value of the PATH environment variable to include that directory. For example, if your shell is bash, add the following line to your .bashrc file:

```
PATH=${PATH}:/usr/local/mysql/bin
```

bash uses different startup files for login and nonlogin shells, so you might want to add the setting to .bashrc for login shells and to .bash_profile for nonlogin shells to make sure that PATH is set regardless.

If your shell is tcsh, add the following line to your .tcshrc file:

```
setenv PATH ${PATH}:/usr/local/mysql/bin
```

If the appropriate startup file does not exist in your home directory, create it with a text editor.

After modifying your PATH setting, open a new console window on Windows or log in again on Unix so that the setting goes into effect.

4.3 MySQL Server and Server-Startup Programs

This section describes mysqld, the MySQL server, and several programs that are used to start the server.

4.3.1 mysqld — The MySQL Server

mysqld, also known as MySQL Server, is the main program that does most of the work in a MySQL installation. MySQL Server manages access to the MySQL data directory that contains databases and tables. The data directory is also the default location for other information such as log files and status files.

When MySQL server starts, it listens for network connections from client programs and manages access to databases on behalf of those clients.

The mysqld program has many options that can be specified at startup. For a complete list of options, run this command:

```
shell> mysqld --verbose --help
```

MySQL Server also has a set of system variables that affect its operation as it runs. System variables can be set at server startup, and many of them can be changed at runtime to effect dynamic server reconfiguration. MySQL Server also has a set of status variables that provide information about its operation. You can monitor these status variables to access runtime performance characteristics.

For a full description of MySQL Server command options, system variables, and status variables, see Section 5.1, “The MySQL Server”. For information about installing MySQL and setting up the initial configuration, see Chapter 2, *Installing and Upgrading MySQL*.

4.3.2 mysqld_safe — MySQL Server Startup Script

mysqld_safe is the recommended way to start a mysqld server on Unix. mysqld_safe adds some safety features such as restarting the server when an error occurs and logging runtime information to an error log file. A description of error logging is given later in this section.

mysqld_safe tries to start an executable named mysqld. To override the default behavior and specify explicitly the name of the server you want to run, specify a --mysqld or --mysqld-version option to mysqld_safe. You can also use --ledir to indicate the directory where mysqld_safe should look for the server.

Many of the options to mysqld_safe are the same as the options to mysqld. See Section 5.1.3, “Server Command Options”.

Options unknown to mysqld_safe are passed to mysqld if they are specified on the command line, but ignored if they are specified in the [mysqld_safe] group of an option file. See Section 4.2.3.3, “Using Option Files”.

mysqld_safe reads all options from the [mysqld], [server], and [mysqld_safe] sections in option files. For example, if you specify a [mysqld] section like this, mysqld_safe will find and use the --log-error option:

```
[mysqld]
log-error=error.log
```

For backward compatibility, mysqld_safe also reads [safe_mysqld] sections, although you should rename such sections to [mysqld_safe] in MySQL 5.7 installations.

mysqld_safe supports the following options. It also reads option files and supports the options for processing them described at Section 4.2.3.4, “Command-Line Options that Affect Option-File Handling”.

Table 4.1 mysqld_safe Options

Format	Option File	Description
--basedir=path	basedir	The path to the MySQL installation directory
--core-file-size=size	core-file-size	The size of the core file that mysqld should be able to create
--datadir=path	datadir	The path to the data directory
--defaults-extra-file=path	defaults-extra-file	Read option file in addition to the usual option files
--defaults-file=file_name	defaults-file	Read only the given option file
--help		Display a help message and exit
--ledir=path	ledir	Use this option to indicate the path name to the directory where the server is located
--log-error=file_name	log-error	Write the error log to the given file
--malloc-lib=[lib-name]	malloc-lib	Alternative malloc library to use for mysqld
--mysqld=prog_name	mysqld	The name of the server program (in the ledir directory) that you want to start
--mysqld-version=suffix	mysqld-version	This option is similar to the --mysqld option, but you specify only the suffix for the server program name
--nice=priority	nice	Use the nice program to set the server's scheduling priority to the given value
--no-defaults	no-defaults	Do not read any option files
--open-files-limit=count	open-files-limit	The number of files that mysqld should be able to open
--pid-file=file_name	pid-file=file_name	The path name of the process ID file
--plugin-dir=path	plugin-dir=path	The directory where plugins are located
--port=number	port	The port number that the server should use when listening for TCP/IP connections
--skip-kill-mysqld	skip-kill-mysqld	Do not try to kill stray mysqld processes
--skip-syslog	skip-syslog	Do not write error messages to syslog; use error log file
--socket=path	socket	The Unix socket file that the server should use when listening for local connections
--syslog	syslog	Write error messages to syslog
--syslog-tag=tag	syslog-tag	Tag suffix for messages written to syslog
--timezone=timezone	timezone	Set the TZ time zone environment variable to the given option value
--user={user_name|user_id}	user	Run the mysqld server as the user having the name user_name or the numeric user ID user_id

- --help

Display a help message and exit.

- `--basedir=path`

  The path to the MySQL installation directory.

- `--core-file-size=size`

  The size of the core file that `mysqld` should be able to create. The option value is passed to `ulimit -c`.

- `--datadir=path`

  The path to the data directory.

- `--defaults-extra-file=path`

  The name of an option file to be read in addition to the usual option files. This must be the first option on the command line if it is used. If the file does not exist or is otherwise inaccessible, the server will exit with an error.

- `--defaults-file=file_name`

  The name of an option file to be read instead of the usual option files. This must be the first option on the command line if it is used.

- `--ledir=path`

  If `mysqld_safe` cannot find the server, use this option to indicate the path name to the directory where the server is located.

- `--log-error=file_name`

  Write the error log to the given file. See Section 5.2.2, “The Error Log”.

- `--malloc-lib=[lib_name]`

  The name of the library to use for memory allocation instead of the system `malloc()` library. Any library can be used by specifying its path name, but there is a shortcut form to enable use of the `tcmalloc` library that is shipped with binary MySQL distributions for Linux in MySQL 5.7. It is possible that the shortcut form will not work under certain configurations, in which case you should specify a path name instead.

  The `--malloc-lib` option works by modifying the `LD_PRELOAD` environment value to affect dynamic linking to enable the loader to find the memory-allocation library when `mysqld` runs:

  - If the option is not given, or is given without a value (`--malloc-lib=`), `LD_PRELOAD` is not modified and no attempt is made to use `tcmalloc`.

  - If the option is given as `--malloc-lib=tcmalloc`, `mysqld_safe` looks for a `tcmalloc` library in `/usr/lib` and then in the MySQL `pkglibdir` location (for example, `/usr/local/mysql/lib` or whatever is appropriate). If `tmalloc` is found, its path name is added to the beginning of the `LD_PRELOAD` value for `mysqld`. If `tcmalloc` is not found, `mysqld_safe` aborts with an error.

  - If the option is given as `--malloc-lib=/path/to/some/library`, that full path is added to the beginning of the `LD_PRELOAD` value. If the full path points to a nonexistent or unreadable file, `mysqld_safe` aborts with an error.

- For cases where mysqld_safe adds a path name to LD_PRELOAD, it adds the path to the beginning of any existing value the variable already has.

Linux users can use the libtcmalloc_minimal.so included in binary packages by adding these lines to the my.cnf file:

```
[mysqld_safe]
malloc-lib=tcmalloc
```

Those lines also suffice for users on any platform who have installed a tcmalloc package in /usr/lib. To use a specific tcmalloc library, specify its full path name. Example:

```
[mysqld_safe]
malloc-lib=/opt/lib/libtcmalloc_minimal.so
```

- --mysqld=*prog_name*

  The name of the server program (in the ledir directory) that you want to start. This option is needed if you use the MySQL binary distribution but have the data directory outside of the binary distribution. If mysqld_safe cannot find the server, use the --ledir option to indicate the path name to the directory where the server is located.

- --mysqld-version=*suffix*

  This option is similar to the --mysqld option, but you specify only the suffix for the server program name. The basename is assumed to be mysqld. For example, if you use --mysqld-version=debug, mysqld_safe starts the mysqld-debug program in the ledir directory. If the argument to --mysqld-version is empty, mysqld_safe uses mysqld in the ledir directory.

- --nice=*priority*

  Use the nice program to set the server's scheduling priority to the given value.

- --no-defaults

  Do not read any option files. This must be the first option on the command line if it is used.

- --open-files-limit=*count*

  The number of files that mysqld should be able to open. The option value is passed to ulimit -n. Note that you need to start mysqld_safe as root for this to work properly!

- --pid-file=*file_name*

  The path name of the process ID file.

  In MySQL 5.7.2 and later, mysqld_safe when starting up creates a PID file named mysqld_safe.pid in the MySQL data directory (Bug #16776528).

- --plugin-dir=*path*

  The path name of the plugin directory.

- --port=*port_num*

  The port number that the server should use when listening for TCP/IP connections. The port number must be 1024 or higher unless the server is started by the root system user.

- `--skip-kill-mysqld`

  Do not try to kill stray `mysqld` processes at startup. This option works only on Linux.

- `--socket=`*`path`*

  The Unix socket file that the server should use when listening for local connections.

- `--syslog`, `--skip-syslog`

  `--syslog` causes error messages to be sent to `syslog` on systems that support the `logger` program. `--skip-syslog` suppresses the use of `syslog`; messages are written to an error log file.

  When `syslog` is used, the `daemon.err` syslog priority/facility is used for all log messages.

- `--syslog-tag=`*`tag`*

  For logging to `syslog`, messages from `mysqld_safe` and `mysqld` are written with a tag of `mysqld_safe` and `mysqld`, respectively. To specify a suffix for the tag, use `--syslog-tag=`*`tag`*, which modifies the tags to be `mysqld_safe-`*`tag`* and `mysqld-`*`tag`*.

- `--timezone=`*`timezone`*

  Set the `TZ` time zone environment variable to the given option value. Consult your operating system documentation for legal time zone specification formats.

- `--user={`*`user_name`*`|`*`user_id`*`}`

  Run the `mysqld` server as the user having the name *`user_name`* or the numeric user ID *`user_id`*. (“User” in this context refers to a system login account, not a MySQL user listed in the grant tables.)

If you execute `mysqld_safe` with the `--defaults-file` or `--defaults-extra-file` option to name an option file, the option must be the first one given on the command line or the option file will not be used. For example, this command will not use the named option file:

```
mysql> mysqld_safe --port=port_num --defaults-file=file_name
```

Instead, use the following command:

```
mysql> mysqld_safe --defaults-file=file_name --port=port_num
```

The `mysqld_safe` script is written so that it normally can start a server that was installed from either a source or a binary distribution of MySQL, even though these types of distributions typically install the server in slightly different locations. (See Section 2.1.5, “Installation Layouts”.) `mysqld_safe` expects one of the following conditions to be true:

- The server and databases can be found relative to the working directory (the directory from which `mysqld_safe` is invoked). For binary distributions, `mysqld_safe` looks under its working directory for `bin` and `data` directories. For source distributions, it looks for `libexec` and `var` directories. This condition should be met if you execute `mysqld_safe` from your MySQL installation directory (for example, `/usr/local/mysql` for a binary distribution).

- If the server and databases cannot be found relative to the working directory, `mysqld_safe` attempts to locate them by absolute path names. Typical locations are `/usr/local/libexec` and `/usr/local/var`. The actual locations are determined from the values configured into the distribution at the time it was built. They should be correct if MySQL is installed in the location specified at configuration time.

Because `mysqld_safe` tries to find the server and databases relative to its own working directory, you can install a binary distribution of MySQL anywhere, as long as you run `mysqld_safe` from the MySQL installation directory:

```
shell> cd mysql_installation_directory
shell> bin/mysqld_safe &
```

If `mysqld_safe` fails, even when invoked from the MySQL installation directory, you can specify the `--ledir` and `--datadir` options to indicate the directories in which the server and databases are located on your system.

In MySQL 5.7, `mysqld_safe` tries to use the `sleep` and `date` system utilities to determine how many times it has attempted to start this second, and—if these are present and this is greater than 5 times—is forced to wait 1 full second before starting again. This is intended to prevent excessive CPU usage in the event of repeated failures. (Bug #11761530, Bug #54035)

When you use `mysqld_safe` to start `mysqld`, `mysqld_safe` arranges for error (and notice) messages from itself and from `mysqld` to go to the same destination.

There are several `mysqld_safe` options for controlling the destination of these messages:

- `--syslog`: Write error messages to `syslog` on systems that support the `logger` program.
- `--skip-syslog`: Do not write error messages to `syslog`. Messages are written to the default error log file (*`host_name`*`.err` in the data directory), or to a named file if the `--log-error` option is given.
- `--log-error=`*`file_name`*: Write error messages to the named error file.

If none of these options is given, the default is `--skip-syslog`.

If `--syslog` and `--log-error` are both given, a warning is issued and `--log-error` takes precedence.

When `mysqld_safe` writes a message, notices go to the logging destination (`syslog` or the error log file) and `stdout`. Errors go to the logging destination and `stderr`.

Normally, you should not edit the `mysqld_safe` script. Instead, configure `mysqld_safe` by using command-line options or options in the `[mysqld_safe]` section of a `my.cnf` option file. In rare cases, it might be necessary to edit `mysqld_safe` to get it to start the server properly. However, if you do this, your modified version of `mysqld_safe` might be overwritten if you upgrade MySQL in the future, so you should make a copy of your edited version that you can reinstall.

4.3.3 `mysql.server` — MySQL Server Startup Script

MySQL distributions on Unix include a script named `mysql.server`. It can be used on systems such as Linux and Solaris that use System V-style run directories to start and stop system services. It is also used by the Mac OS X Startup Item for MySQL.

`mysql.server` can be found in the `support-files` directory under your MySQL installation directory or in a MySQL source distribution.

If you use the Linux server RPM package (`MySQL-server-`*`VERSION`*`.rpm`), the `mysql.server` script will be installed in the `/etc/init.d` directory with the name `mysql`. You need not install it manually. See Section 2.5.3, “Installing MySQL on Linux Using RPM Packages”, for more information on the Linux RPM packages.

Some vendors provide RPM packages that install a startup script under a different name such as mysqld.

If you install MySQL from a source distribution or using a binary distribution format that does not install mysql.server automatically, you can install it manually. Instructions are provided in Section 2.9.1.2, “Starting and Stopping MySQL Automatically”.

mysql.server reads options from the [mysql.server] and [mysqld] sections of option files. For backward compatibility, it also reads [mysql_server] sections, although you should rename such sections to [mysql.server] when using MySQL 5.7.

mysql.server supports the following options.

- --basedir=*path*

  The path to the MySQL installation directory.

- --datadir=*path*

  The path to the MySQL data directory.

- --pid-file=*file_name*

  The path name of the file in which the server should write its process ID.

- --service-startup-timeout=*file_name*

  How long in seconds to wait for confirmation of server startup. If the server does not start within this time, mysql.server exits with an error. The default value is 900. A value of 0 means not to wait at all for startup. Negative values mean to wait forever (no timeout).

- --use-mysqld_safe

  Use mysqld_safe to start the server. This is the default.

- --user=*user_name*

  The login user name to use for running mysqld.

4.3.4 mysqld_multi — Manage Multiple MySQL Servers

mysqld_multi is designed to manage several mysqld processes that listen for connections on different Unix socket files and TCP/IP ports. It can start or stop servers, or report their current status.

mysqld_multi searches for groups named [mysqld*N*] in my.cnf (or in the file named by the --defaults-file option). *N* can be any positive integer. This number is referred to in the following discussion as the option group number, or *GNR*. Group numbers distinguish option groups from one another and are used as arguments to mysqld_multi to specify which servers you want to start, stop, or obtain a status report for. Options listed in these groups are the same that you would use in the [mysqld] group used for starting mysqld. (See, for example, Section 2.9.1.2, “Starting and Stopping MySQL Automatically”.) However, when using multiple servers, it is necessary that each one use its own value for options such as the Unix socket file and TCP/IP port number. For more information on which options must be unique per server in a multiple-server environment, see Section 5.3, “Running Multiple MySQL Instances on One Machine”.

To invoke mysqld_multi, use the following syntax:

```
shell> mysqld_multi [options] {start|stop|reload|report} [GNR[,GNR] ...]
```

start, stop, reload (stop and restart), and report indicate which operation to perform. You can perform the designated operation for a single server or multiple servers, depending on the *GNR* list that follows the option name. If there is no list, mysqld_multi performs the operation for all servers in the option file.

Each *GNR* value represents an option group number or range of group numbers. The value should be the number at the end of the group name in the option file. For example, the *GNR* for a group named [mysqld17] is 17. To specify a range of numbers, separate the first and last numbers by a dash. The *GNR* value 10-13 represents groups [mysqld10] through [mysqld13]. Multiple groups or group ranges can be specified on the command line, separated by commas. There must be no whitespace characters (spaces or tabs) in the *GNR* list; anything after a whitespace character is ignored.

This command starts a single server using option group [mysqld17]:

```
shell> mysqld_multi start 17
```

This command stops several servers, using option groups [mysqld8] and [mysqld10] through [mysqld13]:

```
shell> mysqld_multi stop 8,10-13
```

For an example of how you might set up an option file, use this command:

```
shell> mysqld_multi --example
```

mysqld_multi searches for option files as follows:

- With --no-defaults, no option files are read.
- With --defaults-file=*file_name*, only the named file is read.
- Otherwise, option files in the standard list of locations are read, including any file named by the --defaults-extra-file=*file_name* option, if one is given. (If the option is given multiple times, the last value is used.)

Option files read are searched for [mysqld_multi] and [mysqld*N*] option groups. The [mysqld_multi] group can be used for options to mysqld_multi itself. [mysqld*N*] groups can be used for options passed to specific mysqld instances.

The [mysqld] or [mysqld_safe] groups can be used for common options read by all instances of mysqld or mysqld_safe. You can specify a --defaults-file=*file_name* option to use a different configuration file for that instance, in which case the [mysqld] or [mysqld_safe] groups from that file will be used for that instance.

mysqld_multi supports the following options.

- --help

  Display a help message and exit.

- --example

  Display a sample option file.

- --log=*file_name*

Specify the name of the log file. If the file exists, log output is appended to it.

- `--mysqladmin=prog_name`

  The `mysqladmin` binary to be used to stop servers.

- `--mysqld=prog_name`

  The `mysqld` binary to be used. Note that you can specify `mysqld_safe` as the value for this option also. If you use `mysqld_safe` to start the server, you can include the `mysqld` or `ledir` options in the corresponding `[mysqldN]` option group. These options indicate the name of the server that `mysqld_safe` should start and the path name of the directory where the server is located. (See the descriptions for these options in Section 4.3.2, "`mysqld_safe` — MySQL Server Startup Script".) Example:

```
[mysqld38]
mysqld = mysqld-debug
ledir  = /opt/local/mysql/libexec
```

- `--no-log`

  Print log information to `stdout` rather than to the log file. By default, output goes to the log file.

- `--password=password`

  The password of the MySQL account to use when invoking `mysqladmin`. Note that the password value is not optional for this option, unlike for other MySQL programs.

- `--silent`

  Silent mode; disable warnings.

- `--tcp-ip`

  Connect to each MySQL server through the TCP/IP port instead of the Unix socket file. (If a socket file is missing, the server might still be running, but accessible only through the TCP/IP port.) By default, connections are made using the Unix socket file. This option affects `stop` and `report` operations.

- `--user=user_name`

  The user name of the MySQL account to use when invoking `mysqladmin`.

- `--verbose`

  Be more verbose.

- `--version`

  Display version information and exit.

Some notes about `mysqld_multi`:

- **Most important**: Before using `mysqld_multi` be sure that you understand the meanings of the options that are passed to the `mysqld` servers and *why* you would want to have separate `mysqld` processes. Beware of the dangers of using multiple `mysqld` servers with the same data directory. Use separate data directories, unless you *know* what you are doing. Starting multiple servers with the same data directory does *not* give you extra performance in a threaded system. See Section 5.3, "Running Multiple MySQL Instances on One Machine".

- ⚠ **Important**

  Make sure that the data directory for each server is fully accessible to the Unix account that the specific mysqld process is started as. *Do not* use the Unix *root* account for this, unless you *know* what you are doing. See Section 6.1.5, “How to Run MySQL as a Normal User”.

- Make sure that the MySQL account used for stopping the mysqld servers (with the mysqladmin program) has the same user name and password for each server. Also, make sure that the account has the SHUTDOWN privilege. If the servers that you want to manage have different user names or passwords for the administrative accounts, you might want to create an account on each server that has the same user name and password. For example, you might set up a common multi_admin account by executing the following commands for each server:

  ```
  shell> mysql -u root -S /tmp/mysql.sock -p
  Enter password:
  mysql> GRANT SHUTDOWN ON *.*
      -> TO 'multi_admin'@'localhost' IDENTIFIED BY 'multipass';
  ```

  See Section 6.2, “The MySQL Access Privilege System”. You have to do this for each mysqld server. Change the connection parameters appropriately when connecting to each one. Note that the host name part of the account name must permit you to connect as multi_admin from the host where you want to run mysqld_multi.

- The Unix socket file and the TCP/IP port number must be different for every mysqld. (Alternatively, if the host has multiple network addresses, you can use --bind-address to cause different servers to listen to different interfaces.)

- The --pid-file option is very important if you are using mysqld_safe to start mysqld (for example, --mysqld=mysqld_safe) Every mysqld should have its own process ID file. The advantage of using mysqld_safe instead of mysqld is that mysqld_safe monitors its mysqld process and restarts it if the process terminates due to a signal sent using kill -9 or for other reasons, such as a segmentation fault. Please note that the mysqld_safe script might require that you start it from a certain place. This means that you might have to change location to a certain directory before running mysqld_multi. If you have problems starting, please see the mysqld_safe script. Check especially the lines:

  ```
  -----------------------------------------------------------------
  MY_PWD=`pwd`
  # Check if we are starting this relative (for the binary release)
  if test -d $MY_PWD/data/mysql -a \
     -f ./share/mysql/english/errmsg.sys -a \
     -x ./bin/mysqld
  -----------------------------------------------------------------
  ```

  The test performed by these lines should be successful, or you might encounter problems. See Section 4.3.2, “mysqld_safe — MySQL Server Startup Script”.

- You might want to use the --user option for mysqld, but to do this you need to run the mysqld_multi script as the Unix superuser (root). Having the option in the option file doesn't matter; you just get a warning if you are not the superuser and the mysqld processes are started under your own Unix account.

The following example shows how you might set up an option file for use with mysqld_multi. The order in which the mysqld programs are started or stopped depends on the order in which they appear in the option file. Group numbers need not form an unbroken sequence. The first and fifth [mysqld*N*] groups were intentionally omitted from the example to illustrate that you can have “gaps” in the option file. This gives you more flexibility.

```
# This file should probably be in your home dir (~/.my.cnf)
# or /etc/my.cnf
# Version 2.1 by Jani Tolonen

[mysqld_multi]
mysqld     = /usr/local/bin/mysqld_safe
mysqladmin = /usr/local/bin/mysqladmin
user       = multi_admin
password   = multipass

[mysqld2]
socket     = /tmp/mysql.sock2
port       = 3307
pid-file   = /usr/local/mysql/var2/hostname.pid2
datadir    = /usr/local/mysql/var2
language   = /usr/local/share/mysql/english
user       = john

[mysqld3]
socket     = /tmp/mysql.sock3
port       = 3308
pid-file   = /usr/local/mysql/var3/hostname.pid3
datadir    = /usr/local/mysql/var3
language   = /usr/local/share/mysql/swedish
user       = monty

[mysqld4]
socket     = /tmp/mysql.sock4
port       = 3309
pid-file   = /usr/local/mysql/var4/hostname.pid4
datadir    = /usr/local/mysql/var4
language   = /usr/local/share/mysql/estonia
user       = tonu

[mysqld6]
socket     = /tmp/mysql.sock6
port       = 3311
pid-file   = /usr/local/mysql/var6/hostname.pid6
datadir    = /usr/local/mysql/var6
language   = /usr/local/share/mysql/japanese
user       = jani
```

See Section 4.2.3.3, “Using Option Files”.

4.4 MySQL Installation-Related Programs

The programs in this section are used when installing or upgrading MySQL.

4.4.1 `comp_err` — Compile MySQL Error Message File

`comp_err` creates the `errmsg.sys` file that is used by `mysqld` to determine the error messages to display for different error codes. `comp_err` normally is run automatically when MySQL is built. It compiles the `errmsg.sys` file from the plaintext file located at `sql/share/errmsg.txt` in MySQL source distributions.

`comp_err` also generates `mysqld_error.h`, `mysqld_ername.h`, and `sql_state.h` header files.

For more information about how error messages are defined, see the MySQL Internals Manual.

Invoke `comp_err` like this:

```
shell> comp_err [options]
```

comp_err supports the following options.

- --help, -?

  Display a help message and exit.

- --charset=*path*, -C *path*

  The character set directory. The default is ../sql/share/charsets.

- --debug=*debug_options*, -# *debug_options*

  Write a debugging log. A typical *debug_options* string is d:t:O,*file_name*. The default is d:t:O,/tmp/comp_err.trace.

- --debug-info, -T

  Print some debugging information when the program exits.

- --header_file=*file_name*, -H *file_name*

  The name of the error header file. The default is mysqld_error.h.

- --in_file=*file_name*, -F *file_name*

  The name of the input file. The default is ../sql/share/errmsg.txt.

- --name_file=*file_name*, -N *file_name*

  The name of the error name file. The default is mysqld_ername.h.

- --out_dir=*path*, -D *path*

  The name of the output base directory. The default is ../sql/share/.

- --out_file=*file_name*, -O *file_name*

  The name of the output file. The default is errmsg.sys.

- --statefile=*file_name*, -S *file_name*

  The name for the SQLSTATE header file. The default is sql_state.h.

- --version, -V

  Display version information and exit.

4.4.2 mysqlbug — Generate Bug Report

This program is obsolete.

The normal way to report bugs is to visit http://bugs.mysql.com/, which is the address for our bugs database. This database is public and can be browsed and searched by anyone. If you log in to the system, you can enter new reports.

4.4.3 mysql_install_db — Initialize MySQL Data Directory

mysql_install_db initializes the MySQL data directory and creates the system tables that it contains, if they do not exist. It also initializes the system tablespace and related data structures needed to manage InnoDB tables. mysql_install_db is a Perl script and can be used on any system with Perl installed.

As of MySQL 5.7.4, MySQL deployments installed using RPM packages are secure by default and have these characteristics:

- The installation process creates a single root account, 'root'@'localhost', automatically generates a random password for this account, and marks the password expired.
- The initial random root password is written to the .mysql_secret file in the home directory of the effective user running the script. .mysql_secret is created with mode 600 to be accessible only to the system user for whom it is created.

  If .mysql_secret already exists, the new password information is appended to it. Each password entry includes a timestamp so that in the event of multiple install operations it is possible to determine the password associated with each one.
- No anonymous-user MySQL accounts are created.
- No test database is created.

As a result of these actions, it is necessary after installation to start the server, connect as root using the password written to the .mysql_secret file, and select a new root password. Until this is done, root cannot do anything else. To change the password, you can use the SET PASSWORD statement (for example, with the mysql client). You can also use mysqladmin or mysql_secure_installation.

For information about overriding some of the characteristics just described, see the description of the --skip-random-passwords option.

On Unix platforms, mysql_install_db creates a default option file named my.cnf in the base installation directory. This file is created from a template included in the distribution package named my-default.cnf. You can find the template in or under the base installation directory. When started using mysqld_safe, the server uses my.cnf file by default. If my.cnf already exists, mysql_install_db assumes it to be in use and writes a new file named my-new.cnf instead.

With one exception, the settings in the default option file are commented and have no effect. The exception is that the file changes the sql_mode system variable from its default of NO_ENGINE_SUBSTITUTION to also include STRICT_TRANS_TABLES. This setting produces a server configuration that results in errors rather than warnings for bad data in operations that modify transactional tables. See Section 5.1.7, “Server SQL Modes”.

As of MySQL 5.7.2, when mysql_install_db invokes the server to initialize the mysql database, the server assigns every user table row a nonempty plugin column value. The value is 'mysql_native_password' unless the default_authentication_plugin system variable is set otherwise.

To invoke mysql_install_db, use the following syntax:

```
shell> mysql_install_db [options]
```

Because the MySQL server, mysqld, must access the data directory when it runs later, you should either run mysql_install_db from the same system account that will be used for running mysqld or run it as root and use the --user option to indicate the user name that mysqld will run as. It might be necessary to specify other options such as --basedir or --datadir if mysql_install_db does not use the correct locations for the installation directory or data directory. For example:

```
shell> scripts/mysql_install_db --user=mysql \
         --basedir=/opt/mysql/mysql \
         --datadir=/opt/mysql/mysql/data
```

Note

After mysql_install_db sets up the InnoDB system tablespace, changes to some tablespace characteristics require setting up a whole new instance. This includes the file name of the first file in the system tablespace and the number of undo logs. If you do not want to use the default values, make sure that the settings for the innodb_data_file_path and innodb_log_file_size configuration parameters are in place in the MySQL configuration file before running mysql_install_db. Also make sure to specify as necessary other parameters that affect the creation and location of InnoDB files, such as innodb_data_home_dir and innodb_log_group_home_dir.

If those options are in your configuration file but that file is not in a location that MySQL reads by default, specify the file location using the --defaults-extra-file option when you run mysql_install_db.

Note

If you have set a custom TMPDIR environment variable when performing the installation, and the specified directory is not accessible, mysql_install_db may fail. If so, unset TMPDIR or set TMPDIR to point to the system temporary directory (usually /tmp).

mysql_install_db supports the following options, which can be specified on the command line or in the [mysql_install_db] group of an option file. (Options that are common to mysqld can also be specified in the [mysqld] group.) Other options are passed to mysqld. For information about option files, see Section 4.2.3.3, “Using Option Files”.

Table 4.2 mysql_install_db Options

Format	Option File	Description	Introduced	Removed
--basedir=path	basedir	The MySQL base directory		
--builddir=path	builddir	The build directory (for out-of-source builds)		
--cross-bootstrap	cross-bootstrap	For internal use		
--datadir=path	datadir	The MySQL data directory		
--defaults-extra-file=file_name		Read option file in addition to the usual option files		
--defaults-file=file_name		Read only the given option file		
--force	force	Run even if DNS does not work		
--help	help	Display help message and exit		
--ldata=path	ldata	Synonym for --datadir		
--no-defaults		Do not read any option files		
--random-passwords	random-passwords	Generate random root password		5.7.4

Format	Option File	Description	Introduced	Removed
--rpm	rpm	For internal use		
--skip-name-resolve	skip-name-resolve	Use IP addresses rather than host names in grant tables		
--skip-random-passwords	skip-random-passwords	Do not generate random root password	5.7.4	
--srcdir=path	srcdir	For internal use		
--user=user_name	user	System login user under which to execute		
--verbose	verbose	Verbose mode		
--windows	windows	For internal use		

- `--help`

  Display a help message and exit.

- `--basedir=path`

  The path to the MySQL installation directory.

- `--builddir=path`

  For use with `--srcdir` and out-of-source builds. Set this to the location of the directory where the built files reside.

- `--cross-bootstrap`

  For internal use. This option is used for building system tables on one host intended for another.

- `--datadir=path`, `--ldata=path`

  The path to the MySQL data directory. Only the last component of the path name is created if it does not exist; the parent directory must already exist or an error occurs.

- `--defaults-extra-file=file_name`

  Read this option file after the global option file but (on Unix) before the user option file. If the file does not exist or is otherwise inaccessible, an error occurs. *file_name* is interpreted relative to the current directory if given as a relative path name rather than a full path name.

- `--defaults-file=file_name`

  Use only the given option file. If the file does not exist or is otherwise inaccessible, an error occurs. *file_name* is interpreted relative to the current directory if given as a relative path name rather than a full path name.

- `--force`

  Cause `mysql_install_db` to run even if DNS does not work. Grant table entries normally created using host names will use IP addresses instead.

- `--no-defaults`

  Do not read any option files. If program startup fails due to reading unknown options from an option file, `--no-defaults` can be used to prevent them from being read.

- --random-passwords

  Note

  This option was removed in MySQL 5.7.4 and replaced with --skip-random-passwords.

  On Unix platforms, this option provides for more secure MySQL installation. Invoking mysql_install_db with --random-passwords causes it to perform the following actions in addition to its normal operation:

  - The installation process creates a random password, assigns it to the initial MySQL root accounts, and sets the "password expired" flag for those accounts.
  - The initial random root password is written to the .mysql_secret file in the directory named by the HOME environment variable. Depending on operating system, using a command such as sudo may cause the value of HOME to refer to the home directory of the root system user. .mysql_secret is created with mode 600 to be accessible only to the system user for whom it is created.

    If .mysql_secret already exists, the new password information is appended to it. Each password entry includes a timestamp so that in the event of multiple install operations it is possible to determine the password associated with each one.
  - No anonymous-user MySQL accounts are created.

  As a result of these actions, it is necessary after installation to start the server, connect as root using the password written to the .mysql_secret file, and select a new root password. Until this is done, root cannot do anything else. This must be done for each root account you intend to use. To change the password, you can use the SET PASSWORD statement (for example, with the mysql client). You can also use mysqladmin or mysql_secure_installation.

  New install operations (not upgrades) using RPM packages and Solaris PKG packages invoke mysql_install_db with the --random-passwords option. (Install operations using RPMs for Unbreakable Linux Network are unaffected because they do not use mysql_install_db.)

  For install operations using a binary .tar.gz distribution or a source distribution, you can invoke mysql_install_db with the --random-passwords option manually to make your MySQL installation more secure. This is recommended, particularly for sites with sensitive data.

- --rpm

  For internal use. This option is used during the MySQL installation process for install operations performed using RPM packages.

- --skip-name-resolve

  Use IP addresses rather than host names when creating grant table entries. This option can be useful if your DNS does not work.

- --skip-random-passwords

  As of MySQL 5.7.4, MySQL deployments produced using mysql_install_db are secure by default. When invoked *without* the --skip-random-passwords option, mysql_install_db uses these default deployment characteristics:

  - The installation process creates a single root account, 'root'@'localhost', automatically generates a random password for this account, and marks the password expired.

- The initial random root password is written to the .mysql_secret file in the home directory of the effective user running the script. .mysql_secret is created with mode 600 to be accessible only to the system user for whom it is created.

  If .mysql_secret already exists, the new password information is appended to it. Each password entry includes a timestamp so that in the event of multiple install operations it is possible to determine the password associated with each one.

- No anonymous-user MySQL accounts are created.

- No test database is created.

As a result of these actions, it is necessary after installation to start the server, connect as root using the password written to the .mysql_secret file, and select a new root password. Until this is done, root cannot do anything else. To change the password, you can use the SET PASSWORD statement (for example, with the mysql client). You can also use mysqladmin or mysql_secure_installation.

To produce a MySQL deployment that is not secure by default, you must explicitly specify the --skip-random-passwords option when you invoke mysql_install_db. With this option, mysql_install_db performs the following actions:

- Installation creates a single root account, 'root'@'localhost', that has no password.

- A test database is created that is accessible by any user.

Note

As of MySQL 5.7.4, mysql_install_db no longer creates anonymous-user accounts, even with --skip-random-passwords.

The --skip-random-passwords option was added in MySQL 5.7.4. It replaces the --random-passwords option.

- --srcdir=*path*

  For internal use. This option specifies the directory under which mysql_install_db looks for support files such as the error message file and the file for populating the help tables.

- --user=*user_name*

  The system (login) user name to use for running mysqld. Files and directories created by mysqld will be owned by this user. You must be root to use this option. By default, mysqld runs using your current login name and files and directories that it creates will be owned by you.

- --verbose

  Verbose mode. Print more information about what the program does.

- --windows

  For internal use. This option is used for creating Windows distributions. This is a deprecated alias for --cross-bootstrap

4.4.4 mysql_plugin — Configure MySQL Server Plugins

The mysql_plugin utility enables MySQL administrators to manage which plugins a MySQL server loads. It provides an alternative to manually specifying the --plugin-load option at server startup or using the INSTALL PLUGIN and UNINSTALL PLUGIN statements at runtime.

Depending on whether mysql_plugin is invoked to enable or disable plugins, it inserts or deletes rows in the mysql.plugin table that serves as a plugin registry. (To perform this operation, mysql_plugin invokes the MySQL server in bootstrap mode. This means that the server must not already be running.) For normal server startups, the server loads and enables plugins listed in mysql.plugin automatically. For additional control over plugin activation, use --*plugin_name* options named for specific plugins, as described in Section 5.1.8.1, "Installing and Uninstalling Plugins".

Each invocation of mysql_plugin reads a configuration file to determine how to configure the plugins contained in a single plugin library object file. To invoke mysql_plugin, use this syntax:

```
mysql_plugin [options] plugin {ENABLE|DISABLE}
```

plugin is the name of the plugin to configure. ENABLE or DISABLE (not case sensitive) specify whether to enable or disable components of the plugin library named in the configuration file. The order of the *plugin* and ENABLE or DISABLE arguments does not matter.

For example, to configure components of a plugin library file named myplugins.so on Linux or myplugins.dll on Windows, specify a *plugin* value of myplugins. Suppose that this plugin library contains three plugins, plugin1, plugin2, and plugin3, all of which should be configured under mysql_plugin control. By convention, configuration files have a suffix of .ini and the same basename as the plugin library, so the default configuration file name for this plugin library is myplugins.ini. The configuration file contents look like this:

```
myplugins
plugin1
plugin2
plugin3
```

The first line in the myplugins.ini file is the name of the library object file, without any extension such as .so or .dll. The remaining lines are the names of the components to be enabled or disabled. Each value in the file should be on a separate line. Lines on which the first character is '#' are taken as comments and ignored.

To enable the plugins listed in the configuration file, invoke mysql_plugin this way:

```
shell> mysql_plugin myplugins ENABLE
```

To disable the plugins, use DISABLE rather than ENABLE.

An error occurs if mysql_plugin cannot find the configuration file or plugin library file, or if mysql_plugin cannot start the MySQL server.

mysql_plugin supports the following options, which can be specified on the command line or in the [mysqld] group of any option file. For options specified in a [mysqld] group, mysql_plugin recognizes the --basedir, --datadir, and --plugin-dir options and ignores others. For information about option files, see Section 4.2.3.3, "Using Option Files".

Table 4.3 mysql_plugin Options

Format	Option File	Description
--basedir=path	basedir=path	The server base directory
--datadir=path	datadir=path	The server data directory

Format	Option File	Description
--help		Display help message and exit
--my-print-defaults=path	my-print-defaults=path	The path to my_print_defaults
--mysqld=path	mysqld=path	The path to the server
--no-defaults	no-defaults	Do not read configuration file
--plugin-dir=path	plugin-dir=path	The directory where plugins are located
--plugin-ini=file_name	plugin-ini=file_name	The plugin configuration file
--print-defaults	print-defaults	Show configuration file defaults
--verbose		Verbose mode
--version		Display version information and exit

- `--help`, `-?`

  Display a help message and exit.

- `--basedir=path`, `-b path`

  The server base directory.

- `--datadir=path`, `-d path`

  The server data directory.

- `--my-print-defaults=path`, `-b path`

  The path to the `my_print_defaults` program.

- `--mysqld=path`, `-b path`

  The path to the `mysqld` server.

- `--no-defaults`, `-p`

  Do not read values from the configuration file. This option enables an administrator to skip reading defaults from the configuration file.

  With `mysql_plugin`, this option need not be given first on the command line, unlike most other MySQL programs that support `--no-defaults`.

- `--plugin-dir=path`, `-p path`

  The server plugin directory.

- `--plugin-ini=file_name`, `-i file_name`

  The `mysql_plugin` configuration file. Relative path names are interpreted relative to the current directory. If this option is not given, the default is `plugin.ini` in the plugin directory, where `plugin` is the `plugin` argument on the command line.

- `--print-defaults`, `-P`

Display the default values from the configuration file. This option causes mysql_plugin to print the defaults for --basedir, --datadir, and --plugin-dir if they are found in the configuration file. If no value for a variable is found, nothing is shown.

With mysql_plugin, this option need not be given first on the command line, unlike most other MySQL programs that support --print-defaults.

- --verbose, -v

  Verbose mode. Print more information about what the program does. This option can be used multiple times to increase the amount of information.

- --version, -V

  Display version information and exit.

4.4.5 mysql_secure_installation — Improve MySQL Installation Security

This program enables you to improve the security of your MySQL installation in the following ways:

- You can set a password for root accounts.
- You can remove root accounts that are accessible from outside the local host.
- You can remove anonymous-user accounts.
- You can remove the test database (which by default can be accessed by all users, even anonymous users), and privileges that permit anyone to access databases with names that start with test_.

mysql_secure_installation helps you implement security recommendations similar to those described at Section 2.9.2, “Securing the Initial MySQL Accounts”.

As of MySQL 5.7.2, mysql_secure_installation is an executable binary available on all platforms. Before 5.7.2, it was a script available for Unix and Unix-like systems.

Normal usage is to connect to the local MySQL server; invoke mysql_secure_installation without arguments:

```
shell> mysql_secure_installation
```

When executed, mysql_secure_installation prompts you to determine which actions to perform.

As of MySQL 5.7.2, mysql_secure_installation supports these additional features:

- The validate_password plugin can be used for password strength checking. If the plugin is not installed, mysql_secure_installation prompts the user whether to install it. Any passwords entered later are checked using the plugin if it is enabled.
- Most of the usual MySQL client options such as --host and --port can be used on the command line and in option files. For example, to connect to the local server over IPv6 using port 3307, use this command:

  ```
  shell> mysql_secure_installation --host=::1 --port=3307
  ```

mysql_secure_installation supports the following options, which can be specified on the command line or in the [mysql_secure_installation] and [client] groups of an option file. For information about option files, see Section 4.2.3.3, “Using Option Files”.

Table 4.4 `mysql_secure_installation` Options

Format	Option File	Description	Introduced
--defaults-extra-file=file_name		Read option file in addition to the usual option files	5.7.2
--defaults-file=file_name		Read only the given option file	5.7.2
--defaults-group-suffix=str		Option group suffix value	5.7.2
--help		Display help message and exit	5.7.2
--host	host	Host to connect to (IP address or hostname)	5.7.2
--no-defaults		Do not read any option files	5.7.2
--password=password	password	Accepted but always ignored. Whenever mysql_secure_installation is invoked, the user is prompted for a password, regardless.	5.7.2
--port=port_num	port	The TCP/IP port number to use for the connection	5.7.2
--print-defaults		Print defaults	5.7.2
--protocol=type	protocol	The connection protocol to use	5.7.2
--socket=path	socket	For connections to localhost	5.7.2
--ssl	ssl	Enable SSL for connection	5.7.2
--ssl-ca=file_name	ssl-ca	The path to a file that contains a list of trusted SSL CAs	5.7.2
--ssl-capath=dir_name	ssl-capath	The path to a directory that contains trusted SSL CA certificates in PEM format	5.7.2
--ssl-cert=file_name	ssl-cert	The name of the SSL certificate file to use for establishing a secure connection	5.7.2
--ssl-cipher=cipher_list	ssl-cipher	A list of allowable ciphers to use for SSL encryption	5.7.2
--ssl-crl=file_name	ssl-crl	The path to a file that contains certificate revocation lists	5.7.2
--ssl-crlpath=dir_name	ssl-crlpath	The path to a directory that contains certificate revocation list files	5.7.2
--ssl-key=file_name	ssl-key	The name of the SSL key file to use for establishing a secure connection	5.7.2
--ssl-verify-server-cert	ssl-verify-server-cert	The server's Common Name value in its certificate is verified against the host name used when connecting to the server	5.7.2
--use-default	use-default	Execute with no user interactivity	5.7.4
--user=user_name	user	MySQL user name to use when connecting to server	5.7.2

- `--help`, `-?`

  Display a help message and exit.

- `--defaults-extra-file=`*`file_name`*

Read this option file after the global option file but (on Unix) before the user option file. If the file does not exist or is otherwise inaccessible, an error occurs. *file_name* is interpreted relative to the current directory if given as a relative path name rather than a full path name.

- --defaults-file=*file_name*

  Use only the given option file. If the file does not exist or is otherwise inaccessible, an error occurs. *file_name* is interpreted relative to the current directory if given as a relative path name rather than a full path name.

- --defaults-group-suffix=*str*

  Read not only the usual option groups, but also groups with the usual names and a suffix of *str*. For example, mysql_secure_installation normally reads the [client] and [mysql_secure_installation] groups. If the --defaults-group-suffix=_other option is given, mysql_secure_installation also reads the [client_other] and [mysql_secure_installation_other] groups.

- --host=*host_name*, -h *host_name*

  Connect to the MySQL server on the given host.

- --no-defaults

  Do not read any option files. If program startup fails due to reading unknown options from an option file, --no-defaults can be used to prevent them from being read.

  The exception is that the .mylogin.cnf file, if it exists, is read in all cases. This permits passwords to be specified in a safer way than on the command line even when --no-defaults is used. (.mylogin.cnf is created by the mysql_config_editor utility. See Section 4.6.6, “mysql_config_editor — MySQL Configuration Utility”.)

- --password=*password*, -p *password*

  This option is accepted but ignored. Whether or not this option is used, mysql_secure_installation always prompts the user for a password.

- --port=*port_num*, -P *port_num*

  The TCP/IP port number to use for the connection.

- --print-defaults

  Print the program name and all options that it gets from option files.

- --protocol={TCP|SOCKET|PIPE|MEMORY}

  The connection protocol to use for connecting to the server. It is useful when the other connection parameters normally would cause a protocol to be used other than the one you want. For details on the permissible values, see Section 4.2.2, “Connecting to the MySQL Server”.

- --socket=*path*, -S *path*

  For connections to localhost, the Unix socket file to use, or, on Windows, the name of the named pipe to use.

- --ssl*

Options that begin with --ssl specify whether to connect to the server using SSL and indicate where to find SSL keys and certificates. See Section 6.3.11.4, “SSL Command Options”.

- --use-default

  Execute noninteractively. This option can be used for unattended installation operations. This option was added in MySQL 5.7.4.

- --user=*user_name*, -u *user_name*

  The MySQL user name to use when connecting to the server.

4.4.6 mysql_tzinfo_to_sql — Load the Time Zone Tables

The mysql_tzinfo_to_sql program loads the time zone tables in the mysql database. It is used on systems that have a *zoneinfo* database (the set of files describing time zones). Examples of such systems are Linux, FreeBSD, Solaris, and Mac OS X. One likely location for these files is the /usr/share/zoneinfo directory (/usr/share/lib/zoneinfo on Solaris). If your system does not have a zoneinfo database, you can use the downloadable package described in Section 10.6, “MySQL Server Time Zone Support”.

mysql_tzinfo_to_sql can be invoked several ways:

```
shell> mysql_tzinfo_to_sql tz_dir
shell> mysql_tzinfo_to_sql tz_file tz_name
shell> mysql_tzinfo_to_sql --leap tz_file
```

For the first invocation syntax, pass the zoneinfo directory path name to mysql_tzinfo_to_sql and send the output into the mysql program. For example:

```
shell> mysql_tzinfo_to_sql /usr/share/zoneinfo | mysql -u root mysql
```

mysql_tzinfo_to_sql reads your system's time zone files and generates SQL statements from them. mysql processes those statements to load the time zone tables.

The second syntax causes mysql_tzinfo_to_sql to load a single time zone file *tz_file* that corresponds to a time zone name *tz_name*:

```
shell> mysql_tzinfo_to_sql tz_file tz_name | mysql -u root mysql
```

If your time zone needs to account for leap seconds, invoke mysql_tzinfo_to_sql using the third syntax, which initializes the leap second information. *tz_file* is the name of your time zone file:

```
shell> mysql_tzinfo_to_sql --leap tz_file | mysql -u root mysql
```

After running mysql_tzinfo_to_sql, it is best to restart the server so that it does not continue to use any previously cached time zone data.

4.4.7 mysql_upgrade — Check and Upgrade MySQL Tables

mysql_upgrade examines all tables in all databases for incompatibilities with the current version of MySQL Server. mysql_upgrade also upgrades the system tables so that you can take advantage of new privileges or capabilities that might have been added.

mysql_upgrade should be executed each time you upgrade MySQL.

Important

If you upgrade to MySQL 5.7.2 or later from a version older than 5.7.2, a change to the mysql.user table requires a special sequence of steps to perform an upgrade using mysql_upgrade. For details, see Section 2.10.1.2, "Upgrading from MySQL 5.6 to 5.7".

If mysql_upgrade finds that a table has a possible incompatibility, it performs a table check and, if problems are found, attempts a table repair. If the table cannot be repaired, see Section 2.10.4, "Rebuilding or Repairing Tables or Indexes" for manual table repair strategies.

Note

On Windows Server 2008, Vista, and newer, you must run mysql_upgrade with administrator privileges. You can do this by running a Command Prompt as Administrator and running the command. Failure to do so may result in the upgrade failing to execute correctly.

Caution

You should always back up your current MySQL installation *before* performing an upgrade. See Section 7.2, "Database Backup Methods".

Some upgrade incompatibilities may require special handling before you upgrade your MySQL installation and run mysql_upgrade. See Section 2.10.1, "Upgrading MySQL", for instructions on determining whether any such incompatibilities apply to your installation and how to handle them.

To use mysql_upgrade, make sure that the server is running, and then invoke it like this:

```
shell> mysql_upgrade [options]
```

After running mysql_upgrade, stop the server and restart it so that any changes made to the system tables take effect.

mysql_upgrade executes the following commands to check and repair tables and to upgrade the system tables:

```
mysqlcheck --all-databases --check-upgrade --auto-repair
mysql < fix_priv_tables
mysqlcheck --all-databases --check-upgrade --fix-db-names --fix-table-names
```

Notes about the preceding commands:

- Because mysql_upgrade invokes mysqlcheck with the --all-databases option, it processes all tables in all databases, which might take a long time to complete. Each table is locked and therefore unavailable to other sessions while it is being processed. Check and repair operations can be time-consuming, particularly for large tables.

- For details about what checks the --check-upgrade option entails, see the description of the FOR UPGRADE option of the CHECK TABLE statement (see Section 13.7.2.2, "CHECK TABLE Syntax").

- *fix_priv_tables* represents a script generated internally by mysql_upgrade that contains SQL statements to upgrade the tables in the mysql database.

All checked and repaired tables are marked with the current MySQL version number. This ensures that next time you run mysql_upgrade with the same version of the server, it can tell whether there is any need to check or repair the table again.

mysql_upgrade also saves the MySQL version number in a file named mysql_upgrade_info in the data directory. This is used to quickly check whether all tables have been checked for this release so that table-checking can be skipped. To ignore this file and perform the check regardless, use the --force option.

If you install MySQL from RPM packages on Linux, you must install the server and client RPMs. mysql_upgrade is included in the server RPM but requires the client RPM because the latter includes mysqlcheck. (See Section 2.5.3, “Installing MySQL on Linux Using RPM Packages”.)

As of MySQL 5.7.2, mysql_upgrade checks user table rows and, for any row with an empty plugin column, sets that column to 'mysql_native_password' or 'mysql_old_password' depending on the hash format of the Password column value.

mysql_upgrade does not upgrade the contents of the help tables. For upgrade instructions, see Section 5.1.10, “Server-Side Help”.

mysql_upgrade runs by default as the MySQL root user. If the root password is expired when you run mysql_upgrade, you will see a message that your password is expired and that mysql_upgrade failed as a result. To correct this, reset the root password to unexpire it and run mysql_upgrade again:

```
shell> mysql -u root -p
Enter password: ****  <- enter root password here
mysql> SET PASSWORD = PASSWORD('root-password');
mysql> quit

shell> mysql_upgrade
```

mysql_upgrade supports the following options, which can be specified on the command line or in the [mysql_upgrade] and [client] groups of an option file. Other options are passed to mysqlcheck. For example, it might be necessary to specify the --password[=*password*] option. For information about option files, see Section 4.2.3.3, “Using Option Files”.

Table 4.5 mysql_upgrade Options

Format	Option File	Description	Introduced	Removed
--basedir	basedir	Not used; exists only for compatibility with some very old applications		5.7.2
--character-sets-dir=path	character-sets-dir	Directory where character sets are.		
--compress	compress	Use compression in server/client protocol.		
--datadir=path	datadir	Not used; exists only for compatibility with some very old applications		5.7.2
--debug[=#]	debug	If this is a non-debug version, catch error and exit.		
--debug-check	debug-check	--debug-check Check memory and open file usage at exit.		
--default-auth=plugin	default-auth=plugin	The authentication plugin to use		
--default-character-set=name	default-character-set	Set the default character set.		
--defaults-extra-file=file_name		Read option file in addition to the usual option files		

Format	Option File	Description	Introduced	Removed
--defaults-file=file_name		Read only the given option file		
--defaults-group-suffix=str		Option group suffix value		
--force	force	Force execution even if mysql_upgrade has already been executed for the current version of MySQL.		
--help	help	Display a help message and exit		
--host=name	host	Connect to host.		
--no-defaults		Do not read any option files		
--password[=name]	password	Password to use when connecting to server. If password is not given it's solicited on the tty.		
--plugin-dir=path	plugin-dir=path	The directory where plugins are located		
--port=#	port	Port number to use for connection or 0 for default to, in order of preference, my.cnf, $MYSQL_TCP_PORT, /etc/services, built-in default (3306).		
--print-defaults		Print defaults		
--protocol=name	protocol	The connection protocol (TCP=default, socket, pipe, memory)		
--socket=name	socket	Socket file to use for connection.		
--tmpdir=path	tmpdir	Directory for temporary files		
--user=name	user	User for login if not current user.		
--verbose	verbose	Show more information about the process		
--version-check	version-check	Check for proper server version	5.7.2	
--write-binlog	write-binlog	Enables binary logging of all commands including mysqlcheck.		

- `--help`

  Display a short help message and exit.

- `--basedir=*path*`

  The path to the MySQL installation directory. This option was removed in MySQL 5.7.2.

- `--datadir=*path*`

  The path to the data directory. This option was removed in MySQL 5.7.2.

- `--debug=*debug_options*`, `-# *debug_options*`

  Write a debugging log. A typical *`debug_options`* string is `d:t:o,*file_name*`. The default is `d:t:O,/tmp/mysql_upgrade.trace`.

- `--debug-check`

  Print some debugging information when the program exits.

- `--debug-info`, `-T`

  Print debugging information and memory and CPU usage statistics when the program exits.

- `--default-auth=plugin`

  The client-side authentication plugin to use. See Section 6.3.8, “Pluggable Authentication”.

- `--defaults-extra-file=file_name`

  Read this option file after the global option file but (on Unix) before the user option file. If the file does not exist or is otherwise inaccessible, an error occurs. *file_name* is interpreted relative to the current directory if given as a relative path name rather than a full path name.

- `--defaults-file=file_name`

  Use only the given option file. If the file does not exist or is otherwise inaccessible, an error occurs. *file_name* is interpreted relative to the current directory if given as a relative path name rather than a full path name.

- `--defaults-group-suffix=str`

  Read not only the usual option groups, but also groups with the usual names and a suffix of *str*. For example, `mysql_upgrade` normally reads the `[client]` and `[mysql_upgrade]` groups. If the `--defaults-group-suffix=_other` option is given, `mysql_upgrade` also reads the `[client_other]` and `[mysql_upgrade_other]` groups.

- `--force`

  Ignore the `mysql_upgrade_info` file and force execution of `mysqlcheck` even if `mysql_upgrade` has already been executed for the current version of MySQL.

- `--no-defaults`

  Do not read any option files. If program startup fails due to reading unknown options from an option file, `--no-defaults` can be used to prevent them from being read.

  The exception is that the `.mylogin.cnf` file, if it exists, is read in all cases. This permits passwords to be specified in a safer way than on the command line even when `--no-defaults` is used. (`.mylogin.cnf` is created by the `mysql_config_editor` utility. See Section 4.6.6, “mysql_config_editor — MySQL Configuration Utility”.)

- `--plugin-dir=path`

  The directory in which to look for plugins. It may be necessary to specify this option if the `--default-auth` option is used to specify an authentication plugin but `mysql_upgrade` does not find it. See Section 6.3.8, “Pluggable Authentication”.

- `--print-defaults`

  Print the program name and all options that it gets from option files.

- `--tmpdir=path`, `-t path`

  The path name of the directory to use for creating temporary files.

- `--upgrade-system-tables`, `-s`

  Upgrade only the system tables, do not upgrade data.

- `--user=user_name`, `-u user_name`

  The MySQL user name to use when connecting to the server. The default user name is `root`.

- `--verbose`

  Verbose mode. Print more information about what the program does.

- `--version-check`, `-k`

  Check the version of the server to which `mysql_upgrade` is connecting to verify that it is the same as the version for which `mysql_upgrade` was built. If not, `mysql_upgrade` exits. This option is enabled by default; to disable the check, use `--skip-version-check`. This option was added in MySQL 5.7.2.

- `--write-binlog`

  Binary logging by `mysql_upgrade` is disabled by default, and you must invoke the program explicitly with `--write-binlog` if you want its actions to be written to the binary log.

  Running `mysql_upgrade` is not recommended with a MySQL Server that is running with global transaction identifiers enabled (Bug #13833710). This is because enabling GTIDs means that any updates which `mysql_upgrade` might need to perform on system tables using a nontransactional storage engine such as `MyISAM` to fail. See Section 16.1.3.4, “Restrictions on Replication with GTIDs”, for more information.

4.5 MySQL Client Programs

This section describes client programs that connect to the MySQL server.

4.5.1 `mysql` — The MySQL Command-Line Tool

`mysql` is a simple SQL shell with input line editing capabilities. It supports interactive and noninteractive use. When used interactively, query results are presented in an ASCII-table format. When used noninteractively (for example, as a filter), the result is presented in tab-separated format. The output format can be changed using command options.

If you have problems due to insufficient memory for large result sets, use the `--quick` option. This forces `mysql` to retrieve results from the server a row at a time rather than retrieving the entire result set and buffering it in memory before displaying it. This is done by returning the result set using the `mysql_use_result()` C API function in the client/server library rather than `mysql_store_result()`.

Using `mysql` is very easy. Invoke it from the prompt of your command interpreter as follows:

```
shell> mysql db_name
```

Or:

```
shell> mysql --user=user_name --password=your_password db_name
```

Then type an SQL statement, end it with “`;`”, `\g`, or `\G` and press Enter.

Typing **Control+C** interrupts the current statement if there is one, or cancels any partial input line otherwise.

You can execute SQL statements in a script file (batch file) like this:

```
shell> mysql db_name < script.sql > output.tab
```

On Unix, the mysql client logs statements executed interactively to a history file. See Section 4.5.1.3, “mysql Logging”.

4.5.1.1 mysql Options

mysql supports the following options, which can be specified on the command line or in the [mysql] and [client] groups of an option file. For information about option files, see Section 4.2.3.3, “Using Option Files”.

Table 4.6 mysql Options

Format	Option File	Description	Introduced
--auto-rehash	auto-rehash	Enable automatic rehashing	
--auto-vertical-output	auto-vertical-output	Enable automatic vertical result set display	
--batch	batch	Don't use history file	
--binary-mode	binary-mode	Disable \r\n - to - \n translation and treatment of \0 as end-of-query	
--bind-address=ip_address	bind-address	Use the specified network interface to connect to the MySQL Server	
--character-sets-dir=path	character-sets-dir	Set the default character set	
--column-names	column-names	Write column names in results	
--column-type-info	column-type-info	Display result set metadata	
--comments	comments	Whether to retain or strip comments in statements sent to the server	
--compress	compress	Compress all information sent between the client and the server	
--connect-expired-password		Indicate to server that client can handle expired-password sandbox mode.	5.7.2
--connect_timeout=value	connect_timeout	The number of seconds before connection timeout	
--database=dbname	database	The database to use	
--debug[=debug_options]	debug	Write a debugging log	
--debug-check	debug-check	Print debugging information when the program exits	
--debug-info	debug-info	Print debugging information, memory and CPU statistics when the program exits	
--default-auth=plugin	default-auth=plugin	The authentication plugin to use	
--default-character-set=charset_name	default-character-set	Use charset_name as the default character set	
--defaults-extra-file=file_name		Read option file in addition to the usual option files	

Format	Option File	Description	Introduced
--defaults-file=file_name		Read only the given option file	
--defaults-group-suffix=str		Option group suffix value	
--delimiter=str	delimiter	Set the statement delimiter	
--enable-cleartext-plugin	enable-cleartext-plugin	Enable cleartext authentication plugin	
--execute=statement	execute	Execute the statement and quit	
--force	force	Continue even if an SQL error occurs	
--help		Display help message and exit	
--histignore=pattern_list	histignore	Patterns specifying which statements to ignore for logging	
--host=host_name	host	Connect to the MySQL server on the given host	
--html	html	Produce HTML output	
--ignore-spaces	ignore-spaces	Ignore spaces after function names	
--init-command=str	init-command	SQL statement to execute after connecting	
--line-numbers	line-numbers	Write line numbers for errors	
--local-infile[={0|1}]	local-infile	Enable or disable for LOCAL capability for LOAD DATA INFILE	
--login-path=name		Read login path options from .mylogin.cnf	
--max_allowed_packet=value	max_allowed_packet	The maximum packet length to send to or receive from the server	
--max_join_size=value	max_join_size	The automatic limit for rows in a join when using --safe-updates	
--named-commands	named-commands	Enable named mysql commands	
--net_buffer_length=value	net_buffer_length	The buffer size for TCP/IP and socket communication	
--no-auto-rehash		Disable automatic rehashing	
--no-beep	no-beep	Do not beep when errors occur	
--no-defaults		Do not read any option files	
--one-database	one-database	Ignore statements except those for the default database named on the command line	
--pager[=command]	pager	Use the given command for paging query output	
--password[=password]	password	The password to use when connecting to the server	
--pipe		On Windows, connect to server using a named pipe	
--plugin-dir=path	plugin-dir=path	The directory where plugins are located	
--port=port_num	port	The TCP/IP port number to use for the connection	

Format	Option File	Description	Introduced
--print-defaults		Print defaults	
--prompt=format_str	prompt	Set the prompt to the specified format	
--protocol=type	protocol	The connection protocol to use	
--quick	quick	Do not cache each query result	
--raw	raw	Write column values without escape conversion	
--reconnect	reconnect	If the connection to the server is lost, automatically try to reconnect	
--safe-updates	safe-updates	Allow only UPDATE and DELETE statements that specify key values	
--secure-auth	secure-auth	Do not send passwords to the server in old (pre-4.1.1) format	
--select_limit=value	select_limit	The automatic limit for SELECT statements when using --safe-updates	
--server-public-key-path=file_name	server-public-key-path=file_name	Path name to file containing RSA public key	
--show-warnings	show-warnings	Show warnings after each statement if there are any	
--sigint-ignore	sigint-ignore	Ignore SIGINT signals (typically the result of typing Control+C)	
--silent	silent	Silent mode	
--skip-auto-rehash	skip-auto-rehash	Disable automatic rehashing	
--skip-column-names	skip-column-names	Do not write column names in results	
--skip-line-numbers	skip-line-numbers	Skip line numbers for errors	
--skip-named-commands	skip-named-commands	Disable named mysql commands	
--skip-pager	skip-pager	Disable paging	
--skip-reconnect	skip-reconnect	Disable reconnecting	
--socket=path	socket	For connections to localhost	
--ssl[=TRUE|FALSE]		Enable an SSL connection to the server. This option is set to TRUE when any other SSL option is used, and so is normally not needed.	
--ssl-ca=file_name	ssl-ca	The path to a file that contains a list of trusted SSL CAs	
--ssl-capath=dir_name	ssl-capath	The path to a directory that contains trusted SSL CA certificates in PEM format	
--ssl-cert=file_name	ssl-cert	The name of the SSL certificate file to use for establishing a secure connection	
--ssl-cipher=cipher_list	ssl-cipher	A list of allowable ciphers to use for SSL encryption	

Format	Option File	Description	Introduced
--ssl-crl=file_name	ssl-crl	The path to a file that contains certificate revocation lists	
--ssl-crlpath=dir_name	ssl-crlpath	The path to a directory that contains certificate revocation list files	
--ssl-key=file_name	ssl-key	The name of the SSL key file to use for establishing a secure connection	
--ssl-verify-server-cert	ssl-verify-server-cert	The server's Common Name value in its certificate is verified against the host name used when connecting to the server	
--syslog	syslog	Log interactive statements to syslog	5.7.1
--table	table	Display output in tabular format	
--tee=file_name	tee	Append a copy of output to the given file	
--unbuffered	unbuffered	Flush the buffer after each query	
--user=user_name	user	MySQL user name to use when connecting to server	
--verbose		Verbose mode	
--version		Display version information and exit	
--vertical	vertical	Print query output rows vertically (one line per column value)	
--wait	wait	If the connection cannot be established, wait and retry instead of aborting	
--xml	xml	Produce XML output	

- `--help`, `-?`

  Display a help message and exit.

- `--auto-rehash`

  Enable automatic rehashing. This option is on by default, which enables database, table, and column name completion. Use `--disable-auto-rehash` to disable rehashing. That causes `mysql` to start faster, but you must issue the `rehash` command if you want to use name completion.

  To complete a name, enter the first part and press Tab. If the name is unambiguous, `mysql` completes it. Otherwise, you can press Tab again to see the possible names that begin with what you have typed so far. Completion does not occur if there is no default database.

- `--auto-vertical-output`

  Cause result sets to be displayed vertically if they are too wide for the current window, and using normal tabular format otherwise. (This applies to statements terminated by `;` or `\G`.)

- `--batch`, `-B`

  Print results using tab as the column separator, with each row on a new line. With this option, `mysql` does not use the history file.

  Batch mode results in nontabular output format and escaping of special characters. Escaping may be disabled by using raw mode; see the description for the `--raw` option.

- `--binary-mode`

  This option helps when processing `mysqlbinlog` output that may contain `BLOB` values. By default, `mysql` translates `\r\n` in statement strings to `\n` and interprets `\0` as the statement terminator. `--binary-mode` disables both features. It also disables all `mysql` commands except `charset` and `delimiter` in non-interactive mode (for input piped to `mysql` or loaded using the `source` command).

- `--bind-address=ip_address`

  On a computer having multiple network interfaces, this option can be used to select which interface is employed when connecting to the MySQL server.

- `--character-sets-dir=path`

  The directory where character sets are installed. See Section 10.5, “Character Set Configuration”.

- `--column-names`

  Write column names in results.

- `--column-type-info`, `-m`

  Display result set metadata.

- `--comments`, `-c`

  Whether to preserve comments in statements sent to the server. The default is --skip-comments (discard comments), enable with --comments (preserve comments).

- `--compress`, `-C`

  Compress all information sent between the client and the server if both support compression.

- `--connect-expired-password`

  Indicate to the server that the client is can handle sandbox mode if the account used to connect has an expired password. This can be useful for noninteractive invocations of `mysql` because normally the server disconnects noninteractive clients that attempt to connect using an account with an expired password. (See Section 6.3.7, “Password Expiration and Sandbox Mode”.) This option was added in MySQL 5.7.2.

- `--database=db_name`, `-D db_name`

  The database to use. This is useful primarily in an option file.

- `--debug[=debug_options]`, `-# [debug_options]`

  Write a debugging log. A typical *`debug_options`* string is `d:t:o,file_name`. The default is `d:t:o,/tmp/mysql.trace`.

- `--debug-check`

  Print some debugging information when the program exits.

- `--debug-info`, `-T`

  Print debugging information and memory and CPU usage statistics when the program exits.

- `--default-auth=plugin`

The client-side authentication plugin to use. See Section 6.3.8, “Pluggable Authentication”.

- `--default-character-set=charset_name`

  Use *`charset_name`* as the default character set for the client and connection.

  A common issue that can occur when the operating system uses `utf8` or another multi-byte character set is that output from the `mysql` client is formatted incorrectly, due to the fact that the MySQL client uses the `latin1` character set by default. You can usually fix such issues by using this option to force the client to use the system character set instead.

  See Section 10.5, “Character Set Configuration”, for more information.

- `--defaults-extra-file=file_name`

  Read this option file after the global option file but (on Unix) before the user option file. If the file does not exist or is otherwise inaccessible, an error occurs. *`file_name`* is interpreted relative to the current directory if given as a relative path name rather than a full path name.

- `--defaults-file=file_name`

  Use only the given option file. If the file does not exist or is otherwise inaccessible, an error occurs. *`file_name`* is interpreted relative to the current directory if given as a relative path name rather than a full path name.

- `--defaults-group-suffix=str`

  Read not only the usual option groups, but also groups with the usual names and a suffix of *`str`*. For example, `mysql` normally reads the `[client]` and `[mysql]` groups. If the `--defaults-group-suffix=_other` option is given, `mysql` also reads the `[client_other]` and `[mysql_other]` groups.

- `--delimiter=str`

  Set the statement delimiter. The default is the semicolon character (“`;`”).

- `--disable-named-commands`

  Disable named commands. Use the `\*` form only, or use named commands only at the beginning of a line ending with a semicolon (“`;`”). `mysql` starts with this option *enabled* by default. However, even with this option, long-format commands still work from the first line. See Section 4.5.1.2, “`mysql` Commands”.

- `--enable-cleartext-plugin`

  Enable the `mysql_clear_password` cleartext authentication plugin. (See Section 6.3.9.5, “The Cleartext Client-Side Authentication Plugin”.)

- `--execute=statement`, `-e statement`

  Execute the statement and quit. The default output format is like that produced with `--batch`. See Section 4.2.3.1, “Using Options on the Command Line”, for some examples. With this option, `mysql` does not use the history file.

- `--force`, `-f`

  Continue even if an SQL error occurs.

- `--histignore`

A colon-separated list of one or more patterns specifying statements to ignore for logging purposes. These patterns are added to the default pattern list ("*IDENTIFIED*:*PASSWORD*"). The value specified for this option affects logging of statements written to the history file, and to syslog if the --syslog option is given. For more information, see Section 4.5.1.3, “mysql Logging”.

- --host=*host_name*, -h *host_name*

  Connect to the MySQL server on the given host.

- --html, -H

  Produce HTML output.

- --ignore-spaces, -i

  Ignore spaces after function names. The effect of this is described in the discussion for the IGNORE_SPACE SQL mode (see Section 5.1.7, “Server SQL Modes”).

- --init-command=str

  SQL statement to execute after connecting to the server. If auto-reconnect is enabled, the statement is executed again after reconnection occurs.

- --line-numbers

  Write line numbers for errors. Disable this with --skip-line-numbers.

- --local-infile[={0|1}]

  Enable or disable LOCAL capability for LOAD DATA INFILE. With no value, the option enables LOCAL. The option may be given as --local-infile=0 or --local-infile=1 to explicitly disable or enable LOCAL. Enabling LOCAL has no effect if the server does not also support it.

- --login-path=*name*

  Read options from the named login path in the .mylogin.cnf login file. A “login path” is an option group that permits only a limited set of options: host, user, and password. Think of a login path as a set of values that indicate the server host and the credentials for authenticating with the server. To create the login file, use the mysql_config_editor utility. See Section 4.6.6, “mysql_config_editor — MySQL Configuration Utility”.

- --named-commands, -G

  Enable named mysql commands. Long-format commands are permitted, not just short-format commands. For example, quit and \q both are recognized. Use --skip-named-commands to disable named commands. See Section 4.5.1.2, “mysql Commands”.

- --no-auto-rehash, -A

  This has the same effect as -skip-auto-rehash. See the description for --auto-rehash.

- --no-beep, -b

  Do not beep when errors occur.

- --no-defaults

Do not read any option files. If program startup fails due to reading unknown options from an option file, --no-defaults can be used to prevent them from being read.

The exception is that the .mylogin.cnf file, if it exists, is read in all cases. This permits passwords to be specified in a safer way than on the command line even when --no-defaults is used. (.mylogin.cnf is created by the mysql_config_editor utility. See Section 4.6.6, “mysql_config_editor — MySQL Configuration Utility”.)

- --one-database, -o

  Ignore statements except those that occur while the default database is the one named on the command line. This option is rudimentary and should be used with care. Statement filtering is based only on USE statements.

  Initially, mysql executes statements in the input because specifying a database *db_name* on the command line is equivalent to inserting USE *db_name* at the beginning of the input. Then, for each USE statement encountered, mysql accepts or rejects following statements depending on whether the database named is the one on the command line. The content of the statements is immaterial.

  Suppose that mysql is invoked to process this set of statements:

  ```
  DELETE FROM db2.t2;
  USE db2;
  DROP TABLE db1.t1;
  CREATE TABLE db1.t1 (i INT);
  USE db1;
  INSERT INTO t1 (i) VALUES(1);
  CREATE TABLE db2.t1 (j INT);
  ```

  If the command line is mysql --force --one-database db1, mysql handles the input as follows:

  - The DELETE statement is executed because the default database is db1, even though the statement names a table in a different database.

  - The DROP TABLE and CREATE TABLE statements are not executed because the default database is not db1, even though the statements name a table in db1.

  - The INSERT and CREATE TABLE statements are executed because the default database is db1, even though the CREATE TABLE statement names a table in a different database.

- --pager[=*command*]

  Use the given command for paging query output. If the command is omitted, the default pager is the value of your PAGER environment variable. Valid pagers are less, more, cat [> filename], and so forth. This option works only on Unix and only in interactive mode. To disable paging, use --skip-pager. Section 4.5.1.2, “mysql Commands”, discusses output paging further.

- --password[=*password*], -p[*password*]

  The password to use when connecting to the server. If you use the short option form (-p), you *cannot* have a space between the option and the password. If you omit the *password* value following the --password or -p option on the command line, mysql prompts for one.

  Specifying a password on the command line should be considered insecure. See Section 6.1.2.1, “End-User Guidelines for Password Security”. You can use an option file to avoid giving the password on the command line.

- `--pipe`, `-W`

  On Windows, connect to the server using a named pipe. This option applies only if the server supports named-pipe connections.

- `--plugin-dir=path`

  The directory in which to look for plugins. It may be necessary to specify this option if the `--default-auth` option is used to specify an authentication plugin but `mysql` does not find it. See Section 6.3.8, “Pluggable Authentication”.

- `--port=port_num`, `-P port_num`

  The TCP/IP port number to use for the connection.

- `--print-defaults`

  Print the program name and all options that it gets from option files.

- `--prompt=format_str`

  Set the prompt to the specified format. The default is `mysql>`. The special sequences that the prompt can contain are described in Section 4.5.1.2, “`mysql` Commands”.

- `--protocol={TCP|SOCKET|PIPE|MEMORY}`

  The connection protocol to use for connecting to the server. It is useful when the other connection parameters normally would cause a protocol to be used other than the one you want. For details on the permissible values, see Section 4.2.2, “Connecting to the MySQL Server”.

- `--quick`, `-q`

  Do not cache each query result, print each row as it is received. This may slow down the server if the output is suspended. With this option, `mysql` does not use the history file.

- `--raw`, `-r`

  For tabular output, the “boxing” around columns enables one column value to be distinguished from another. For nontabular output (such as is produced in batch mode or when the `--batch` or `--silent` option is given), special characters are escaped in the output so they can be identified easily. Newline, tab, `NUL`, and backslash are written as `\n`, `\t`, `\0`, and `\\`. The `--raw` option disables this character escaping.

  The following example demonstrates tabular versus nontabular output and the use of raw mode to disable escaping:

```
% mysql
mysql> SELECT CHAR(92);
+----------+
| CHAR(92) |
+----------+
| \        |
+----------+

% mysql -s
mysql> SELECT CHAR(92);
CHAR(92)
\\

% mysql -s -r
```

```
mysql> SELECT CHAR(92);
CHAR(92)
\
```

- `--reconnect`

  If the connection to the server is lost, automatically try to reconnect. A single reconnect attempt is made each time the connection is lost. To suppress reconnection behavior, use `--skip-reconnect`.

- `--safe-updates`, `--i-am-a-dummy`, `-U`

  Permit only those `UPDATE` and `DELETE` statements that specify which rows to modify by using key values. If you have set this option in an option file, you can override it by using `--safe-updates` on the command line. See Section 4.5.1.6, "`mysql` Tips", for more information about this option.

- `--secure-auth`

  Do not send passwords to the server in old (pre-4.1) format. This prevents connections except for servers that use the newer password format. This option is enabled by default; use `--skip-secure-auth` to disable it.

  Note

  Passwords that use the pre-4.1 hashing method are less secure than passwords that use the native password hashing method and should be avoided. Pre-4.1 passwords are deprecated and support for them will be removed in a future MySQL release. For account upgrade instructions, see Section 6.3.9.3, "Migrating Away from Pre-4.1 Password Hashing and the `mysql_old_password` Plugin".

- `--server-public-key-path=file_name`

  The path name to a file containing the server RSA public key. The file must be in PEM format. The public key is used for RSA encryption of the client password for connections to the server made using accounts that authenticate with the `sha256_password` plugin. This option is ignored for client accounts that do not authenticate with that plugin. It is also ignored if password encryption is not needed, as is the case when the client connects to the server using an SSL connection.

  The server sends the public key to the client as needed, so it is not necessary to use this option for RSA password encryption to occur. It is more efficient to do so because then the server need not send the key.

  For additional discussion regarding use of the `sha256_password` plugin, including how to get the RSA public key, see Section 6.3.9.4, "The SHA-256 Authentication Plugin".

  This option is available only if MySQL was built using OpenSSL.

- `--show-warnings`

  Cause warnings to be shown after each statement if there are any. This option applies to interactive and batch mode.

- `--sigint-ignore`

  Ignore `SIGINT` signals (typically the result of typing **Control+C**).

- `--silent`, `-s`

Silent mode. Produce less output. This option can be given multiple times to produce less and less output.

This option results in nontabular output format and escaping of special characters. Escaping may be disabled by using raw mode; see the description for the `--raw` option.

- `--skip-column-names`, `-N`

  Do not write column names in results.

- `--skip-line-numbers`, `-L`

  Do not write line numbers for errors. Useful when you want to compare result files that include error messages.

- `--socket=`*`path`*, `-S` *`path`*

  For connections to `localhost`, the Unix socket file to use, or, on Windows, the name of the named pipe to use.

- `--ssl*`

  Options that begin with `--ssl` specify whether to connect to the server using SSL and indicate where to find SSL keys and certificates. See Section 6.3.11.4, “SSL Command Options”.

- `--syslog`, `-j`

  This option causes `mysql` to send interactive statements to the system logging facility. On Unix, this is `syslog`; on Windows, it is the Windows Event Log. The destination where logged messages appear is system dependent. On Linux, the destination is often the `/var/log/messages` file.

  Here is a sample of output generated on Linux by using `--syslog`. This output is formatted for readability; each logged message actually takes a single line.

```
Mar  7 12:39:25 myhost MysqlClient[20824]:
  SYSTEM_USER:'oscar', MYSQL_USER:'my_oscar', CONNECTION_ID:23,
  DB_SERVER:'127.0.0.1', DB:'--', QUERY:'USE test;'
Mar  7 12:39:28 myhost MysqlClient[20824]:
  SYSTEM_USER:'oscar', MYSQL_USER:'my_oscar', CONNECTION_ID:23,
  DB_SERVER:'127.0.0.1', DB:'test', QUERY:'SHOW TABLES;'
```

  For more information, see Section 4.5.1.3, “`mysql` Logging”.

  The `--syslog` option was added in MySQL 5.7.1.

- `--table`, `-t`

  Display output in table format. This is the default for interactive use, but can be used to produce table output in batch mode.

- `--tee=`*`file_name`*

  Append a copy of output to the given file. This option works only in interactive mode. Section 4.5.1.2, “`mysql` Commands”, discusses tee files further.

- `--unbuffered`, `-n`

  Flush the buffer after each query.

- `--user=user_name`, `-u user_name`

  The MySQL user name to use when connecting to the server.

- `--verbose`, `-v`

  Verbose mode. Produce more output about what the program does. This option can be given multiple times to produce more and more output. (For example, `-v -v -v` produces table output format even in batch mode.)

- `--version`, `-V`

  Display version information and exit.

- `--vertical`, `-E`

  Print query output rows vertically (one line per column value). Without this option, you can specify vertical output for individual statements by terminating them with `\G`.

- `--wait`, `-w`

  If the connection cannot be established, wait and retry instead of aborting.

- `--xml`, `-X`

  Produce XML output.

  ```
  <field name="column_name">NULL</field>
  ```

  The output when `--xml` is used with `mysql` matches that of `mysqldump --xml`. See Section 4.5.4, “`mysqldump` — A Database Backup Program” for details.

  The XML output also uses an XML namespace, as shown here:

  ```
  shell> mysql --xml -uroot -e "SHOW VARIABLES LIKE 'version%'"
  <?xml version="1.0"?>

  <resultset statement="SHOW VARIABLES LIKE 'version%'" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <row>
  <field name="Variable_name">version</field>
  <field name="Value">5.0.40-debug</field>
  </row>

  <row>
  <field name="Variable_name">version_comment</field>
  <field name="Value">Source distribution</field>
  </row>

  <row>
  <field name="Variable_name">version_compile_machine</field>
  <field name="Value">i686</field>
  </row>

  <row>
  <field name="Variable_name">version_compile_os</field>
  <field name="Value">suse-linux-gnu</field>
  </row>
  </resultset>
  ```

  (See Bug #25946.)

You can also set the following variables by using `--var_name=value`.

- `connect_timeout`

  The number of seconds before connection timeout. (Default value is `0`.)

- `max_allowed_packet`

  The maximum size of the buffer for client/server communication. The default is 16MB, the maximum is 1GB.

- `max_join_size`

  The automatic limit for rows in a join when using `--safe-updates`. (Default value is 1,000,000.)

- `net_buffer_length`

  The buffer size for TCP/IP and socket communication. (Default value is 16KB.)

- `select_limit`

  The automatic limit for `SELECT` statements when using `--safe-updates`. (Default value is 1,000.)

4.5.1.2 mysql Commands

`mysql` sends each SQL statement that you issue to the server to be executed. There is also a set of commands that `mysql` itself interprets. For a list of these commands, type `help` or `\h` at the `mysql>` prompt:

```
mysql> help

List of all MySQL commands:
Note that all text commands must be first on line and end with ';'
?          (\?) Synonym for `help'.
clear      (\c) Clear the current input statement.
connect    (\r) Reconnect to the server. Optional arguments are db and host.
delimiter (\d) Set statement delimiter.
edit       (\e) Edit command with $EDITOR.
ego        (\G) Send command to mysql server, display result vertically.
exit       (\q) Exit mysql. Same as quit.
go         (\g) Send command to mysql server.
help       (\h) Display this help.
nopager    (\n) Disable pager, print to stdout.
notee      (\t) Don't write into outfile.
pager      (\P) Set PAGER [to_pager]. Print the query results via PAGER.
print      (\p) Print current command.
prompt     (\R) Change your mysql prompt.
quit       (\q) Quit mysql.
rehash     (\#) Rebuild completion hash.
source     (\.) Execute an SQL script file. Takes a file name as an argument.
status     (\s) Get status information from the server.
system     (\!) Execute a system shell command.
tee        (\T) Set outfile [to_outfile]. Append everything into given
               outfile.
use        (\u) Use another database. Takes database name as argument.
charset    (\C) Switch to another charset. Might be needed for processing
               binlog with multi-byte charsets.
warnings  (\W) Show warnings after every statement.
nowarning (\w) Don't show warnings after every statement.
resetconnection(\x) Clean session context.

For server side help, type 'help contents'
```

If mysql is invoked with the --binary-mode option, all mysql commands are disabled except charset and delimiter in non-interactive mode (for input piped to mysql or loaded using the source command).

Each command has both a long and short form. The long form is not case sensitive; the short form is. The long form can be followed by an optional semicolon terminator, but the short form should not.

The use of short-form commands within multi-line /* ... */ comments is not supported.

- help [*arg*], \h [*arg*], \? [*arg*], ? [*arg*]

  Display a help message listing the available mysql commands.

  If you provide an argument to the help command, mysql uses it as a search string to access server-side help from the contents of the MySQL Reference Manual. For more information, see Section 4.5.1.4, "mysql Server-Side Help".

- charset *charset_name*, \C *charset_name*

  Change the default character set and issue a SET NAMES statement. This enables the character set to remain synchronized on the client and server if mysql is run with auto-reconnect enabled (which is not recommended), because the specified character set is used for reconnects.

- clear, \c

  Clear the current input. Use this if you change your mind about executing the statement that you are entering.

- connect [*db_name host_name*]], \r [*db_name host_name*]]

  Reconnect to the server. The optional database name and host name arguments may be given to specify the default database or the host where the server is running. If omitted, the current values are used.

- delimiter *str*, \d *str*

  Change the string that mysql interprets as the separator between SQL statements. The default is the semicolon character (";").

  The delimiter string can be specified as an unquoted or quoted argument on the delimiter command line. Quoting can be done with either single quote ('), double quote ("), or backtick (`) characters. To include a quote within a quoted string, either quote the string with a different quote character or escape the quote with a backslash ("\") character. Backslash should be avoided outside of quoted strings because it is the escape character for MySQL. For an unquoted argument, the delimiter is read up to the first space or end of line. For a quoted argument, the delimiter is read up to the matching quote on the line.

  mysql interprets instances of the delimiter string as a statement delimiter anywhere it occurs, except within quoted strings. Be careful about defining a delimiter that might occur within other words. For example, if you define the delimiter as X, you will be unable to use the word INDEX in statements. mysql interprets this as INDE followed by the delimiter X.

  When the delimiter recognized by mysql is set to something other than the default of ";", instances of that character are sent to the server without interpretation. However, the server itself still interprets ";" as a statement delimiter and processes statements accordingly. This behavior on the server side comes into play for multiple-statement execution (see Section 21.8.17, "C API Support for Multiple Statement Execution"), and for parsing the body of stored procedures and functions, triggers, and events (see Section 18.1, "Defining Stored Programs").

- edit, \e

  Edit the current input statement. mysql checks the values of the EDITOR and VISUAL environment variables to determine which editor to use. The default editor is vi if neither variable is set.

  The edit command works only in Unix.

- ego, \G

  Send the current statement to the server to be executed and display the result using vertical format.

- exit, \q

  Exit mysql.

- go, \g

  Send the current statement to the server to be executed.

- nopager, \n

  Disable output paging. See the description for pager.

  The nopager command works only in Unix.

- notee, \t

  Disable output copying to the tee file. See the description for tee.

- nowarning, \w

  Enable display of warnings after each statement.

- pager [*command*], \P [*command*]

  Enable output paging. By using the --pager option when you invoke mysql, it is possible to browse or search query results in interactive mode with Unix programs such as less, more, or any other similar program. If you specify no value for the option, mysql checks the value of the PAGER environment variable and sets the pager to that. Pager functionality works only in interactive mode.

  Output paging can be enabled interactively with the pager command and disabled with nopager. The command takes an optional argument; if given, the paging program is set to that. With no argument, the pager is set to the pager that was set on the command line, or stdout if no pager was specified.

  Output paging works only in Unix because it uses the popen() function, which does not exist on Windows. For Windows, the tee option can be used instead to save query output, although it is not as convenient as pager for browsing output in some situations.

- print, \p

  Print the current input statement without executing it.

- prompt [*str*], \R [*str*]

  Reconfigure the mysql prompt to the given string. The special character sequences that can be used in the prompt are described later in this section.

  If you specify the prompt command with no argument, mysql resets the prompt to the default of mysql>.

- quit, \q

  Exit mysql.

- rehash, \#

  Rebuild the completion hash that enables database, table, and column name completion while you are entering statements. (See the description for the --auto-rehash option.)

- resetconnection, \x

  Reset the connection to clear the session state. This command was added in MySQL 5.7.3.

  Resetting a connection has effects similar to mysql_change_user() or an auto-reconnect except that the connection is not closed and reopened, and re-authentication is not done. See Section 21.8.7.3, “mysql_change_user()”) and see Section 21.8.16, “Controlling Automatic Reconnection Behavior”).

  This example shows how resetconnection clears a value maintained in the session state:

```
mysql> SELECT LAST_INSERT_ID(3);
+-------------------+
| LAST_INSERT_ID(3) |
+-------------------+
|                 3 |
+-------------------+

mysql> SELECT LAST_INSERT_ID();
+------------------+
| LAST_INSERT_ID() |
+------------------+
|                3 |
+------------------+

mysql> resetconnection;

mysql> SELECT LAST_INSERT_ID();
+------------------+
| LAST_INSERT_ID() |
+------------------+
|                0 |
+------------------+
```

- source *file_name*, \. *file_name*

  Read the named file and executes the statements contained therein. On Windows, you can specify path name separators as / or \\.

- status, \s

  Provide status information about the connection and the server you are using. If you are running in --safe-updates mode, status also prints the values for the mysql variables that affect your queries.

- system *command*, \! *command*

  Execute the given command using your default command interpreter.

  The system command works only in Unix.

- tee [*file_name*], \T [*file_name*]

By using the `--tee` option when you invoke `mysql`, you can log statements and their output. All the data displayed on the screen is appended into a given file. This can be very useful for debugging purposes also. `mysql` flushes results to the file after each statement, just before it prints its next prompt. Tee functionality works only in interactive mode.

You can enable this feature interactively with the `tee` command. Without a parameter, the previous file is used. The `tee` file can be disabled with the `notee` command. Executing `tee` again re-enables logging.

- `use` *`db_name`*, `\u` *`db_name`*

  Use *`db_name`* as the default database.

- `warnings`, `\W`

  Enable display of warnings after each statement (if there are any).

Here are a few tips about the `pager` command:

- You can use it to write to a file and the results go only to the file:

  ```
  mysql> pager cat > /tmp/log.txt
  ```

  You can also pass any options for the program that you want to use as your pager:

  ```
  mysql> pager less -n -i -S
  ```

- In the preceding example, note the `-S` option. You may find it very useful for browsing wide query results. Sometimes a very wide result set is difficult to read on the screen. The `-S` option to `less` can make the result set much more readable because you can scroll it horizontally using the left-arrow and right-arrow keys. You can also use `-S` interactively within `less` to switch the horizontal-browse mode on and off. For more information, read the `less` manual page:

  ```
  shell> man less
  ```

- The `-F` and `-X` options may be used with `less` to cause it to exit if output fits on one screen, which is convenient when no scrolling is necessary:

  ```
  mysql> pager less -n -i -S -F -X
  ```

- You can specify very complex pager commands for handling query output:

  ```
  mysql> pager cat | tee /dr1/tmp/res.txt \
            | tee /dr2/tmp/res2.txt | less -n -i -S
  ```

  In this example, the command would send query results to two files in two different directories on two different file systems mounted on `/dr1` and `/dr2`, yet still display the results onscreen using `less`.

You can also combine the `tee` and `pager` functions. Have a `tee` file enabled and `pager` set to `less`, and you are able to browse the results using the `less` program and still have everything appended into a file the same time. The difference between the Unix `tee` used with the `pager` command and the `mysql` built-in `tee` command is that the built-in `tee` works even if you do not have the Unix `tee` available. The built-in `tee` also logs everything that is printed on the screen, whereas the Unix `tee` used with `pager` does not log quite that much. Additionally, `tee` file logging can be turned on and off interactively from within `mysql`. This is useful when you want to log some queries to a file, but not others.

The prompt command reconfigures the default mysql> prompt. The string for defining the prompt can contain the following special sequences.

Option	Description
\c	A counter that increments for each statement you issue
\D	The full current date
\d	The default database
\h	The server host
\l	The current delimiter
\m	Minutes of the current time
\n	A newline character
\O	The current month in three-letter format (Jan, Feb, …)
\o	The current month in numeric format
\P	am/pm
\p	The current TCP/IP port or socket file
\R	The current time, in 24-hour military time (0–23)
\r	The current time, standard 12-hour time (1–12)
\S	Semicolon
\s	Seconds of the current time
\t	A tab character
\U	Your full *user_name@host_name* account name
\u	Your user name
\v	The server version
\w	The current day of the week in three-letter format (Mon, Tue, …)
\Y	The current year, four digits
\y	The current year, two digits
_	A space
\	A space (a space follows the backslash)
\'	Single quote
\"	Double quote
\\	A literal “\” backslash character
\x	*x*, for any “*x*” not listed above

You can set the prompt in several ways:

- *Use an environment variable.* You can set the MYSQL_PS1 environment variable to a prompt string. For example:

```
shell> export MYSQL_PS1="(\u@\h) [\d]> "
```

- *Use a command-line option.* You can set the --prompt option on the command line to mysql. For example:

```
shell> mysql --prompt="(\u@\h) [\d]> "
(user@host) [database]>
```

- *Use an option file.* You can set the prompt option in the [mysql] group of any MySQL option file, such as /etc/my.cnf or the .my.cnf file in your home directory. For example:

  ```
  [mysql]
  prompt=(\\u@\\h) [\\d]>\\_
  ```

  In this example, note that the backslashes are doubled. If you set the prompt using the prompt option in an option file, it is advisable to double the backslashes when using the special prompt options. There is some overlap in the set of permissible prompt options and the set of special escape sequences that are recognized in option files. (The rules for escape sequences in option files are listed in Section 4.2.3.3, “Using Option Files”.) The overlap may cause you problems if you use single backslashes. For example, \s is interpreted as a space rather than as the current seconds value. The following example shows how to define a prompt within an option file to include the current time in HH:MM:SS> format:

  ```
  [mysql]
  prompt="\\r:\\m:\\s> "
  ```

- *Set the prompt interactively.* You can change your prompt interactively by using the prompt (or \R) command. For example:

  ```
  mysql> prompt (\u@\h) [\d]>\_
  PROMPT set to '(\u@\h) [\d]>\_'
  (user@host) [database]>
  (user@host) [database]> prompt
  Returning to default PROMPT of mysql>
  mysql>
  ```

4.5.1.3 mysql Logging

The mysql client can do these types of logging for statements executed interactively:

- On Unix, mysql writes the statements to a history file. By default, this file is named .mysql_history in your home directory. To specify a different file, set the value of the MYSQL_HISTFILE environment variable.
- On all platforms, if the --syslog option is given, mysql writes the statements to the system logging facility. On Unix, this is syslog; on Windows, it is the Windows Event Log. The destination where logged messages appear is system dependent. On Linux, the destination is often the /var/log/messages file.

The following discussion describes characteristics that apply to all logging types and provides information specific to each logging type.

How Logging Occurs

For each enabled logging destination, statement logging occurs as follows:

- Statements are logged only when executed interactively. Statements are noninteractive, for example, when read from a file or a pipe. It is also possible to suppress statement logging by using the --batch or --execute option.
- Statements are ignored and not logged if they match any pattern in the “ignore” list. This list is described later.

- mysql logs each nonignored, nonempty statement line individually.
- If a nonignored statement spans multiple lines (not including the terminating delimiter), mysql concatenates the lines to form the complete statement, maps newlines to spaces, and logs the result, plus a delimiter.

Consequently, an input statement that spans multiple lines can be logged twice. Consider this input:

```
mysql> SELECT
    -> 'Today is'
    -> ,
    -> CONCAT()
    -> ;
```

In this case, mysql logs the “SELECT”, “'Today is'”, “,”, “CONCAT()”, and “;” lines as it reads them. It also logs the complete statement, after mapping `SELECT\n'Today is'\n,\nCURDATE()` to `SELECT 'Today is' , CURDATE()`, plus a delimiter. Thus, these lines appear in logged output:

```
SELECT
'Today is'
,
CURDATE()
;
SELECT 'Today is' , CURDATE();
```

mysql ignores for logging purposes statements that match any pattern in the “ignore” list. By default, the pattern list is `"*IDENTIFIED*:*PASSWORD*"`, to ignore statements that refer to passwords. Pattern matching is not case sensitive. Within patterns, two characters are special:

- ? matches any single character.
- * matches any sequence of zero or more characters.

To specify additional patterns, use the `--histignore` option or set the `MYSQL_HISTIGNORE` environment variable. (If both are specified, the option value takes precedence.) The value should be a colon-separated list of one or more patterns, which are appended to the default pattern list.

Patterns specified on the command line might need to be quoted or escaped to prevent your command interpreter from treating them specially. For example, to suppress logging for `UPDATE` and `DELETE` statements in addition to statements that refer to passwords, invoke mysql like this:

```
shell> mysql --histignore="*UPDATE*:*DELETE*"
```

Controlling the History File

The `.mysql_history` file should be protected with a restrictive access mode because sensitive information might be written to it, such as the text of SQL statements that contain passwords. See Section 6.1.2.1, “End-User Guidelines for Password Security”.

If you do not want to maintain a history file, first remove `.mysql_history` if it exists. Then use either of the following techniques to prevent it from being created again:

- Set the `MYSQL_HISTFILE` environment variable to `/dev/null`. To cause this setting to take effect each time you log in, put it in one of your shell's startup files.
- Create `.mysql_history` as a symbolic link to `/dev/null`; this need be done only once:

```
shell> ln -s /dev/null $HOME/.mysql_history
```

syslog Logging Characteristics

If the --syslog option is given, mysql writes interactive statements to the system logging facility. Message logging has the following characteristics.

Logging occurs at the “information” level. This corresponds to the LOG_INFO priority for syslog on Unix/Linux syslog capability and to EVENTLOG_INFORMATION_TYPE for the Windows Event Log. Consult your system documentation for configuration of your logging capability.

Message size is limited to 1024 bytes.

Messages consist of the identifier MysqlClient followed by these values:

- SYSTEM_USER

  The system user name (login name) or -- if the user is unknown.

- MYSQL_USER

  The MySQL user name (specified with the --user option) or -- if the user is unknown.

- CONNECTION_ID:

  The client connection identifier. This is the same as the CONNECTION_ID() function value within the session.

- DB_SERVER

  The server host or -- if the host is unknown.

- DB

  The default database or -- if no database has been selected.

- QUERY

  The text of the logged statement.

Here is a sample of output generated on Linux by using --syslog. This output is formatted for readability; each logged message actually takes a single line.

```
Mar  7 12:39:25 myhost MysqlClient[20824]:
  SYSTEM_USER:'oscar', MYSQL_USER:'my_oscar', CONNECTION_ID:23,
  DB_SERVER:'127.0.0.1', DB:'--', QUERY:'USE test;'
Mar  7 12:39:28 myhost MysqlClient[20824]:
  SYSTEM_USER:'oscar', MYSQL_USER:'my_oscar', CONNECTION_ID:23,
  DB_SERVER:'127.0.0.1', DB:'test', QUERY:'SHOW TABLES;'
```

4.5.1.4 mysql Server-Side Help

```
mysql> help search_string
```

If you provide an argument to the help command, mysql uses it as a search string to access server-side help from the contents of the MySQL Reference Manual. The proper operation of this command requires that the help tables in the mysql database be initialized with help topic information (see Section 5.1.10, “Server-Side Help”).

If there is no match for the search string, the search fails:

```
mysql> help me

Nothing found
Please try to run 'help contents' for a list of all accessible topics
```

Use help contents to see a list of the help categories:

```
mysql> help contents
You asked for help about help category: "Contents"
For more information, type 'help <item>', where <item> is one of the
following categories:
   Account Management
   Administration
   Data Definition
   Data Manipulation
   Data Types
   Functions
   Functions and Modifiers for Use with GROUP BY
   Geographic Features
   Language Structure
   Plugins
   Storage Engines
   Stored Routines
   Table Maintenance
   Transactions
   Triggers
```

If the search string matches multiple items, mysql shows a list of matching topics:

```
mysql> help logs
Many help items for your request exist.
To make a more specific request, please type 'help <item>',
where <item> is one of the following topics:
   SHOW
   SHOW BINARY LOGS
   SHOW ENGINE
   SHOW LOGS
```

Use a topic as the search string to see the help entry for that topic:

```
mysql> help show binary logs
Name: 'SHOW BINARY LOGS'
Description:
Syntax:
SHOW BINARY LOGS
SHOW MASTER LOGS

Lists the binary log files on the server. This statement is used as
part of the procedure described in [purge-binary-logs], that shows how
to determine which logs can be purged.

mysql> SHOW BINARY LOGS;
+---------------+-----------+
| Log_name      | File_size |
+---------------+-----------+
| binlog.000015 |    724935 |
| binlog.000016 |    733481 |
+---------------+-----------+
```

The search string can contain the the wildcard characters "%" and "_". These have the same meaning as for pattern-matching operations performed with the LIKE operator. For example, HELP rep% returns a list of topics that begin with rep:

```
mysql> HELP rep%
Many help items for your request exist.
To make a more specific request, please type 'help <item>',
where <item> is one of the following
topics:
   REPAIR TABLE
   REPEAT FUNCTION
   REPEAT LOOP
   REPLACE
   REPLACE FUNCTION
```

4.5.1.5 Executing SQL Statements from a Text File

The mysql client typically is used interactively, like this:

```
shell> mysql db_name
```

However, it is also possible to put your SQL statements in a file and then tell mysql to read its input from that file. To do so, create a text file *text_file* that contains the statements you wish to execute. Then invoke mysql as shown here:

```
shell> mysql db_name < text_file
```

If you place a USE *db_name* statement as the first statement in the file, it is unnecessary to specify the database name on the command line:

```
shell> mysql < text_file
```

If you are already running mysql, you can execute an SQL script file using the source command or \. command:

```
mysql> source file_name
mysql> \. file_name
```

Sometimes you may want your script to display progress information to the user. For this you can insert statements like this:

```
SELECT '<info_to_display>' AS ' ';
```

The statement shown outputs <info_to_display>.

You can also invoke mysql with the --verbose option, which causes each statement to be displayed before the result that it produces.

mysql ignores Unicode byte order mark (BOM) characters at the beginning of input files. Previously, it read them and sent them to the server, resulting in a syntax error. Presence of a BOM does not cause mysql to change its default character set. To do that, invoke mysql with an option such as --default-character-set=utf8.

For more information about batch mode, see Section 3.5, "Using mysql in Batch Mode".

4.5.1.6 mysql Tips

This section describes some techniques that can help you use mysql more effectively.

Input-Line Editing

mysql supports input-line editing, which enables you to modify the current input line in place or recall previous input lines. For example, the **left-arrow** and **right-arrow** keys move horizontally within the current input line, and the **up-arror** and **down-arrow** keys move up and down through the set of previously entered lines. **Backspace** deletes the character before the cursor and typing new characters enters them at the cursor position. To enter the line, press **Enter**.

On Windows, the editing key sequences are the same as supported for command editing in console windows. On Unix, the key sequences depend on the input library used to build mysql (for example, the libedit or readline library).

Documentation for the libedit and readline libraries is available online. To change the set of key sequences permitted by a given input library, define key bindings in the library startup file. This is a file in your home directory: .editrc for libedit and .inputrc for readline.

For example, in libedit, **Control+W** deletes everything before the current cursor position and **Control+U** deletes the entire line. In readline, **Control+W** deletes the word before the cursor and **Control+U** deletes everything before the current cursor position. If mysql was built using libedit, a user who prefers the readline behavior for these two keys can put the following lines in the .editrc file (creating the file if necessary):

```
bind "^W" ed-delete-prev-word
bind "^U" vi-kill-line-prev
```

To see the current set of key bindings, temporarily put a line that says only bind at the end of .editrc. mysql will show the bindings when it starts.

Unicode Support on Windows

Windows provides APIs based on UTF-16LE for reading from and writing to the console; the mysql client for Windows is able to use these APIs. The Windows installer creates an item in the MySQL menu named MySQL command line client - Unicode. This item invokes the mysql client with properties set to communicate through the console to the MySQL server using Unicode.

To take advantage of this support manually, run mysql within a console that uses a compatible Unicode font and set the default character set to a Unicode character set that is supported for communication with the server:

1. Open a console window.

2. Go to the console window properties, select the font tab, and choose Lucida Console or some other compatible Unicode font. This is necessary because console windows start by default using a DOS raster font that is inadequate for Unicode.

3. Execute mysql.exe with the --default-character-set=utf8 (or utf8mb4) option. This option is necessary because utf16le is not supported as a connection character set.

With those changes, mysql will use the Windows APIs to communicate with the console using UTF-16LE, and communicate with the server using UTF-8. (The menu item mentioned previously sets the font and character set as just described.)

To avoid those steps each time you run mysql, you can create a shortcut that invokes mysql.exe. The shortcut should set the console font to Lucida Console or some other compatible Unicode font, and pass the --default-character-set=utf8 (or utf8mb4) option to mysql.exe.

Alternatively, create a shortcut that only sets the console font, and set the character set in the [mysql] group of your my.ini file:

```
[mysql]
default-character-set=utf8
```

Displaying Query Results Vertically

Some query results are much more readable when displayed vertically, instead of in the usual horizontal table format. Queries can be displayed vertically by terminating the query with \G instead of a semicolon. For example, longer text values that include newlines often are much easier to read with vertical output:

```
mysql> SELECT * FROM mails WHERE LENGTH(txt) < 300 LIMIT 300,1\G
*************************** 1. row ***************************
  msg_nro: 3068
     date: 2000-03-01 23:29:50
time_zone: +0200
mail_from: Monty
    reply: monty@no.spam.com
  mail_to: "Thimble Smith" <tim@no.spam.com>
      sbj: UTF-8
      txt: >>>>> "Thimble" == Thimble Smith writes:

Thimble> Hi.  I think this is a good idea.  Is anyone familiar
Thimble> with UTF-8 or Unicode? Otherwise, I'll put this on my
Thimble> TODO list and see what happens.

Yes, please do that.

Regards,
Monty
     file: inbox-jani-1
     hash: 190402944
1 row in set (0.09 sec)
```

Using the --safe-updates Option

For beginners, a useful startup option is --safe-updates (or --i-am-a-dummy, which has the same effect). It is helpful for cases when you might have issued a DELETE FROM *tbl_name* statement but forgotten the WHERE clause. Normally, such a statement deletes all rows from the table. With --safe-updates, you can delete rows only by specifying the key values that identify them. This helps prevent accidents.

When you use the --safe-updates option, mysql issues the following statement when it connects to the MySQL server:

```
SET sql_safe_updates=1, sql_select_limit=1000, max_join_size=1000000;
```

See Section 5.1.4, “Server System Variables”.

The SET statement has the following effects:

- You are not permitted to execute an UPDATE or DELETE statement unless you specify a key constraint in the WHERE clause or provide a LIMIT clause (or both). For example:

```
UPDATE tbl_name SET not_key_column=val WHERE key_column=val;

UPDATE tbl_name SET not_key_column=val LIMIT 1;
```

- The server limits all large SELECT results to 1,000 rows unless the statement includes a LIMIT clause.
- The server aborts multiple-table SELECT statements that probably need to examine more than 1,000,000 row combinations.

To specify limits different from 1,000 and 1,000,000, you can override the defaults by using the --select_limit and --max_join_size options:

```
shell> mysql --safe-updates --select_limit=500 --max_join_size=10000
```

Disabling mysql Auto-Reconnect

If the mysql client loses its connection to the server while sending a statement, it immediately and automatically tries to reconnect once to the server and send the statement again. However, even if mysql succeeds in reconnecting, your first connection has ended and all your previous session objects and settings are lost: temporary tables, the autocommit mode, and user-defined and session variables. Also, any current transaction rolls back. This behavior may be dangerous for you, as in the following example where the server was shut down and restarted between the first and second statements without you knowing it:

```
mysql> SET @a=1;
Query OK, 0 rows affected (0.05 sec)

mysql> INSERT INTO t VALUES(@a);
ERROR 2006: MySQL server has gone away
No connection. Trying to reconnect...
Connection id:    1
Current database: test

Query OK, 1 row affected (1.30 sec)

mysql> SELECT * FROM t;
+------+
| a    |
+------+
| NULL |
+------+
1 row in set (0.05 sec)
```

The @a user variable has been lost with the connection, and after the reconnection it is undefined. If it is important to have mysql terminate with an error if the connection has been lost, you can start the mysql client with the --skip-reconnect option.

For more information about auto-reconnect and its effect on state information when a reconnection occurs, see Section 21.8.16, “Controlling Automatic Reconnection Behavior”.

4.5.2 mysqladmin — Client for Administering a MySQL Server

mysqladmin is a client for performing administrative operations. You can use it to check the server's configuration and current status, to create and drop databases, and more.

Invoke mysqladmin like this:

```
shell> mysqladmin [options] command [command-arg] [command [command-arg]] ...
```

mysqladmin supports the following commands. Some of the commands take an argument following the command name.

- create *db_name*

  Create a new database named *db_name*.

- debug

  Tell the server to write debug information to the error log. Format and content of this information is subject to change.

  This includes information about the Event Scheduler. See Section 18.4.5, “Event Scheduler Status”.

- drop *db_name*

  Delete the database named *db_name* and all its tables.

- extended-status

  Display the server status variables and their values.

- flush-hosts

  Flush all information in the host cache.

- flush-logs

  Flush all logs.

- flush-privileges

  Reload the grant tables (same as reload).

- flush-status

  Clear status variables.

- flush-tables

  Flush all tables.

- flush-threads

  Flush the thread cache.

- kill *id,id,...*

  Kill server threads. If multiple thread ID values are given, there must be no spaces in the list.

- old-password *new-password*

  This is like the password command but stores the password using the old (pre-4.1) password-hashing format. (See Section 6.1.2.4, “Password Hashing in MySQL”.)

- password *new-password*

  Set a new password. This changes the password to *new-password* for the account that you use with mysqladmin for connecting to the server. Thus, the next time you invoke mysqladmin (or any other client program) using the same account, you will need to specify the new password.

If the *new-password* value contains spaces or other characters that are special to your command interpreter, you need to enclose it within quotation marks. On Windows, be sure to use double quotation marks rather than single quotation marks; single quotation marks are not stripped from the password, but rather are interpreted as part of the password. For example:

```
shell> mysqladmin password "my new password"
```

In MySQL 5.7, the new password can be omitted following the password command. In this case, mysqladmin prompts for the password value, which enables you to avoid specifying the password on the command line. Omitting the password value should be done only if password is the final command on the mysqladmin command line. Otherwise, the next argument is taken as the password.

Caution

Do not use this command used if the server was started with the --skip-grant-tables option. No password change will be applied. This is true even if you precede the password command with flush-privileges on the same command line to re-enable the grant tables because the flush operation occurs after you connect. However, you can use mysqladmin flush-privileges to re-enable the grant table and then use a separate mysqladmin password command to change the password.

- ping

  Check whether the server is available. The return status from mysqladmin is 0 if the server is running, 1 if it is not. This is 0 even in case of an error such as Access denied, because this means that the server is running but refused the connection, which is different from the server not running.

- processlist

  Show a list of active server threads. This is like the output of the SHOW PROCESSLIST statement. If the --verbose option is given, the output is like that of SHOW FULL PROCESSLIST. (See Section 13.7.5.28, “SHOW PROCESSLIST Syntax”.)

- reload

  Reload the grant tables.

- refresh

  Flush all tables and close and open log files.

- shutdown

  Stop the server.

- start-slave

  Start replication on a slave server.

- status

  Display a short server status message.

- stop-slave

  Stop replication on a slave server.

- variables

  Display the server system variables and their values.

- version

  Display version information from the server.

All commands can be shortened to any unique prefix. For example:

```
shell> mysqladmin proc stat
+----+-------+-----------+----+---------+------+-------+------------------+
| Id | User  | Host      | db | Command | Time | State | Info             |
+----+-------+-----------+----+---------+------+-------+------------------+
| 51 | monty | localhost |    | Query   | 0    |       | show processlist |
+----+-------+-----------+----+---------+------+-------+------------------+
Uptime: 1473624  Threads: 1  Questions: 39487
Slow queries: 0  Opens: 541  Flush tables: 1
Open tables: 19  Queries per second avg: 0.0268
```

The mysqladmin status command result displays the following values:

- Uptime

  The number of seconds the MySQL server has been running.

- Threads

  The number of active threads (clients).

- Questions

  The number of questions (queries) from clients since the server was started.

- Slow queries

  The number of queries that have taken more than long_query_time seconds. See Section 5.2.5, “The Slow Query Log”.

- Opens

  The number of tables the server has opened.

- Flush tables

  The number of flush-*, refresh, and reload commands the server has executed.

- Open tables

  The number of tables that currently are open.

If you execute mysqladmin shutdown when connecting to a local server using a Unix socket file, mysqladmin waits until the server's process ID file has been removed, to ensure that the server has stopped properly.

mysqladmin supports the following options, which can be specified on the command line or in the [mysqladmin] and [client] groups of an option file. For information about option files, see Section 4.2.3.3, “Using Option Files”.

Table 4.7 mysqladmin Options

Format	Option File	Description	Introduced
--bind-address=ip_address	bind-address	Use the specified network interface to connect to the MySQL Server	
--compress	compress	Compress all information sent between the client and the server	
--connect_timeout=seconds	connect_timeout	The number of seconds before connection timeout	
--count=#	count	The number of iterations to make for repeated command execution	
--debug[=debug_options]	debug	Write a debugging log	
--debug-check	debug-check	Print debugging information when the program exits	
--debug-info	debug-info	Print debugging information, memory and CPU statistics when the program exits	
--default-auth=plugin	default-auth=plugin	The authentication plugin to use	
--default-character-set=charset_name	default-character-set	Use charset_name as the default character set	
--defaults-extra-file=file_name		Read option file in addition to the usual option files	
--defaults-file=file_name		Read only the given option file	
--defaults-group-suffix=str		Option group suffix value	
--enable-cleartext-plugin	enable-cleartext-plugin	Enable cleartext authentication plugin	
--force	force	Continue even if an SQL error occurs	
--help		Display help message and exit	
--host=host_name	host	Connect to the MySQL server on the given host	
--login-path=name		Read login path options from .mylogin.cnf	
--no-beep	no-beep	Do not beep when errors occur	
--no-defaults		Do not read any option files	
--password[=password]	password	The password to use when connecting to the server	
--pipe		On Windows, connect to server using a named pipe	
--plugin-dir=path	plugin-dir=path	The directory where plugins are located	
--port=port_num	port	The TCP/IP port number to use for the connection	
--print-defaults		Print defaults	
--protocol=type	protocol	The connection protocol to use	
--relative	relative	Show the difference between the current and previous values when used with the --sleep option	

Format	Option File	Description	Introduced
--secure-auth	secure-auth	Do not send passwords to the server in old (pre-4.1.1) format	5.7.4
--show-warnings	show-warnings	Show warnings after statement execution	5.7.2
--shutdown_timeout=seconds	shutdown_timeout	The maximum number of seconds to wait for server shutdown	
--silent	silent	Silent mode	
--sleep=delay	sleep	Execute commands repeatedly, sleeping for delay seconds in between	
--socket=path	socket	For connections to localhost	
--ssl-ca=file_name	ssl-ca	The path to a file that contains a list of trusted SSL CAs	
--ssl-capath=dir_name	ssl-capath	The path to a directory that contains trusted SSL CA certificates in PEM format	
--ssl-cert=file_name	ssl-cert	The name of the SSL certificate file to use for establishing a secure connection	
--ssl-cipher=cipher_list	ssl-cipher	A list of allowable ciphers to use for SSL encryption	
--ssl-crl=file_name	ssl-crl	The path to a file that contains certificate revocation lists	
--ssl-crlpath=dir_name	ssl-crlpath	The path to a directory that contains certificate revocation list files	
--ssl-key=file_name	ssl-key	The name of the SSL key file to use for establishing a secure connection	
--ssl-verify-server-cert	ssl-verify-server-cert	The server's Common Name value in its certificate is verified against the host name used when connecting to the server	
--user=user_name,	user	MySQL user name to use when connecting to server	
--verbose		Verbose mode	
--version		Display version information and exit	
--vertical	vertical	Print query output rows vertically (one line per column value)	
--wait	wait	If the connection cannot be established, wait and retry instead of aborting	

- `--help`, `-?`

  Display a help message and exit.

- `--bind-address=ip_address`

  On a computer having multiple network interfaces, this option can be used to select which interface is employed when connecting to the MySQL server.

- `--character-sets-dir=path`

  The directory where character sets are installed. See Section 10.5, “Character Set Configuration”.

- `--compress`, `-C`

  Compress all information sent between the client and the server if both support compression.

- `--count=N`, `-c N`

  The number of iterations to make for repeated command execution if the `--sleep` option is given.

- `--debug[=debug_options]`, `-# [debug_options]`

  Write a debugging log. A typical *`debug_options`* string is `d:t:o,file_name`. The default is `d:t:o,/tmp/mysqladmin.trace`.

- `--debug-check`

  Print some debugging information when the program exits.

- `--debug-info`

  Print debugging information and memory and CPU usage statistics when the program exits.

- `--default-auth=plugin`

  The client-side authentication plugin to use. See Section 6.3.8, “Pluggable Authentication”.

- `--default-character-set=charset_name`

  Use *`charset_name`* as the default character set. See Section 10.5, “Character Set Configuration”.

- `--defaults-extra-file=file_name`

  Read this option file after the global option file but (on Unix) before the user option file. If the file does not exist or is otherwise inaccessible, an error occurs. *`file_name`* is interpreted relative to the current directory if given as a relative path name rather than a full path name.

- `--defaults-file=file_name`

  Use only the given option file. If the file does not exist or is otherwise inaccessible, an error occurs. *`file_name`* is interpreted relative to the current directory if given as a relative path name rather than a full path name.

- `--defaults-group-suffix=str`

  Read not only the usual option groups, but also groups with the usual names and a suffix of *`str`*. For example, `mysqladmin` normally reads the `[client]` and `[mysqladmin]` groups. If the `--defaults-group-suffix=_other` option is given, `mysqladmin` also reads the `[client_other]` and `[mysqladmin_other]` groups.

- `--enable-cleartext-plugin`

  Enable the `mysql_clear_password` cleartext authentication plugin. (See Section 6.3.9.5, “The Cleartext Client-Side Authentication Plugin”.)

- `--force`, `-f`

  Do not ask for confirmation for the `drop db_name` command. With multiple commands, continue even if an error occurs.

- `--host=host_name`, `-h host_name`

Connect to the MySQL server on the given host.

- `--login-path=name`

  Read options from the named login path in the `.mylogin.cnf` login file. A “login path” is an option group that permits only a limited set of options: `host`, `user`, and `password`. Think of a login path as a set of values that indicate the server host and the credentials for authenticating with the server. To create the login file, use the `mysql_config_editor` utility. See Section 4.6.6, “`mysql_config_editor` — MySQL Configuration Utility”.

- `--no-beep`, `-b`

  Suppress the warning beep that is emitted by default for errors such as a failure to connect to the server.

- `--no-defaults`

  Do not read any option files. If program startup fails due to reading unknown options from an option file, `--no-defaults` can be used to prevent them from being read.

  The exception is that the `.mylogin.cnf` file, if it exists, is read in all cases. This permits passwords to be specified in a safer way than on the command line even when `--no-defaults` is used. (`.mylogin.cnf` is created by the `mysql_config_editor` utility. See Section 4.6.6, “`mysql_config_editor` — MySQL Configuration Utility”.)

- `--password[=password]`, `-p[password]`

  The password to use when connecting to the server. If you use the short option form (`-p`), you *cannot* have a space between the option and the password. If you omit the *`password`* value following the `--password` or `-p` option on the command line, `mysqladmin` prompts for one.

  Specifying a password on the command line should be considered insecure. See Section 6.1.2.1, “End-User Guidelines for Password Security”. You can use an option file to avoid giving the password on the command line.

- `--pipe`, `-W`

  On Windows, connect to the server using a named pipe. This option applies only if the server supports named-pipe connections.

- `--plugin-dir=path`

  The directory in which to look for plugins. It may be necessary to specify this option if the `--default-auth` option is used to specify an authentication plugin but `mysqladmin` does not find it. See Section 6.3.8, “Pluggable Authentication”.

- `--port=port_num`, `-P port_num`

  The TCP/IP port number to use for the connection.

- `--print-defaults`

  Print the program name and all options that it gets from option files.

- `--protocol={TCP|SOCKET|PIPE|MEMORY}`

  The connection protocol to use for connecting to the server. It is useful when the other connection parameters normally would cause a protocol to be used other than the one you want. For details on the permissible values, see Section 4.2.2, “Connecting to the MySQL Server”.

- `--relative`, `-r`

  Show the difference between the current and previous values when used with the `--sleep` option. This option works only with the `extended-status` command.

- `--show-warnings`

  Show warnings resulting from execution of statements sent to the server. This option was added in MySQL 5.7.2.

- `--secure-auth`

  Do not send passwords to the server in old (pre-4.1) format. This prevents connections except for servers that use the newer password format. This option is enabled by default; use `--skip-secure-auth` to disable it. This option was added in MySQL 5.7.4.

  Note

  Passwords that use the pre-4.1 hashing method are less secure than passwords that use the native password hashing method and should be avoided. Pre-4.1 passwords are deprecated and support for them will be removed in a future MySQL release. For account upgrade instructions, see Section 6.3.9.3, “Migrating Away from Pre-4.1 Password Hashing and the `mysql_old_password` Plugin”.

- `--silent`, `-s`

  Exit silently if a connection to the server cannot be established.

- `--sleep=`*`delay`*, `-i `*`delay`*

  Execute commands repeatedly, sleeping for *`delay`* seconds in between. The `--count` option determines the number of iterations. If `--count` is not given, `mysqladmin` executes commands indefinitely until interrupted.

- `--socket=`*`path`*, `-S `*`path`*

  For connections to `localhost`, the Unix socket file to use, or, on Windows, the name of the named pipe to use.

- `--ssl*`

  Options that begin with `--ssl` specify whether to connect to the server using SSL and indicate where to find SSL keys and certificates. See Section 6.3.11.4, “SSL Command Options”.

- `--user=`*`user_name`*, `-u `*`user_name`*

  The MySQL user name to use when connecting to the server.

- `--verbose`, `-v`

  Verbose mode. Print more information about what the program does.

- `--version`, `-V`

  Display version information and exit.

- `--vertical`, `-E`

  Print output vertically. This is similar to `--relative`, but prints output vertically.

- `--wait[=count]`, `-w[count]`

  If the connection cannot be established, wait and retry instead of aborting. If a *count* value is given, it indicates the number of times to retry. The default is one time.

You can also set the following variables by using `--var_name=value`.

- `connect_timeout`

  The maximum number of seconds before connection timeout. The default value is 43200 (12 hours).

- `shutdown_timeout`

  The maximum number of seconds to wait for server shutdown. The default value is 3600 (1 hour).

4.5.3 mysqlcheck — A Table Maintenance Program

The mysqlcheck client performs table maintenance: It checks, repairs, optimizes, or analyzes tables.

Each table is locked and therefore unavailable to other sessions while it is being processed, although for check operations, the table is locked with a READ lock only (see Section 13.3.5, "LOCK TABLES and UNLOCK TABLES Syntax", for more information about READ and WRITE locks). Table maintenance operations can be time-consuming, particularly for large tables. If you use the --databases or --all-databases option to process all tables in one or more databases, an invocation of mysqlcheck might take a long time. (This is also true for mysql_upgrade because that program invokes mysqlcheck to check all tables and repair them if necessary.)

mysqlcheck is similar in function to myisamchk, but works differently. The main operational difference is that mysqlcheck must be used when the mysqld server is running, whereas myisamchk should be used when it is not. The benefit of using mysqlcheck is that you do not have to stop the server to perform table maintenance.

mysqlcheck uses the SQL statements CHECK TABLE, REPAIR TABLE, ANALYZE TABLE, and OPTIMIZE TABLE in a convenient way for the user. It determines which statements to use for the operation you want to perform, and then sends the statements to the server to be executed. For details about which storage engines each statement works with, see the descriptions for those statements in Section 13.7.2, "Table Maintenance Statements".

The MyISAM storage engine supports all four maintenance operations, so mysqlcheck can be used to perform any of them on MyISAM tables. Other storage engines do not necessarily support all operations. In such cases, an error message is displayed. For example, if test.t is a MEMORY table, an attempt to check it produces this result:

```
shell> mysqlcheck test t
test.t
note     : The storage engine for the table doesn't support check
```

If mysqlcheck is unable to repair a table, see Section 2.10.4, "Rebuilding or Repairing Tables or Indexes" for manual table repair strategies. This will be the case, for example, for InnoDB tables, which can be checked with CHECK TABLE, but not repaired with REPAIR TABLE.

⚠

Caution

It is best to make a backup of a table before performing a table repair operation; under some circumstances the operation might cause data loss. Possible causes include but are not limited to file system errors.

There are three general ways to invoke mysqlcheck:

```
shell> mysqlcheck [options] db_name [tbl_name ...]
shell> mysqlcheck [options] --databases db_name ...
shell> mysqlcheck [options] --all-databases
```

If you do not name any tables following *db_name* or if you use the `--databases` or `--all-databases` option, entire databases are checked.

`mysqlcheck` has a special feature compared to other client programs. The default behavior of checking tables (`--check`) can be changed by renaming the binary. If you want to have a tool that repairs tables by default, you should just make a copy of `mysqlcheck` named `mysqlrepair`, or make a symbolic link to `mysqlcheck` named `mysqlrepair`. If you invoke `mysqlrepair`, it repairs tables.

The names shown in the following table can be used to change `mysqlcheck` default behavior.

Command	Meaning
`mysqlrepair`	The default option is `--repair`
`mysqlanalyze`	The default option is `--analyze`
`mysqloptimize`	The default option is `--optimize`

`mysqlcheck` supports the following options, which can be specified on the command line or in the `[mysqlcheck]` and `[client]` groups of an option file. For information about option files, see Section 4.2.3.3, “Using Option Files”.

Table 4.8 `mysqlcheck` Options

Format	Option File	Description	Introduced
--all-databases	all-databases	Check all tables in all databases	
--all-in-1	all-in-1	Execute a single statement for each database that names all the tables from that database	
--analyze	analyze	Analyze the tables	
--auto-repair	auto-repair	If a checked table is corrupted, automatically fix it	
--bind-address=ip_address	bind-address	Use the specified network interface to connect to the MySQL Server	
--character-sets-dir=path	character-sets-dir	The directory where character sets are installed	
--check	check	Check the tables for errors	
--check-only-changed	check-only-changed	Check only tables that have changed since the last check	
--check-upgrade	check-upgrade	Invoke CHECK TABLE with the FOR UPGRADE option	
--compress	compress	Compress all information sent between the client and the server	
--databases	databases	Process all tables in the named databases	
--debug[=debug_options]	debug	Write a debugging log	
--debug-check	debug-check	Print debugging information when the program exits	
--debug-info	debug-info	Print debugging information, memory and CPU statistics when the program exits	

Format	Option File	Description	Introduced
--default-auth=plugin	default-auth=plugin	The authentication plugin to use	
--default-character-set=charset_name	default-character-set	Use charset_name as the default character set	
--defaults-extra-file=file_name		Read option file in addition to the usual option files	
--defaults-file=file_name		Read only the given option file	
--defaults-group-suffix=str		Option group suffix value	
--extended	extended	Check and repair tables	
--fast	fast	Check only tables that have not been closed properly	
--fix-db-names	fix-db-names	Convert database names to 5.1 format	
--fix-table-names	fix-table-names	Convert table names to 5.1 format	
--force	force	Continue even if an SQL error occurs	
--help		Display help message and exit	
--host=host_name	host	Connect to the MySQL server on the given host	
--login-path=name		Read login path options from .mylogin.cnf	
--medium-check	medium-check	Do a check that is faster than an --extended operation	
--no-defaults		Do not read any option files	
--optimize	optimize	Optimize the tables	
--password[=password]	password	The password to use when connecting to the server	
--pipe		On Windows, connect to server using a named pipe	
--plugin-dir=path	plugin-dir=path	The directory where plugins are located	
--port=port_num	port	The TCP/IP port number to use for the connection	
--print-defaults		Print defaults	
--protocol=type	protocol	The connection protocol to use	
--quick	quick	The fastest method of checking	
--repair	repair	Perform a repair that can fix almost anything except unique keys that are not unique	
--secure-auth	secure-auth	Do not send passwords to the server in old (pre-4.1.1) format	5.7.4
--silent	silent	Silent mode	
--skip-database=db_name	skip-database	Omit this database from performed operations	5.7.1
--socket=path	socket	For connections to localhost	

Format	Option File	Description	Introduced
--ssl-ca=file_name	ssl-ca	The path to a file that contains a list of trusted SSL CAs	
--ssl-capath=dir_name	ssl-capath	The path to a directory that contains trusted SSL CA certificates in PEM format	
--ssl-cert=file_name	ssl-cert	The name of the SSL certificate file to use for establishing a secure connection	
--ssl-cipher=cipher_list	ssl-cipher	A list of allowable ciphers to use for SSL encryption	
--ssl-crl=file_name	ssl-crl	The path to a file that contains certificate revocation lists	
--ssl-crlpath=dir_name	ssl-crlpath	The path to a directory that contains certificate revocation list files	
--ssl-key=file_name	ssl-key	The name of the SSL key file to use for establishing a secure connection	
--ssl-verify-server-cert	ssl-verify-server-cert	The server's Common Name value in its certificate is verified against the host name used when connecting to the server	
--tables	tables	Overrides the --databases or -B option	
--use-frm	use-frm	For repair operations on MyISAM tables	
--user=user_name,	user	MySQL user name to use when connecting to server	
--verbose		Verbose mode	
--version		Display version information and exit	
--write-binlog	write-binlog	Log ANALYZE, OPTIMIZE, REPAIR statements to binary log. --skip-write-binlog adds NO_WRITE_TO_BINLOG to these statements.	

- `--help`, `-?`

  Display a help message and exit.

- `--all-databases`, `-A`

  Check all tables in all databases. This is the same as using the `--databases` option and naming all the databases on the command line.

- `--all-in-1`, `-1`

  Instead of issuing a statement for each table, execute a single statement for each database that names all the tables from that database to be processed.

- `--analyze`, `-a`

  Analyze the tables.

- `--auto-repair`

  If a checked table is corrupted, automatically fix it. Any necessary repairs are done after all tables have been checked.

- `--bind-address=ip_address`

  On a computer having multiple network interfaces, this option can be used to select which interface is employed when connecting to the MySQL server.

- `--character-sets-dir=path`

  The directory where character sets are installed. See Section 10.5, “Character Set Configuration”.

- `--check`, `-c`

  Check the tables for errors. This is the default operation.

- `--check-only-changed`, `-C`

  Check only tables that have changed since the last check or that have not been closed properly.

- `--check-upgrade`, `-g`

  Invoke `CHECK TABLE` with the `FOR UPGRADE` option to check tables for incompatibilities with the current version of the server. This option automatically enables the `--fix-db-names` and `--fix-table-names` options.

- `--compress`

  Compress all information sent between the client and the server if both support compression.

- `--databases`, `-B`

  Process all tables in the named databases. Normally, `mysqlcheck` treats the first name argument on the command line as a database name and following names as table names. With this option, it treats all name arguments as database names.

- `--debug[=debug_options]`, `-# [debug_options]`

  Write a debugging log. A typical *`debug_options`* string is `d:t:o,file_name`. The default is `d:t:o`.

- `--debug-check`

  Print some debugging information when the program exits.

- `--debug-info`

  Print debugging information and memory and CPU usage statistics when the program exits.

- `--default-character-set=charset_name`

  Use *`charset_name`* as the default character set. See Section 10.5, “Character Set Configuration”.

- `--defaults-extra-file=file_name`

  Read this option file after the global option file but (on Unix) before the user option file. If the file does not exist or is otherwise inaccessible, an error occurs. *`file_name`* is interpreted relative to the current directory if given as a relative path name rather than a full path name.

- `--defaults-file=file_name`

  Use only the given option file. If the file does not exist or is otherwise inaccessible, an error occurs. *`file_name`* is interpreted relative to the current directory if given as a relative path name rather than a full path name.

- `--defaults-group-suffix=str`

  Read not only the usual option groups, but also groups with the usual names and a suffix of *str*. For example, `mysqlcheck` normally reads the `[client]` and `[mysqlcheck]` groups. If the `--defaults-group-suffix=_other` option is given, `mysqlcheck` also reads the `[client_other]` and `[mysqlcheck_other]` groups.

- `--extended`, `-e`

  If you are using this option to check tables, it ensures that they are 100% consistent but takes a long time.

  If you are using this option to repair tables, it runs an extended repair that may not only take a long time to execute, but may produce a lot of garbage rows also!

- `--default-auth=plugin`

  The client-side authentication plugin to use. See Section 6.3.8, “Pluggable Authentication”.

- `--fast`, `-F`

  Check only tables that have not been closed properly.

- `--fix-db-names`

  Convert database names to 5.1 format. Only database names that contain special characters are affected.

- `--fix-table-names`

  Convert table names to 5.1 format. Only table names that contain special characters are affected. This option also applies to views.

- `--force`, `-f`

  Continue even if an SQL error occurs.

- `--host=host_name`, `-h host_name`

  Connect to the MySQL server on the given host.

- `--login-path=name`

  Read options from the named login path in the `.mylogin.cnf` login file. A “login path” is an option group that permits only a limited set of options: `host`, `user`, and `password`. Think of a login path as a set of values that indicate the server host and the credentials for authenticating with the server. To create the login file, use the `mysql_config_editor` utility. See Section 4.6.6, “`mysql_config_editor` — MySQL Configuration Utility”.

- `--medium-check`, `-m`

  Do a check that is faster than an `--extended` operation. This finds only 99.99% of all errors, which should be good enough in most cases.

- `--no-defaults`

  Do not read any option files. If program startup fails due to reading unknown options from an option file, `--no-defaults` can be used to prevent them from being read.

The exception is that the .mylogin.cnf file, if it exists, is read in all cases. This permits passwords to be specified in a safer way than on the command line even when --no-defaults is used. (.mylogin.cnf is created by the mysql_config_editor utility. See Section 4.6.6, “mysql_config_editor — MySQL Configuration Utility”.)

- --optimize, -o

  Optimize the tables.

- --password[=*password*], -p[*password*]

  The password to use when connecting to the server. If you use the short option form (-p), you *cannot* have a space between the option and the password. If you omit the *password* value following the --password or -p option on the command line, mysqlcheck prompts for one.

  Specifying a password on the command line should be considered insecure. See Section 6.1.2.1, “End-User Guidelines for Password Security”. You can use an option file to avoid giving the password on the command line.

- --pipe, -W

  On Windows, connect to the server using a named pipe. This option applies only if the server supports named-pipe connections.

- --plugin-dir=*path*

  The directory in which to look for plugins. It may be necessary to specify this option if the --default-auth option is used to specify an authentication plugin but mysqlcheck does not find it. See Section 6.3.8, “Pluggable Authentication”.

- --port=*port_num*, -P *port_num*

  The TCP/IP port number to use for the connection.

- --print-defaults

  Print the program name and all options that it gets from option files.

- --protocol={TCP|SOCKET|PIPE|MEMORY}

  The connection protocol to use for connecting to the server. It is useful when the other connection parameters normally would cause a protocol to be used other than the one you want. For details on the permissible values, see Section 4.2.2, “Connecting to the MySQL Server”.

- --quick, -q

  If you are using this option to check tables, it prevents the check from scanning the rows to check for incorrect links. This is the fastest check method.

  If you are using this option to repair tables, it tries to repair only the index tree. This is the fastest repair method.

- --repair, -r

  Perform a repair that can fix almost anything except unique keys that are not unique.

- --secure-auth

Do not send passwords to the server in old (pre-4.1) format. This prevents connections except for servers that use the newer password format. This option is enabled by default; use --skip-secure-auth to disable it. This option was added in MySQL 5.7.4.

Note

Passwords that use the pre-4.1 hashing method are less secure than passwords that use the native password hashing method and should be avoided. Pre-4.1 passwords are deprecated and support for them will be removed in a future MySQL release. For account upgrade instructions, see Section 6.3.9.3, “Migrating Away from Pre-4.1 Password Hashing and the mysql_old_password Plugin”.

- --silent, -s

  Silent mode. Print only error messages.

- --skip-database=*db_name*

  Do not include the named database (case sensitive) in the operations performed by mysqlcheck.

- --socket=*path*, -S *path*

  For connections to localhost, the Unix socket file to use, or, on Windows, the name of the named pipe to use.

- --ssl*

  Options that begin with --ssl specify whether to connect to the server using SSL and indicate where to find SSL keys and certificates. See Section 6.3.11.4, “SSL Command Options”.

- --tables

  Override the --databases or -B option. All name arguments following the option are regarded as table names.

- --use-frm

  For repair operations on MyISAM tables, get the table structure from the .frm file so that the table can be repaired even if the .MYI header is corrupted.

- --user=*user_name*, -u *user_name*

  The MySQL user name to use when connecting to the server.

- --verbose, -v

  Verbose mode. Print information about the various stages of program operation.

- --version, -V

  Display version information and exit.

- --write-binlog

  This option is enabled by default, so that ANALYZE TABLE, OPTIMIZE TABLE, and REPAIR TABLE statements generated by mysqlcheck are written to the binary log. Use --skip-write-binlog to cause NO_WRITE_TO_BINLOG to be added to the statements so that they are not logged. Use the --

skip-write-binlog when these statements should not be sent to replication slaves or run when using the binary logs for recovery from backup.

4.5.4 mysqldump — A Database Backup Program

The mysqldump client is a utility that performs logical backups, producing a set of SQL statements that can be run to reproduce the original schema objects, table data, or both. It dumps one or more MySQL database for backup or transfer to another SQL server. The mysqldump command can also generate output in CSV, other delimited text, or XML format.

mysqldump requires at least the SELECT privilege for dumped tables, SHOW VIEW for dumped views, TRIGGER for dumped triggers, and LOCK TABLES if the --single-transaction option is not used. Certain options might require other privileges as noted in the option descriptions.

To reload a dump file, you must have the same privileges needed to create each of the dumped objects by issuing CREATE statements manually.

mysqldump output can include ALTER DATABASE statements that change the database collation. These may be used when dumping stored programs to preserve their character encodings. To reload a dump file containing such statements, the ALTER privilege for the affected database is required.

Performance and Scalability Considerations

mysqldump advantages include the convenience and flexibility of viewing or even editing the output before restoring. You can clone databases for development and DBA work, or produce slight variations of an existing database for testing. It is not intended as a fast or scalable solution for backing up substantial amounts of data. With large data sizes, even if the backup step takes a reasonable time, restoring the data can be very slow because replaying the SQL statements involves disk I/O for insertion, index creation, and so on.

For large-scale backup and restore, a physical backup is more appropriate, to copy the data files in their original format that can be restored quickly:

- If your tables are primarily InnoDB tables, or if you have a mix of InnoDB and MyISAM tables, consider using the mysqlbackup command of the MySQL Enterprise Backup product. (Available as part of the Enterprise subscription.) It provides the best performance for InnoDB backups with minimal disruption; it can also back up tables from MyISAM and other storage engines; and it provides a number of convenient options to accommodate different backup scenarios. See Section 23.2, “MySQL Enterprise Backup”.

- If your tables are primarily MyISAM tables, consider using the mysqlhotcopy instead, for better performance than mysqldump of backup and restore operations. See Section 4.6.9, “mysqlhotcopy — A Database Backup Program”.

mysqldump can retrieve and dump table contents row by row, or it can retrieve the entire content from a table and buffer it in memory before dumping it. Buffering in memory can be a problem if you are dumping large tables. To dump tables row by row, use the --quick option (or --opt, which enables --quick). The --opt option (and hence --quick) is enabled by default, so to enable memory buffering, use --skip-quick.

If you are using a recent version of mysqldump to generate a dump to be reloaded into a very old MySQL server, use the --skip-opt option instead of the --opt or --extended-insert option.

For additional information about mysqldump, see Section 7.4, “Using mysqldump for Backups”.

Syntax

There are in general three ways to use mysqldump—in order to dump a set of one or more tables, a set of one or more complete databases, or an entire MySQL server—as shown here:

```
shell> mysqldump [options] db_name [tbl_name ...]
shell> mysqldump [options] --databases db_name ...
shell> mysqldump [options] --all-databases
```

To dump entire databases, do not name any tables following *db_name*, or use the `--databases` or `--all-databases` option.

To see a list of the options your version of `mysqldump` supports, issue the command `mysqldump --help`.

Option Syntax - Alphabetical Summary

`mysqldump` supports the following options, which can be specified on the command line or in the `[mysqldump]` and `[client]` groups of an option file. For information about option files, see Section 4.2.3.3, "Using Option Files".

Table 4.9 `mysqldump` Options

Format	Option File	Description	Introduced
--add-drop-database	add-drop-database	Add a DROP DATABASE statement before each CREATE DATABASE statement	
--add-drop-table	add-drop-table	Add a DROP TABLE statement before each CREATE TABLE statement	
--add-drop-trigger	add-drop-trigger	Add a DROP TRIGGER statement before each CREATE TRIGGER statement	
--add-locks	add-locks	Surround each table dump with LOCK TABLES and UNLOCK TABLES statements	
--all-databases	all-databases	Dump all tables in all databases	
--allow-keywords	allow-keywords	Allow creation of column names that are keywords	
--apply-slave-statements	apply-slave-statements	Include STOP SLAVE prior to CHANGE MASTER statement and START SLAVE at end of output	
--bind-address=ip_address	bind-address	Use the specified network interface to connect to the MySQL Server	
--comments	comments	Add comments to the dump file	
--compact	compact	Produce more compact output	
--compatible=name[,name,...]	compatible	Produce output that is more compatible with other database systems or with older MySQL servers	
--complete-insert	complete-insert	Use complete INSERT statements that include column names	
--create-options	create-options	Include all MySQL-specific table options in CREATE TABLE statements	
--databases	databases	Dump several databases	
--debug[=debug_options]	debug	Write a debugging log	
--debug-check	debug-check	Print debugging information when the program exits	
--debug-info	debug-info	Print debugging information, memory and CPU statistics when the program exits	

Format	Option File	Description	Introduced
--default-auth=plugin	default-auth=plugin	The authentication plugin to use	
--default-character-set=charset_name	default-character-set	Use charset_name as the default character set	
--defaults-extra-file=file_name		Read option file in addition to the usual option files	
--defaults-file=file_name		Read only the given option file	
--defaults-group-suffix=str		Option group suffix value	
--delete-master-logs	delete-master-logs	On a master replication server, delete the binary logs after performing the dump operation	
--disable-keys	disable-keys	For each table, surround the INSERT statements with statements to disable and enable keys	
--dump-date	dump-date	Include dump date as "Dump completed on" comment if --comments is given	
--dump-slave[=value]	dump-slave	Include CHANGE MASTER statement that lists binary log coordinates of slave's master	
--events	events	Dump events from the dumped databases	
--extended-insert	extended-insert	Use multiple-row INSERT syntax that include several VALUES lists	
--fields-enclosed-by=string	fields-enclosed-by	This option is used with the --tab option and has the same meaning as the corresponding clause for LOAD DATA INFILE	
--fields-escaped-by	fields-escaped-by	This option is used with the --tab option and has the same meaning as the corresponding clause for LOAD DATA INFILE	
--fields-optionally-enclosed-by=string	fields-optionally-enclosed-by	This option is used with the --tab option and has the same meaning as the corresponding clause for LOAD DATA INFILE	
--fields-terminated-by=string	fields-terminated-by	This option is used with the --tab option and has the same meaning as the corresponding clause for LOAD DATA INFILE	
--flush-logs	flush-logs	Flush the MySQL server log files before starting the dump	
--flush-privileges	flush-privileges	Emit a FLUSH PRIVILEGES statement after dumping the mysql database	
--help		Display help message and exit	
--hex-blob	hex-blob	Dump binary columns using hexadecimal notation (for example, 'abc' becomes 0x616263)	
--host	host	Host to connect to (IP address or hostname)	
--ignore-error=error[,error]...	ignore-error	Ignore the specified errors	5.7.1

Format	Option File	Description	Introduced
--ignore-table=db_name.tbl_name	ignore-table	Do not dump the given table	
--include-master-host-port	include-master-host-port	Include MASTER_HOST/MASTER_PORT options in CHANGE MASTER statement produced with --dump-slave	
--insert-ignore	insert-ignore	Write INSERT IGNORE statements rather than INSERT statements	
--lines-terminated-by=string	lines-terminated-by	This option is used with the --tab option and has the same meaning as the corresponding clause for LOAD DATA INFILE	
--lock-all-tables	lock-all-tables	Lock all tables across all databases	
--lock-tables	lock-tables	Lock all tables before dumping them	
--log-error=file_name	log-error	Append warnings and errors to the named file	
--login-path=name		Read login path options from .mylogin.cnf	
--master-data[=value]	master-data	Write the binary log file name and position to the output	
--max_allowed_packet=value	max_allowed_packet	The maximum packet length to send to or receive from the server	
--net_buffer_length=value	net_buffer_length	The buffer size for TCP/IP and socket communication	
--no-autocommit	no-autocommit	Enclose the INSERT statements for each dumped table within SET autocommit = 0 and COMMIT statements	
--no-create-db	no-create-db	This option suppresses the CREATE DATABASE statements	
--no-create-info	no-create-info	Do not write CREATE TABLE statements that re-create each dumped table	
--no-data	no-data	Do not dump table contents	
--no-defaults		Do not read any option files	
--no-set-names	no-set-names	Same as --skip-set-charset	
--no-tablespaces	no-tablespaces	Do not write any CREATE LOGFILE GROUP or CREATE TABLESPACE statements in output	
--opt	opt	Shorthand for --add-drop-table --add-locks --create-options --disable-keys --extended-insert --lock-tables --quick --set-charset.	
--order-by-primary	order-by-primary	Dump each table's rows sorted by its primary key, or by its first unique index	
--password[=password]	password	The password to use when connecting to the server	
--pipe		On Windows, connect to server using a named pipe	
--plugin-dir=path	plugin-dir=path	The directory where plugins are located	
--port=port_num	port	The TCP/IP port number to use for the connection	

Format	Option File	Description	Introduced
--print-defaults		Print defaults	
--protocol=type	protocol	The connection protocol to use	
--quick	quick	Retrieve rows for a table from the server a row at a time	
--quote-names	quote-names	Quote identifiers within backtick characters	
--replace	replace	Write REPLACE statements rather than INSERT statements	
--result-file=file	result-file	Direct output to a given file	
--routines	routines	Dump stored routines (procedures and functions) from the dumped databases	
--secure-auth	secure-auth	Do not send passwords to the server in old (pre-4.1.1) format	5.7.4
--set-charset	set-charset	Add SET NAMES default_character_set to output	
--set-gtid-purged=value	set-gtid-purged	Whether to add SET @@GLOBAL.GTID_PURGED to output	
--single-transaction	single-transaction	This option issues a BEGIN SQL statement before dumping data from the server	
--skip-add-drop-table	skip-add-drop-table	Do not add a DROP TABLE statement before each CREATE TABLE statement	
--skip-add-locks	skip-add-locks	Do not add locks	
--skip-comments	skip-comments	Do not add comments to the dump file	
--skip-compact	skip-compact	Do not produce more compact output	
--skip-disable-keys	skip-disable-keys	Do not disable keys	
--skip-extended-insert	skip-extended-insert	Turn off extended-insert	
--skip-opt	skip-opt	Turn off the options set by --opt	
--skip-quick	skip-quick	Do not retrieve rows for a table from the server a row at a time	
--skip-quote-names	skip-quote-names	Do not quote identifiers	
--skip-set-charset	skip-set-charset	Suppress the SET NAMES statement	
--skip-triggers	skip-triggers	Do not dump triggers	
--skip-tz-utc	skip-tz-utc	Turn off tz-utc	
--socket=path	socket	For connections to localhost	
--ssl-ca=file_name	ssl-ca	The path to a file that contains a list of trusted SSL CAs	
--ssl-capath=dir_name	ssl-capath	The path to a directory that contains trusted SSL CA certificates in PEM format	
--ssl-cert=file_name	ssl-cert	The name of the SSL certificate file to use for establishing a secure connection	
--ssl-cipher=cipher_list	ssl-cipher	A list of allowable ciphers to use for SSL encryption	

Format	Option File	Description	Introduced
--ssl-crl=file_name	ssl-crl	The path to a file that contains certificate revocation lists	
--ssl-crlpath=dir_name	ssl-crlpath	The path to a directory that contains certificate revocation list files	
--ssl-key=file_name	ssl-key	The name of the SSL key file to use for establishing a secure connection	
--ssl-verify-server-cert	ssl-verify-server-cert	The server's Common Name value in its certificate is verified against the host name used when connecting to the server	
--tab=path	tab	Produce tab-separated data files	
--tables	tables	Override the --databases or -B option	
--triggers	triggers	Dump triggers for each dumped table	
--tz-utc	tz-utc	Add SET TIME_ZONE='+00:00' to the dump file	
--user=user_name	user	MySQL user name to use when connecting to server	
--verbose		Verbose mode	
--version		Display version information and exit	
--where='where_condition'	where	Dump only rows selected by the given WHERE condition	
--xml	xml	Produce XML output	

Connection Options

The mysqldump command logs into a MySQL server to extract information. The following options specify how to connect to the MySQL server, either on the same machine or a remote system.

- --bind-address=*ip_address*

  On a computer having multiple network interfaces, this option can be used to select which interface is employed when connecting to the MySQL server.

- --compress, -C

  Compress all information sent between the client and the server if both support compression.

- --default-auth=*plugin*

  The client-side authentication plugin to use. See Section 6.3.8, “Pluggable Authentication”.

- --host=*host_name*, -h *host_name*

  Dump data from the MySQL server on the given host. The default host is localhost.

- --login-path=*name*

  Read options from the named login path in the .mylogin.cnf login file. A “login path” is an option group that permits only a limited set of options: host, user, and password. Think of a login path as a set of values that indicate the server host and the credentials for authenticating with the server. To create the login file, use the mysql_config_editor utility. See Section 4.6.6, “mysql_config_editor — MySQL Configuration Utility”.

- `--password[=password]`, `-p[password]`

  The password to use when connecting to the server. If you use the short option form (`-p`), you *cannot* have a space between the option and the password. If you omit the *`password`* value following the `--password` or `-p` option on the command line, `mysqldump` prompts for one.

  Specifying a password on the command line should be considered insecure. See Section 6.1.2.1, “End-User Guidelines for Password Security”. You can use an option file to avoid giving the password on the command line.

- `--pipe`, `-W`

  On Windows, connect to the server using a named pipe. This option applies only if the server supports named-pipe connections.

- `--plugin-dir=path`

  The directory in which to look for plugins. It may be necessary to specify this option if the `--default-auth` option is used to specify an authentication plugin but `mysqldump` does not find it. See Section 6.3.8, “Pluggable Authentication”.

- `--port=port_num`, `-P port_num`

  The TCP/IP port number to use for the connection.

- `--protocol={TCP|SOCKET|PIPE|MEMORY}`

  The connection protocol to use for connecting to the server. It is useful when the other connection parameters normally would cause a protocol to be used other than the one you want. For details on the permissible values, see Section 4.2.2, “Connecting to the MySQL Server”.

- `--secure-auth`

  Do not send passwords to the server in old (pre-4.1) format. This prevents connections except for servers that use the newer password format. This option is enabled by default; use `--skip-secure-auth` to disable it. This option was added in MySQL 5.7.4.

  > **Note**
  >
  > Passwords that use the pre-4.1 hashing method are less secure than passwords that use the native password hashing method and should be avoided. Pre-4.1 passwords are deprecated and support for them will be removed in a future MySQL release. For account upgrade instructions, see Section 6.3.9.3, “Migrating Away from Pre-4.1 Password Hashing and the `mysql_old_password` Plugin”.

- `--socket=path`, `-S path`

  For connections to `localhost`, the Unix socket file to use, or, on Windows, the name of the named pipe to use.

- `--ssl*`

  Options that begin with `--ssl` specify whether to connect to the server using SSL and indicate where to find SSL keys and certificates. See Section 6.3.11.4, “SSL Command Options”.

- `--user=user_name`, `-u user_name`

  The MySQL user name to use when connecting to the server.

You can also set the following variables by using --*var_name*=*value* syntax:

- max_allowed_packet

  The maximum size of the buffer for client/server communication. The default is 24MB, the maximum is 1GB.

- net_buffer_length

  The initial size of the buffer for client/server communication. When creating multiple-row INSERT statements (as with the --extended-insert or --opt option), mysqldump creates rows up to net_buffer_length length. If you increase this variable, ensure that the net_buffer_length variable in the MySQL server is at least this large.

Option-File Options

These options are used to control which option files to read.

- --defaults-extra-file=*file_name*

  Read this option file after the global option file but (on Unix) before the user option file. If the file does not exist or is otherwise inaccessible, an error occurs. *file_name* is interpreted relative to the current directory if given as a relative path name rather than a full path name.

- --defaults-file=*file_name*

  Use only the given option file. If the file does not exist or is otherwise inaccessible, an error occurs. *file_name* is interpreted relative to the current directory if given as a relative path name rather than a full path name.

- --defaults-group-suffix=*str*

  Read not only the usual option groups, but also groups with the usual names and a suffix of *str*. For example, mysqldump normally reads the [client] and [mysqldump] groups. If the --defaults-group-suffix=_other option is given, mysqldump also reads the [client_other] and [mysqldump_other] groups.

- --no-defaults

  Do not read any option files. If program startup fails due to reading unknown options from an option file, --no-defaults can be used to prevent them from being read.

  The exception is that the .mylogin.cnf file, if it exists, is read in all cases. This permits passwords to be specified in a safer way than on the command line even when --no-defaults is used. (.mylogin.cnf is created by the mysql_config_editor utility. See Section 4.6.6, “mysql_config_editor — MySQL Configuration Utility”.)

- --print-defaults

  Print the program name and all options that it gets from option files.

DDL Options

Usage scenarios for mysqldump include setting up an entire new MySQL instance (including database tables), and replacing data inside an existing instance with existing databases and tables. The following options let you specify which things to tear down and set up when restoring a dump, by encoding various DDL statements within the dump file.

- `--add-drop-database`

  Add a `DROP DATABASE` statement before each `CREATE DATABASE` statement. This option is typically used in conjunction with the `--all-databases` or `--databases` option because no `CREATE DATABASE` statements are written unless one of those options is specified.

- `--add-drop-table`

  Add a `DROP TABLE` statement before each `CREATE TABLE` statement.

- `--add-drop-trigger`

  Add a `DROP TRIGGER` statement before each `CREATE TRIGGER` statement.

- `--all-tablespaces`, `-Y`

  Adds to a table dump all SQL statements needed to create any tablespaces used by an `NDB` table. This information is not otherwise included in the output from `mysqldump`. This option is currently relevant only to MySQL Cluster tables, which are not supported in MySQL 5.7.

- `--no-create-db`, `-n`

  This option suppresses the `CREATE DATABASE` statements that are otherwise included in the output if the `--databases` or `--all-databases` option is given.

- `--no-create-info`, `-t`

  Do not write `CREATE TABLE` statements that re-create each dumped table.

  Note

  This option does *not* not exclude statements creating log file groups or tablespaces from `mysqldump` output; however, you can use the `--no-tablespaces` option for this purpose.

- `--no-tablespaces`, `-y`

  This option suppresses all `CREATE LOGFILE GROUP` and `CREATE TABLESPACE` statements in the output of `mysqldump`.

- `--replace`

  Write `REPLACE` statements rather than `INSERT` statements.

Debug Options

The following options print debugging information, encode debugging information in the dump file, or let the dump operation proceed regardless of potential problems.

- `--allow-keywords`

  Permit creation of column names that are keywords. This works by prefixing each column name with the table name.

- `--comments`, `-i`

  Write additional information in the dump file such as program version, server version, and host. This option is enabled by default. To suppress this additional information, use `--skip-comments`.

- --debug[=*debug_options*], -# [*debug_options*]

  Write a debugging log. A typical *debug_options* string is d:t:o,*file_name*. The default value is d:t:o,/tmp/mysqldump.trace.

- --debug-check

  Print some debugging information when the program exits.

- --debug-info

  Print debugging information and memory and CPU usage statistics when the program exits.

- --dump-date

  If the --comments option is given, mysqldump produces a comment at the end of the dump of the following form:

  ```
  -- Dump completed on DATE
  ```

  However, the date causes dump files taken at different times to appear to be different, even if the data are otherwise identical. --dump-date and --skip-dump-date control whether the date is added to the comment. The default is --dump-date (include the date in the comment). --skip-dump-date suppresses date printing.

- --force, -f

  Ignore all errors; continue even if an SQL error occurs during a table dump.

  One use for this option is to cause mysqldump to continue executing even when it encounters a view that has become invalid because the definition refers to a table that has been dropped. Without --force, mysqldump exits with an error message. With --force, mysqldump prints the error message, but it also writes an SQL comment containing the view definition to the dump output and continues executing.

  If the --ignore-error option is also given to ignore specific errors, --force takes precedence.

- --log-error=*file_name*

  Log warnings and errors by appending them to the named file. The default is to do no logging.

- --skip-comments

  See the description for the --comments option.

- --verbose, -v

  Verbose mode. Print more information about what the program does.

Help Options

The following options display information about the mysqldump command itself.

- --help, -?

  Display a help message and exit.

- --version, -V

Display version information and exit.

Internationalization Options

The following options change how the mysqldump command represents character data with national language settings.

- --character-sets-dir=*path*

  The directory where character sets are installed. See Section 10.5, “Character Set Configuration”.

- --default-character-set=*charset_name*

  Use *charset_name* as the default character set. See Section 10.5, “Character Set Configuration”. If no character set is specified, mysqldump uses utf8, and earlier versions use latin1.

- --no-set-names, -N

  Turns off the --set-charset setting, the same as specifying --skip-set-charset.

- --set-charset

  Add SET NAMES *default_character_set* to the output. This option is enabled by default. To suppress the SET NAMES statement, use --skip-set-charset.

Replication Options

The mysqldump command is frequently used to create an empty instance, or an instance including data, on a slave server in a replication configuration. The following options apply to dumping and restoring data on replication master and slave servers.

- --apply-slave-statements

  For a slave dump produced with the --dump-slave option, add a STOP SLAVE statement before the CHANGE MASTER TO statement and a START SLAVE statement at the end of the output.

- --delete-master-logs

  On a master replication server, delete the binary logs by sending a PURGE BINARY LOGS statement to the server after performing the dump operation. This option automatically enables --master-data.

- --dump-slave[=*value*]

  This option is similar to --master-data except that it is used to dump a replication slave server to produce a dump file that can be used to set up another server as a slave that has the same master as the dumped server. It causes the dump output to include a CHANGE MASTER TO statement that indicates the binary log coordinates (file name and position) of the dumped slave's master. These are the master server coordinates from which the slave should start replicating.

  --dump-slave causes the coordinates from the master to be used rather than those of the dumped server, as is done by the --master-data option. In addition, specfiying this option causes the --master-data option to be overridden, if used, and effectively ignored.

  The option value is handled the same way as for --master-data (setting no value or 1 causes a CHANGE MASTER TO statement to be written to the dump, setting 2 causes the statement to be written but encased in SQL comments) and has the same effect as --master-data in terms of enabling or disabling other options and in how locking is handled.

This option causes mysqldump to stop the slave SQL thread before the dump and restart it again after.

In conjunction with --dump-slave, the --apply-slave-statements and --include-master-host-port options can also be used.

- --include-master-host-port

  For the CHANGE MASTER TO statement in a slave dump produced with the --dump-slave option, add MASTER_HOST and MASTER_PORT options for the host name and TCP/IP port number of the slave's master.

- --master-data[=*value*]

  Use this option to dump a master replication server to produce a dump file that can be used to set up another server as a slave of the master. It causes the dump output to include a CHANGE MASTER TO statement that indicates the binary log coordinates (file name and position) of the dumped server. These are the master server coordinates from which the slave should start replicating after you load the dump file into the slave.

  If the option value is 2, the CHANGE MASTER TO statement is written as an SQL comment, and thus is informative only; it has no effect when the dump file is reloaded. If the option value is 1, the statement is not written as a comment and takes effect when the dump file is reloaded. If no option value is specified, the default value is 1.

  This option requires the RELOAD privilege and the binary log must be enabled.

  The --master-data option automatically turns off --lock-tables. It also turns on --lock-all-tables, unless --single-transaction also is specified, in which case, a global read lock is acquired only for a short time at the beginning of the dump (see the description for --single-transaction). In all cases, any action on logs happens at the exact moment of the dump.

  It is also possible to set up a slave by dumping an existing slave of the master, using the --dump-slave option, which overrides --master-data and causes it to be ignored if both options are used.

- --set-gtid-purged=*value*

  This option enables control over global transaction ID (GTID) information written to the dump file, by indicating whether to add a SET @@global.gtid_purged statement to the output.

  The following table shows the permitted option values. The default value is AUTO.

Value	**Meaning**
OFF	Add no SET statement to the output.
ON	Add a SET statement to the output. An error occurs if GTIDs are not enabled on the server.
AUTO	Add a SET statement to the output if GTIDs are enabled on the server.

Format Options

The following options specify how to represent the entire dump file or certain kinds of data in the dump file. They also control whether certain optional information is written to the dump file.

- --compact

  Produce more compact output. This option enables the --skip-add-drop-table, --skip-add-locks, --skip-comments, --skip-disable-keys, and --skip-set-charset options.

- `--compatible=name`

  Produce output that is more compatible with other database systems or with older MySQL servers. The value of *name* can be `ansi`, `mysql323`, `mysql40`, `postgresql`, `oracle`, `mssql`, `db2`, `maxdb`, `no_key_options`, `no_table_options`, or `no_field_options`. To use several values, separate them by commas. These values have the same meaning as the corresponding options for setting the server SQL mode. See Section 5.1.7, “Server SQL Modes”.

  This option does not guarantee compatibility with other servers. It only enables those SQL mode values that are currently available for making dump output more compatible. For example, `--compatible=oracle` does not map data types to Oracle types or use Oracle comment syntax.

  This option requires a server version of 4.1.0 or higher. With older servers, it does nothing.

- `--complete-insert`, `-c`

  Use complete `INSERT` statements that include column names.

- `--create-options`

  Include all MySQL-specific table options in the `CREATE TABLE` statements.

- `--fields-terminated-by=...`, `--fields-enclosed-by=...`, `--fields-optionally-enclosed-by=...`, `--fields-escaped-by=...`

  These options are used with the `--tab` option and have the same meaning as the corresponding `FIELDS` clauses for `LOAD DATA INFILE`. See Section 13.2.6, “`LOAD DATA INFILE` Syntax”.

- `--hex-blob`

  Dump binary columns using hexadecimal notation (for example, `'abc'` becomes `0x616263`). The affected data types are `BINARY`, `VARBINARY`, the `BLOB` types, and `BIT`.

- `--lines-terminated-by=...`

  This option is used with the `--tab` option and has the same meaning as the corresponding `LINES` clause for `LOAD DATA INFILE`. See Section 13.2.6, “`LOAD DATA INFILE` Syntax”.

- `--quote-names`, `-Q`

  Quote identifiers (such as database, table, and column names) within “`” characters. If the `ANSI_QUOTES` SQL mode is enabled, identifiers are quoted within “`"`” characters. This option is enabled by default. It can be disabled with `--skip-quote-names`, but this option should be given after any option such as `--compatible` that may enable `--quote-names`.

- `--result-file=file_name`, `-r file_name`

  Direct output to a given file. This option should be used on Windows to prevent newline “`\n`” characters from being converted to “`\r\n`” carriage return/newline sequences. The result file is created and its previous contents overwritten, even if an error occurs while generating the dump.

- `--tab=path`, `-T path`

  Produce tab-separated text-format data files. For each dumped table, `mysqldump` creates a `tbl_name.sql` file that contains the `CREATE TABLE` statement that creates the table, and the server writes a `tbl_name.txt` file that contains its data. The option value is the directory in which to write the files.

Note

This option should be used only when mysqldump is run on the same machine as the mysqld server. You must have the FILE privilege, and the server must have permission to write files in the directory that you specify.

By default, the .txt data files are formatted using tab characters between column values and a newline at the end of each line. The format can be specified explicitly using the --fields-*xxx* and --lines-terminated-by options.

Column values are converted to the character set specified by the --default-character-set option.

- --tz-utc

  This option enables TIMESTAMP columns to be dumped and reloaded between servers in different time zones. mysqldump sets its connection time zone to UTC and adds SET TIME_ZONE='+00:00' to the dump file. Without this option, TIMESTAMP columns are dumped and reloaded in the time zones local to the source and destination servers, which can cause the values to change if the servers are in different time zones. --tz-utc also protects against changes due to daylight saving time. --tz-utc is enabled by default. To disable it, use --skip-tz-utc.

- --xml, -X

  Write dump output as well-formed XML.

  NULL, 'NULL', and Empty Values: For a column named *column_name*, the NULL value, an empty string, and the string value 'NULL' are distinguished from one another in the output generated by this option as follows.

Value:	XML Representation:
NULL (*unknown value*)	<field name="*column_name*" xsi:nil="true" />
'' (*empty string*)	<field name="*column_name*"></field>
'NULL' (*string value*)	<field name="*column_name*">NULL</field>

  The output from the mysql client when run using the --xml option also follows the preceding rules. (See Section 4.5.1.1, “mysql Options”.)

  XML output from mysqldump includes the XML namespace, as shown here:

```
shell> mysqldump --xml -u root world City
<?xml version="1.0"?>
<mysqldump xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<database name="world">
<table_structure name="City">
<field Field="ID" Type="int(11)" Null="NO" Key="PRI" Extra="auto_increment" />
<field Field="Name" Type="char(35)" Null="NO" Key="" Default="" Extra="" />
<field Field="CountryCode" Type="char(3)" Null="NO" Key="" Default="" Extra="" />
<field Field="District" Type="char(20)" Null="NO" Key="" Default="" Extra="" />
<field Field="Population" Type="int(11)" Null="NO" Key="" Default="0" Extra="" />
<key Table="City" Non_unique="0" Key_name="PRIMARY" Seq_in_index="1" Column_name="ID"
Collation="A" Cardinality="4079" Null="" Index_type="BTREE" Comment="" />
<options Name="City" Engine="MyISAM" Version="10" Row_format="Fixed" Rows="4079"
Avg_row_length="67" Data_length="273293" Max_data_length="18858823439613951"
```

```
Index_length="43008" Data_free="0" Auto_increment="4080"
Create_time="2007-03-31 01:47:01" Update_time="2007-03-31 01:47:02"
Collation="latin1_swedish_ci" Create_options="" Comment="" />
</table_structure>
<table_data name="City">
<row>
<field name="ID">1</field>
<field name="Name">Kabul</field>
<field name="CountryCode">AFG</field>
<field name="District">Kabol</field>
<field name="Population">1780000</field>
</row>

...

<row>
<field name="ID">4079</field>
<field name="Name">Rafah</field>
<field name="CountryCode">PSE</field>
<field name="District">Rafah</field>
<field name="Population">92020</field>
</row>
</table_data>
</database>
</mysqldump>
```

Filtering Options

The following options control which kinds of schema objects are written to the dump file: by category, such as triggers or events; by name, for example, choosing which databases and tables to dump; or even filtering rows from the table data using a `WHERE` clause.

- `--all-databases`, `-A`

  Dump all tables in all databases. This is the same as using the `--databases` option and naming all the databases on the command line.

- `--databases`, `-B`

  Dump several databases. Normally, `mysqldump` treats the first name argument on the command line as a database name and following names as table names. With this option, it treats all name arguments as database names. `CREATE DATABASE` and `USE` statements are included in the output before each new database.

- `--events`, `-E`

  Include Event Scheduler events for the dumped databases in the output.

- `--ignore-error=*error[,error]...*`

  Ignore the specified errors. The option value is a comma-separated list of error numbers specifying the errors to ignore during `mysqldump` execution. If the `--force` option is also given to ignore all errors, `--force` takes precedence.

  This option was added in MySQL 5.7.1.

- `--ignore-table=*db_name.tbl_name*`

  Do not dump the given table, which must be specified using both the database and table names. To ignore multiple tables, use this option multiple times. This option also can be used to ignore views.

- `--no-data`, `-d`

Do not write any table row information (that is, do not dump table contents). This is useful if you want to dump only the CREATE TABLE statement for the table (for example, to create an empty copy of the table by loading the dump file).

- --routines, -R

  Include stored routines (procedures and functions) for the dumped databases in the output. Use of this option requires the SELECT privilege for the mysql.proc table. The output generated by using --routines contains CREATE PROCEDURE and CREATE FUNCTION statements to re-create the routines. However, these statements do not include attributes such as the routine creation and modification timestamps. This means that when the routines are reloaded, they will be created with the timestamps equal to the reload time.

  If you require routines to be re-created with their original timestamp attributes, do not use --routines. Instead, dump and reload the contents of the mysql.proc table directly, using a MySQL account that has appropriate privileges for the mysql database.

- --tables

  Override the --databases or -B option. mysqldump regards all name arguments following the option as table names.

- --triggers

  Include triggers for each dumped table in the output. This option is enabled by default; disable it with --skip-triggers.

  Before MySQL 5.7.2, a table cannot have multiple triggers that have the same combination of trigger event (INSERT, UPDATE, DELETE) and action time (BEFORE, AFTER). MySQL 5.7.2 lifts this limitation and multiple triggers are permitted. mysqldump dumps triggers in activation order so that when the dump file is reloaded, triggers are re-created in the same activation order. However, if a mysqldump dump file contains multiple triggers for a table that have the same trigger event and action time, an error occurs for attempts to load the dump file into an older server that does not support multiple triggers. (For a workaround, see Section 2.10.2.1, “Downgrading to MySQL 5.6”; you can convert triggers to be compatible with older servers.)

- --where='*where_condition*', -w '*where_condition*'

  Dump only rows selected by the given WHERE condition. Quotes around the condition are mandatory if it contains spaces or other characters that are special to your command interpreter.

  Examples:

```
--where="user='jimf'"
-w"userid>1"
-w"userid<1"
```

Performance Options

The following options are the most relevant for the performance particularly of the restore operations. For large data sets, restore operation (processing the INSERT statements in the dump file) is the most time-consuming part. When it is urgent to restore data quickly, plan and test the performance of this stage in advance. For restore times measured in hours, you might prefer an alternative backup and restore solution, such as MySQL Enterprise Backup for InnoDB-only and mixed-use databases, or mysqlhotcopy for MyISAM-only databases.

Performance is also affected by the transactional options, primarily for the dump operation.

- --disable-keys, -K

  For each table, surround the INSERT statements with /*!40000 ALTER TABLE *tbl_name* DISABLE KEYS */; and /*!40000 ALTER TABLE *tbl_name* ENABLE KEYS */; statements. This makes loading the dump file faster because the indexes are created after all rows are inserted. This option is effective only for nonunique indexes of MyISAM tables.

- --extended-insert, -e

  Use multiple-row INSERT syntax that include several VALUES lists. This results in a smaller dump file and speeds up inserts when the file is reloaded.

- --insert-ignore

  Write INSERT IGNORE statements rather than INSERT statements.

- --opt

  This option, enabled by default, is shorthand for the combination of --add-drop-table --add-locks --create-options --disable-keys --extended-insert --lock-tables --quick --set-charset. It gives a fast dump operation and produces a dump file that can be reloaded into a MySQL server quickly.

  Because the --opt option is enabled by default, you only specify its converse, the --skip-opt to turn off several default settings. See the discussion of mysqldump option groups for information about selectively enabling or disabling a subset of the options affected by --opt.

- --quick, -q

  This option is useful for dumping large tables. It forces mysqldump to retrieve rows for a table from the server a row at a time rather than retrieving the entire row set and buffering it in memory before writing it out.

- --skip-opt

  See the description for the --opt option.

Transactional Options

The following options trade off the performance of the dump operation, against the reliability and consistency of the exported data.

- --add-locks

  Surround each table dump with LOCK TABLES and UNLOCK TABLES statements. This results in faster inserts when the dump file is reloaded. See Section 8.2.2.1, “Speed of INSERT Statements”.

- --flush-logs, -F

  Flush the MySQL server log files before starting the dump. This option requires the RELOAD privilege. If you use this option in combination with the --all-databases option, the logs are flushed *for each database dumped*. The exception is when using --lock-all-tables, --master-data, or --single-transaction: In this case, the logs are flushed only once, corresponding to the moment that all tables are locked. If you want your dump and the log flush to happen at exactly the same moment, you should use --flush-logs together with --lock-all-tables, --master-data, or --single-transaction.

- `--flush-privileges`

  Add a `FLUSH PRIVILEGES` statement to the dump output after dumping the `mysql` database. This option should be used any time the dump contains the `mysql` database and any other database that depends on the data in the `mysql` database for proper restoration.

  > **Note**
  >
  > For upgrades to MySQL 5.7.2 or higher from older versions, do not use `--flush-privileges`. For upgrade instructions in this case, see Section 2.10.1.2, “Upgrading from MySQL 5.6 to 5.7”.

- `--lock-all-tables`, `-x`

  Lock all tables across all databases. This is achieved by acquiring a global read lock for the duration of the whole dump. This option automatically turns off `--single-transaction` and `--lock-tables`.

- `--lock-tables`, `-l`

  For each dumped database, lock all tables to be dumped before dumping them. The tables are locked with `READ LOCAL` to permit concurrent inserts in the case of `MyISAM` tables. For transactional tables such as `InnoDB`, `--single-transaction` is a much better option than `--lock-tables` because it does not need to lock the tables at all.

  Because `--lock-tables` locks tables for each database separately, this option does not guarantee that the tables in the dump file are logically consistent between databases. Tables in different databases may be dumped in completely different states.

  Some options, such as `--opt`, automatically enable `--lock-tables`. If you want to override this, use `--skip-lock-tables` at the end of the option list.

- `--no-autocommit`

  Enclose the `INSERT` statements for each dumped table within `SET autocommit = 0` and `COMMIT` statements.

- `--order-by-primary`

  Dump each table's rows sorted by its primary key, or by its first unique index, if such an index exists. This is useful when dumping a `MyISAM` table to be loaded into an `InnoDB` table, but makes the dump operation take considerably longer.

- `--single-transaction`

  This option sets the transaction isolation mode to `REPEATABLE READ` and sends a `START TRANSACTION` SQL statement to the server before dumping data. It is useful only with transactional tables such as `InnoDB`, because then it dumps the consistent state of the database at the time when `START TRANSACTION` was issued without blocking any applications.

  When using this option, you should keep in mind that only `InnoDB` tables are dumped in a consistent state. For example, any `MyISAM` or `MEMORY` tables dumped while using this option may still change state.

  While a `--single-transaction` dump is in process, to ensure a valid dump file (correct table contents and binary log coordinates), no other connection should use the following statements: `ALTER TABLE`, `CREATE TABLE`, `DROP TABLE`, `RENAME TABLE`, `TRUNCATE TABLE`. A consistent read is not isolated from those statements, so use of them on a table to be dumped can cause the SELECT that is performed by mysqldump to retrieve the table contents to obtain incorrect contents or fail.

The --single-transaction option and the --lock-tables option are mutually exclusive because LOCK TABLES causes any pending transactions to be committed implicitly.

To dump large tables, combine the --single-transaction option with the --quick option.

Option Groups

- The --opt option turns on several settings that work together to perform a fast dump operation. All of these settings are on by default, because --opt is on by default. Thus you rarely if ever specify --opt. Instead, you can turn these settings off as a group by specifying --skip-opt, the optionally re-enable certain settings by specifying the associated options later on the command line.
- The --compact option turns off several settings that control whether optional statements and comments appear in the output. Again, you can follow this option with other options that re-enable certain settings, or turn all the settings on by using the --skip-compact form.

When you selectively enable or disable the effect of a group option, order is important because options are processed first to last. For example, --disable-keys --lock-tables --skip-opt would not have the intended effect; it is the same as --skip-opt by itself.

Examples

To make a backup of an entire database:

```
shell> mysqldump db_name > backup-file.sql
```

To load the dump file back into the server:

```
shell> mysql db_name < backup-file.sql
```

Another way to reload the dump file:

```
shell> mysql -e "source /path-to-backup/backup-file.sql" db_name
```

mysqldump is also very useful for populating databases by copying data from one MySQL server to another:

```
shell> mysqldump --opt db_name | mysql --host=remote_host -C db_name
```

You can dump several databases with one command:

```
shell> mysqldump --databases db_name1 [db_name2 ...] > my_databases.sql
```

To dump all databases, use the --all-databases option:

```
shell> mysqldump --all-databases > all_databases.sql
```

For InnoDB tables, mysqldump provides a way of making an online backup:

```
shell> mysqldump --all-databases --single-transaction > all_databases.sql
```

This backup acquires a global read lock on all tables (using FLUSH TABLES WITH READ LOCK) at the beginning of the dump. As soon as this lock has been acquired, the binary log coordinates are read and the lock is released. If long updating statements are running when the FLUSH statement is issued, the MySQL server may get stalled until those statements finish. After that, the dump becomes lock free and does not disturb reads and writes on the tables. If the update statements that the MySQL server receives are short (in terms of execution time), the initial lock period should not be noticeable, even with many updates.

For point-in-time recovery (also known as “roll-forward,” when you need to restore an old backup and replay the changes that happened since that backup), it is often useful to rotate the binary log (see Section 5.2.4, “The Binary Log”) or at least know the binary log coordinates to which the dump corresponds:

```
shell> mysqldump --all-databases --master-data=2 > all_databases.sql
```

Or:

```
shell> mysqldump --all-databases --flush-logs --master-data=2
              > all_databases.sql
```

The --master-data and --single-transaction options can be used simultaneously, which provides a convenient way to make an online backup suitable for use prior to point-in-time recovery if tables are stored using the InnoDB storage engine.

For more information on making backups, see Section 7.2, “Database Backup Methods”, and Section 7.3, “Example Backup and Recovery Strategy”.

- To select the effect of --opt except for some features, use the --skip option for each feature. To disable extended inserts and memory buffering, use --opt --skip-extended-insert --skip-quick. (Actually, --skip-extended-insert --skip-quick is sufficient because --opt is on by default.)

- To reverse --opt for all features except index disabling and table locking, use --skip-opt --disable-keys --lock-tables.

Restrictions

mysqldump does not dump the INFORMATION_SCHEMA database by default. To dump INFORMATION_SCHEMA, name it explicitly on the command line and also use the --skip-lock-tables option.

It is not recommended to restore from a dump made using mysqldump to a MySQL 5.6.9 or earlier server that has GTIDs enabled. See Section 16.1.3.4, “Restrictions on Replication with GTIDs”.

mysqldump never dumps the performance_schema database.

mysqldump includes statements to recreate the general_log and slow_query_log tables for dumps of the mysql database. Log table contents are not dumped.

If you encounter problems backing up views due to insufficient privileges, see Section E.5, “Restrictions on Views” for a workaround.

4.5.5 mysqlimport — A Data Import Program

The mysqlimport client provides a command-line interface to the LOAD DATA INFILE SQL statement. Most options to mysqlimport correspond directly to clauses of LOAD DATA INFILE syntax. See Section 13.2.6, “LOAD DATA INFILE Syntax”.

Invoke mysqlimport like this:

```
shell> mysqlimport [options] db_name textfile1 [textfile2 ...]
```

For each text file named on the command line, mysqlimport strips any extension from the file name and uses the result to determine the name of the table into which to import the file's contents. For example, files named patient.txt, patient.text, and patient all would be imported into a table named patient.

mysqlimport supports the following options, which can be specified on the command line or in the [mysqlimport] and [client] groups of an option file. For information about option files, see Section 4.2.3.3, “Using Option Files”.

Table 4.10 mysqlimport Options

Format	Option File	Description	Introduced
--bind-address=ip_address	bind-address	Use the specified network interface to connect to the MySQL Server	
--columns=column_list	columns	This option takes a comma-separated list of column names as its value	
--compress	compress	Compress all information sent between the client and the server	
--debug[=debug_options]	debug	Write a debugging log	
--debug-check	debug-check	Print debugging information when the program exits	
--debug-info	debug-info	Print debugging information, memory and CPU statistics when the program exits	
--default-auth=plugin	default-auth=plugin	The authentication plugin to use	
--default-character-set=charset_name	default-character-set	Use charset_name as the default character set	
--defaults-extra-file=file_name		Read option file in addition to the usual option files	
--defaults-file=file_name		Read only the given option file	
--defaults-group-suffix=str		Option group suffix value	
--delete	delete	Empty the table before importing the text file	
--fields-enclosed-by=string	fields-enclosed-by	This option has the same meaning as the corresponding clause for LOAD DATA INFILE	
--fields-escaped-by	fields-escaped-by	This option has the same meaning as the corresponding clause for LOAD DATA INFILE	
--fields-optionally-enclosed-by=string	fields-optionally-enclosed-by	This option has the same meaning as the corresponding clause for LOAD DATA INFILE	
--fields-terminated-by=string	fields-terminated-by	-- This option has the same meaning as the corresponding clause for LOAD DATA INFILE	
--force	force	Continue even if an SQL error occurs	
--help		Display help message and exit	
--host=host_name	host	Connect to the MySQL server on the given host	

Format	Option File	Description	Introduced
--ignore	ignore	See the description for the --replace option	
--ignore-lines=#	ignore-lines	Ignore the first N lines of the data file	
--lines-terminated-by=string	lines-terminated-by	This option has the same meaning as the corresponding clause for LOAD DATA INFILE	
--local	local	Read input files locally from the client host	
--lock-tables	lock-tables	Lock all tables for writing before processing any text files	
--login-path=name		Read login path options from .mylogin.cnf	
--low-priority	low-priority	Use LOW_PRIORITY when loading the table.	
--no-defaults		Do not read any option files	
--password[=password]	password	The password to use when connecting to the server	
--pipe		On Windows, connect to server using a named pipe	
--plugin-dir=path	plugin-dir=path	The directory where plugins are located	
--port=port_num	port	The TCP/IP port number to use for the connection	
--print-defaults		Print defaults	
--protocol=type	protocol	The connection protocol to use	
--replace	replace	The --replace and --ignore options control handling of input rows that duplicate existing rows on unique key values	
--secure-auth	secure-auth	Do not send passwords to the server in old (pre-4.1.1) format	5.7.4
--silent	silent	Produce output only when errors occur	
--socket=path	socket	For connections to localhost	
--ssl-ca=file_name	ssl-ca	The path to a file that contains a list of trusted SSL CAs	
--ssl-capath=dir_name	ssl-capath	The path to a directory that contains trusted SSL CA certificates in PEM format	
--ssl-cert=file_name	ssl-cert	The name of the SSL certificate file to use for establishing a secure connection	
--ssl-cipher=cipher_list	ssl-cipher	A list of allowable ciphers to use for SSL encryption	
--ssl-crl=file_name	ssl-crl	The path to a file that contains certificate revocation lists	
--ssl-crlpath=dir_name	ssl-crlpath	The path to a directory that contains certificate revocation list files	
--ssl-key=file_name	ssl-key	The name of the SSL key file to use for establishing a secure connection	
--ssl-verify-server-cert	ssl-verify-server-cert	The server's Common Name value in its certificate is verified against the host name used when connecting to the server	
--use-threads=#	use-threads	The number of threads for parallel file-loading	

Format	Option File	Description	Introduced
--user=user_name,	user	MySQL user name to use when connecting to server	
--verbose		Verbose mode	
--version		Display version information and exit	

- --help, -?

  Display a help message and exit.

- --bind-address=*ip_address*

  On a computer having multiple network interfaces, this option can be used to select which interface is employed when connecting to the MySQL server.

- --character-sets-dir=*path*

  The directory where character sets are installed. See Section 10.5, “Character Set Configuration”.

- --columns=*column_list*, -c *column_list*

  This option takes a comma-separated list of column names as its value. The order of the column names indicates how to match data file columns with table columns.

- --compress, -C

  Compress all information sent between the client and the server if both support compression.

- --debug[=*debug_options*], -# [*debug_options*]

  Write a debugging log. A typical *debug_options* string is d:t:o,*file_name*. The default is d:t:o.

- --debug-check

  Print some debugging information when the program exits.

- --debug-info

  Print debugging information and memory and CPU usage statistics when the program exits.

- --default-character-set=*charset_name*

  Use *charset_name* as the default character set. See Section 10.5, “Character Set Configuration”.

- --default-auth=*plugin*

  The client-side authentication plugin to use. See Section 6.3.8, “Pluggable Authentication”.

- --defaults-extra-file=*file_name*

  Read this option file after the global option file but (on Unix) before the user option file. If the file does not exist or is otherwise inaccessible, an error occurs. *file_name* is interpreted relative to the current directory if given as a relative path name rather than a full path name.

- --defaults-file=*file_name*

Use only the given option file. If the file does not exist or is otherwise inaccessible, an error occurs. *file_name* is interpreted relative to the current directory if given as a relative path name rather than a full path name.

- --defaults-group-suffix=*str*

  Read not only the usual option groups, but also groups with the usual names and a suffix of *str*. For example, mysqlimport normally reads the [client] and [mysqlimport] groups. If the --defaults-group-suffix=_other option is given, mysqlimport also reads the [client_other] and [mysqlimport_other] groups.

- --delete, -D

  Empty the table before importing the text file.

- --fields-terminated-by=..., --fields-enclosed-by=..., --fields-optionally-enclosed-by=..., --fields-escaped-by=...

  These options have the same meaning as the corresponding clauses for LOAD DATA INFILE. See Section 13.2.6, "LOAD DATA INFILE Syntax".

- --force, -f

  Ignore errors. For example, if a table for a text file does not exist, continue processing any remaining files. Without --force, mysqlimport exits if a table does not exist.

- --host=*host_name*, -h *host_name*

  Import data to the MySQL server on the given host. The default host is localhost.

- --ignore, -i

  See the description for the --replace option.

- --ignore-lines=*N*

  Ignore the first *N* lines of the data file.

- --lines-terminated-by=...

  This option has the same meaning as the corresponding clause for LOAD DATA INFILE. For example, to import Windows files that have lines terminated with carriage return/linefeed pairs, use --lines-terminated-by="\r\n". (You might have to double the backslashes, depending on the escaping conventions of your command interpreter.) See Section 13.2.6, "LOAD DATA INFILE Syntax".

- --local, -L

  Read input files locally from the client host.

- --lock-tables, -l

  Lock *all* tables for writing before processing any text files. This ensures that all tables are synchronized on the server.

- --login-path=*name*

  Read options from the named login path in the .mylogin.cnf login file. A "login path" is an option group that permits only a limited set of options: host, user, and password. Think of a login path as a set of values that indicate the server host and the credentials for authenticating with the server. To create the login file, use the mysql_config_editor utility. See Section 4.6.6, "mysql_config_editor — MySQL Configuration Utility".

- --low-priority

  Use LOW_PRIORITY when loading the table. This affects only storage engines that use only table-level locking (such as MyISAM, MEMORY, and MERGE).

- --no-defaults

  Do not read any option files. If program startup fails due to reading unknown options from an option file, --no-defaults can be used to prevent them from being read.

  The exception is that the .mylogin.cnf file, if it exists, is read in all cases. This permits passwords to be specified in a safer way than on the command line even when --no-defaults is used. (.mylogin.cnf is created by the mysql_config_editor utility. See Section 4.6.6, "mysql_config_editor — MySQL Configuration Utility".)

- --password[=*password*], -p[*password*]

  The password to use when connecting to the server. If you use the short option form (-p), you *cannot* have a space between the option and the password. If you omit the *password* value following the --password or -p option on the command line, mysqlimport prompts for one.

  Specifying a password on the command line should be considered insecure. See Section 6.1.2.1, "End-User Guidelines for Password Security". You can use an option file to avoid giving the password on the command line.

- --pipe, -W

  On Windows, connect to the server using a named pipe. This option applies only if the server supports named-pipe connections.

- --plugin-dir=*path*

  The directory in which to look for plugins. It may be necessary to specify this option if the --default-auth option is used to specify an authentication plugin but mysqlimport does not find it. See Section 6.3.8, "Pluggable Authentication".

- --port=*port_num*, -P *port_num*

  The TCP/IP port number to use for the connection.

- --print-defaults

  Print the program name and all options that it gets from option files.

- --protocol={TCP|SOCKET|PIPE|MEMORY}

  The connection protocol to use for connecting to the server. It is useful when the other connection parameters normally would cause a protocol to be used other than the one you want. For details on the permissible values, see Section 4.2.2, "Connecting to the MySQL Server".

- --replace, -r

  The --replace and --ignore options control handling of input rows that duplicate existing rows on unique key values. If you specify --replace, new rows replace existing rows that have the same unique key value. If you specify --ignore, input rows that duplicate an existing row on a unique key value are skipped. If you do not specify either option, an error occurs when a duplicate key value is found, and the rest of the text file is ignored.

- --secure-auth

  Do not send passwords to the server in old (pre-4.1) format. This prevents connections except for servers that use the newer password format. This option is enabled by default; use --skip-secure-auth to disable it. This option was added in MySQL 5.7.4.

  Note

  Passwords that use the pre-4.1 hashing method are less secure than passwords that use the native password hashing method and should be avoided. Pre-4.1 passwords are deprecated and support for them will be removed in a future MySQL release. For account upgrade instructions, see Section 6.3.9.3, "Migrating Away from Pre-4.1 Password Hashing and the mysql_old_password Plugin".

- --silent, -s

  Silent mode. Produce output only when errors occur.

- --socket=*path*, -S *path*

  For connections to localhost, the Unix socket file to use, or, on Windows, the name of the named pipe to use.

- --ssl*

  Options that begin with --ssl specify whether to connect to the server using SSL and indicate where to find SSL keys and certificates. See Section 6.3.11.4, "SSL Command Options".

- --user=*user_name*, -u *user_name*

  The MySQL user name to use when connecting to the server.

- --use-threads=*N*

  Load files in parallel using *N* threads.

- --verbose, -v

  Verbose mode. Print more information about what the program does.

- --version, -V

  Display version information and exit.

Here is a sample session that demonstrates use of mysqlimport:

```
shell> mysql -e 'CREATE TABLE imptest(id INT, n VARCHAR(30))' test
shell> ed
a
100     Max Sydow
101     Count Dracula
.
w imptest.txt
32
q
```

```
shell> od -c imptest.txt
0000000   1   0   0  \t   M   a   x       S   y   d   o   w  \n   1   0
0000020   1  \t   C   o   u   n   t       D   r   a   c   u   l   a  \n
0000040
shell> mysqlimport --local test imptest.txt
test.imptest: Records: 2  Deleted: 0  Skipped: 0  Warnings: 0
shell> mysql -e 'SELECT * FROM imptest' test
+------+---------------+
| id   | n             |
+------+---------------+
|  100 | Max Sydow     |
|  101 | Count Dracula |
+------+---------------+
```

4.5.6 mysqlshow — Display Database, Table, and Column Information

The mysqlshow client can be used to quickly see which databases exist, their tables, or a table's columns or indexes.

mysqlshow provides a command-line interface to several SQL SHOW statements. See Section 13.7.5, “SHOW Syntax”. The same information can be obtained by using those statements directly. For example, you can issue them from the mysql client program.

Invoke mysqlshow like this:

```
shell> mysqlshow [options] [db_name [tbl_name [col_name]]]
```

- If no database is given, a list of database names is shown.
- If no table is given, all matching tables in the database are shown.
- If no column is given, all matching columns and column types in the table are shown.

The output displays only the names of those databases, tables, or columns for which you have some privileges.

If the last argument contains shell or SQL wildcard characters (“*”, “?”, “%”, or “_”), only those names that are matched by the wildcard are shown. If a database name contains any underscores, those should be escaped with a backslash (some Unix shells require two) to get a list of the proper tables or columns. “*” and “?” characters are converted into SQL “%” and “_” wildcard characters. This might cause some confusion when you try to display the columns for a table with a “_” in the name, because in this case, mysqlshow shows you only the table names that match the pattern. This is easily fixed by adding an extra “%” last on the command line as a separate argument.

mysqlshow supports the following options, which can be specified on the command line or in the [mysqlshow] and [client] groups of an option file. For information about option files, see Section 4.2.3.3, “Using Option Files”.

Table 4.11 mysqlshow Options

Format	Option File	Description	Introduced
--bind-address=ip_address	bind-address	Use the specified network interface to connect to the MySQL Server	
--compress	compress	Compress all information sent between the client and the server	
--count	count	Show the number of rows per table	
--debug[=debug_options]	debug	Write a debugging log	

Format	Option File	Description	Introduced
--debug-check	debug-check	Print debugging information when the program exits	
--debug-info	debug-info	Print debugging information, memory and CPU statistics when the program exits	
--default-auth=plugin	default-auth=plugin	The authentication plugin to use	
--default-character-set=charset_name	default-character-set	Use charset_name as the default character set	
--defaults-extra-file=file_name		Read option file in addition to the usual option files	
--defaults-file=file_name		Read only the given option file	
--defaults-group-suffix=str		Option group suffix value	
--help		Display help message and exit	
--host=host_name	host	Connect to the MySQL server on the given host	
--keys	keys	Show table indexes	
--login-path=name		Read login path options from .mylogin.cnf	
--no-defaults		Do not read any option files	
--password[=password]	password	The password to use when connecting to the server	
--pipe		On Windows, connect to server using a named pipe	
--plugin-dir=path	plugin-dir=path	The directory where plugins are located	
--port=port_num	port	The TCP/IP port number to use for the connection	
--print-defaults		Print defaults	
--protocol=type	protocol	The connection protocol to use	
--secure-auth	secure-auth	Do not send passwords to the server in old (pre-4.1.1) format	5.7.4
--show-table-type		Show a column indicating the table type	
--socket=path	socket	For connections to localhost	
--ssl-ca=file_name	ssl-ca	The path to a file that contains a list of trusted SSL CAs	
--ssl-capath=dir_name	ssl-capath	The path to a directory that contains trusted SSL CA certificates in PEM format	
--ssl-cert=file_name	ssl-cert	The name of the SSL certificate file to use for establishing a secure connection	
--ssl-cipher=cipher_list	ssl-cipher	A list of allowable ciphers to use for SSL encryption	
--ssl-crl=file_name	ssl-crl	The path to a file that contains certificate revocation lists	
--ssl-crlpath=dir_name	ssl-crlpath	The path to a directory that contains certificate revocation list files	

Format	Option File	Description	Introduced
--ssl-key=file_name	ssl-key	The name of the SSL key file to use for establishing a secure connection	
--ssl-verify-server-cert	ssl-verify-server-cert	The server's Common Name value in its certificate is verified against the host name used when connecting to the server	
--status	status	Display extra information about each table	
--user=user_name,	user	MySQL user name to use when connecting to server	
--verbose		Verbose mode	
--version		Display version information and exit	

- `--help`, `-?`

  Display a help message and exit.

- `--bind-address=ip_address`

  On a computer having multiple network interfaces, this option can be used to select which interface is employed when connecting to the MySQL server.

- `--character-sets-dir=path`

  The directory where character sets are installed. See Section 10.5, “Character Set Configuration”.

- `--compress`, `-C`

  Compress all information sent between the client and the server if both support compression.

- `--count`

  Show the number of rows per table. This can be slow for non-`MyISAM` tables.

- `--debug[=debug_options]`, `-# [debug_options]`

  Write a debugging log. A typical *`debug_options`* string is `d:t:o,file_name`. The default is `d:t:o`.

- `--debug-check`

  Print some debugging information when the program exits.

- `--debug-info`

  Print debugging information and memory and CPU usage statistics when the program exits.

- `--default-character-set=charset_name`

  Use *`charset_name`* as the default character set. See Section 10.5, “Character Set Configuration”.

- `--default-auth=plugin`

  The client-side authentication plugin to use. See Section 6.3.8, “Pluggable Authentication”.

- `--defaults-extra-file=file_name`

Read this option file after the global option file but (on Unix) before the user option file. If the file does not exist or is otherwise inaccessible, an error occurs. *file_name* is interpreted relative to the current directory if given as a relative path name rather than a full path name.

- --defaults-file=*file_name*

  Use only the given option file. If the file does not exist or is otherwise inaccessible, an error occurs. *file_name* is interpreted relative to the current directory if given as a relative path name rather than a full path name.

- --defaults-group-suffix=*str*

  Read not only the usual option groups, but also groups with the usual names and a suffix of *str*. For example, mysqlshow normally reads the [client] and [mysqlshow] groups. If the --defaults-group-suffix=_other option is given, mysqlshow also reads the [client_other] and [mysqlshow_other] groups.

- --host=*host_name*, -h *host_name*

  Connect to the MySQL server on the given host.

- --keys, -k

  Show table indexes.

- --login-path=*name*

  Read options from the named login path in the .mylogin.cnf login file. A “login path” is an option group that permits only a limited set of options: host, user, and password. Think of a login path as a set of values that indicate the server host and the credentials for authenticating with the server. To create the login file, use the mysql_config_editor utility. See Section 4.6.6, “mysql_config_editor — MySQL Configuration Utility”.

- --no-defaults

  Do not read any option files. If program startup fails due to reading unknown options from an option file, --no-defaults can be used to prevent them from being read.

  The exception is that the .mylogin.cnf file, if it exists, is read in all cases. This permits passwords to be specified in a safer way than on the command line even when --no-defaults is used. (.mylogin.cnf is created by the mysql_config_editor utility. See Section 4.6.6, “mysql_config_editor — MySQL Configuration Utility”.)

- --password[=*password*], -p[*password*]

  The password to use when connecting to the server. If you use the short option form (-p), you *cannot* have a space between the option and the password. If you omit the *password* value following the --password or -p option on the command line, mysqlshow prompts for one.

  Specifying a password on the command line should be considered insecure. See Section 6.1.2.1, “End-User Guidelines for Password Security”. You can use an option file to avoid giving the password on the command line.

- --pipe, -W

  On Windows, connect to the server using a named pipe. This option applies only if the server supports named-pipe connections.

- --plugin-dir=*path*

  The directory in which to look for plugins. It may be necessary to specify this option if the --default-auth option is used to specify an authentication plugin but mysqlshow does not find it. See Section 6.3.8, “Pluggable Authentication”.

- --port=*port_num*, -P *port_num*

  The TCP/IP port number to use for the connection.

- --print-defaults

  Print the program name and all options that it gets from option files.

- --protocol={TCP|SOCKET|PIPE|MEMORY}

  The connection protocol to use for connecting to the server. It is useful when the other connection parameters normally would cause a protocol to be used other than the one you want. For details on the permissible values, see Section 4.2.2, “Connecting to the MySQL Server”.

- --secure-auth

  Do not send passwords to the server in old (pre-4.1) format. This prevents connections except for servers that use the newer password format. This option is enabled by default; use --skip-secure-auth to disable it. This option was added in MySQL 5.7.4.

  Note

  Passwords that use the pre-4.1 hashing method are less secure than passwords that use the native password hashing method and should be avoided. Pre-4.1 passwords are deprecated and support for them will be removed in a future MySQL release. For account upgrade instructions, see Section 6.3.9.3, “Migrating Away from Pre-4.1 Password Hashing and the mysql_old_password Plugin”.

- --show-table-type, -t

  Show a column indicating the table type, as in SHOW FULL TABLES. The type is BASE TABLE or VIEW.

- --socket=*path*, -S *path*

  For connections to localhost, the Unix socket file to use, or, on Windows, the name of the named pipe to use.

- --ssl*

  Options that begin with --ssl specify whether to connect to the server using SSL and indicate where to find SSL keys and certificates. See Section 6.3.11.4, “SSL Command Options”.

- --status, -i

  Display extra information about each table.

- --user=*user_name*, -u *user_name*

  The MySQL user name to use when connecting to the server.

- --verbose, -v

Verbose mode. Print more information about what the program does. This option can be used multiple times to increase the amount of information.

- --version, -V

  Display version information and exit.

4.5.7 mysqlslap — Load Emulation Client

mysqlslap is a diagnostic program designed to emulate client load for a MySQL server and to report the timing of each stage. It works as if multiple clients are accessing the server.

Invoke mysqlslap like this:

```
shell> mysqlslap [options]
```

Some options such as --create or --query enable you to specify a string containing an SQL statement or a file containing statements. If you specify a file, by default it must contain one statement per line. (That is, the implicit statement delimiter is the newline character.) Use the --delimiter option to specify a different delimiter, which enables you to specify statements that span multiple lines or place multiple statements on a single line. You cannot include comments in a file; mysqlslap does not understand them.

mysqlslap runs in three stages:

1. Create schema, table, and optionally any stored programs or data to use for the test. This stage uses a single client connection.

2. Run the load test. This stage can use many client connections.

3. Clean up (disconnect, drop table if specified). This stage uses a single client connection.

Examples:

Supply your own create and query SQL statements, with 50 clients querying and 200 selects for each (enter the command on a single line):

```
mysqlslap --delimiter=";"
  --create="CREATE TABLE a (b int);INSERT INTO a VALUES (23)"
  --query="SELECT * FROM a" --concurrency=50 --iterations=200
```

Let mysqlslap build the query SQL statement with a table of two INT columns and three VARCHAR columns. Use five clients querying 20 times each. Do not create the table or insert the data (that is, use the previous test's schema and data):

```
mysqlslap --concurrency=5 --iterations=20
  --number-int-cols=2 --number-char-cols=3
  --auto-generate-sql
```

Tell the program to load the create, insert, and query SQL statements from the specified files, where the create.sql file has multiple table creation statements delimited by ';' and multiple insert statements delimited by ';'. The --query file will have multiple queries delimited by ';'. Run all the load statements, then run all the queries in the query file with five clients (five times each):

```
mysqlslap --concurrency=5
  --iterations=5 --query=query.sql --create=create.sql
  --delimiter=";"
```

mysqlslap supports the following options, which can be specified on the command line or in the [mysqlslap] and [client] groups of an option file. For information about option files, see Section 4.2.3.3, “Using Option Files”.

Table 4.12 mysqlslap Options

Format	Option File	Description	Introduced
--auto-generate-sql	auto-generate-sql	Generate SQL statements automatically when they are not supplied in files or using command options	
--auto-generate-sql-add-autoincrement	auto-generate-sql-add-autoincrement	Add AUTO_INCREMENT column to automatically generated tables	
--auto-generate-sql-execute-number=#	auto-generate-sql-execute-number	Specify how many queries to generate automatically	
--auto-generate-sql-guid-primary	auto-generate-sql-guid-primary	Add a GUID-based primary key to automatically generated tables	
--auto-generate-sql-load-type=type	auto-generate-sql-load-type	Specify how many queries to generate automatically	
--auto-generate-sql-secondary-indexes=#	auto-generate-sql-secondary-indexes	Specify how many secondary indexes to add to automatically generated tables	
--auto-generate-sql-unique-query-number=#	auto-generate-sql-unique-query-number	How many different queries to generate for automatic tests.	
--auto-generate-sql-unique-write-number=#	auto-generate-sql-unique-write-number	How many different queries to generate for --auto-generate-sql-write-number	
--auto-generate-sql-write-number=#	auto-generate-sql-write-number	How many row inserts to perform on each thread	
--commit=#	commit	How many statements to execute before committing.	
--compress	compress	Compress all information sent between the client and the server	
--concurrency=#	concurrency	The number of clients to simulate when issuing the SELECT statement	
--create=value	create	The file or string containing the statement to use for creating the table	
--create-schema=value	create-schema	The schema in which to run the tests	
--csv=[file]	csv	Generate output in comma-separated values format	
--debug[=debug_options]	debug	Write a debugging log	
--debug-check	debug-check	Print debugging information when the program exits	
--debug-info	debug-info	Print debugging information, memory and CPU statistics when the program exits	

Format	Option File	Description	Introduced
--default-auth=plugin	default-auth=plugin	The authentication plugin to use	
--defaults-extra-file=file_name		Read option file in addition to the usual option files	
--defaults-file=file_name		Read only the given option file	
--defaults-group-suffix=str		Option group suffix value	
--delimiter=str	delimiter	The delimiter to use in SQL statements	
--detach=#	detach	Detach (close and reopen) each connection after each N statements	
--enable-cleartext-plugin	enable-cleartext-plugin	Enable cleartext authentication plugin	
--engine=engine_name	engine	The storage engine to use for creating the table	
--help		Display help message and exit	
--host=host_name	host	Connect to the MySQL server on the given host	
--iterations=#	iterations	The number of times to run the tests	
--login-path=name		Read login path options from .mylogin.cnf	
--no-defaults		Do not read any option files	
--no-drop	no-drop	Do not drop any schema created during the test run	
--number-char-cols=#	number-char-cols	The number of VARCHAR columns to use if --auto-generate-sql is specified	
--number-int-cols=#	number-int-cols	The number of INT columns to use if --auto-generate-sql is specified	
--number-of-queries=#	number-of-queries	Limit each client to approximately this number of queries	
--only-print	only-print	Do not connect to databases. mysqlslap only prints what it would have done	
--password[=password]	password	The password to use when connecting to the server	
--pipe		On Windows, connect to server using a named pipe	
--plugin-dir=path	plugin-dir=path	The directory where plugins are located	
--port=port_num	port	The TCP/IP port number to use for the connection	
--post-query=value	post-query	The file or string containing the statement to execute after the tests have completed	
--post-system=str	post-system	The string to execute using system() after the tests have completed	
--pre-query=value	pre-query	The file or string containing the statement to execute before running the tests	
--pre-system=str	pre-system	The string to execute using system() before running the tests	

Format	Option File	Description	Introduced
--print-defaults		Print defaults	
--protocol=type	protocol	The connection protocol to use	
--query=value	query	The file or string containing the SELECT statement to use for retrieving data	
--secure-auth	secure-auth	Do not send passwords to the server in old (pre-4.1.1) format	5.7.4
--silent	silent	Silent mode	
--socket=path	socket	For connections to localhost	
--ssl-ca=file_name	ssl-ca	The path to a file that contains a list of trusted SSL CAs	
--ssl-capath=dir_name	ssl-capath	The path to a directory that contains trusted SSL CA certificates in PEM format	
--ssl-cert=file_name	ssl-cert	The name of the SSL certificate file to use for establishing a secure connection	
--ssl-cipher=cipher_list	ssl-cipher	A list of allowable ciphers to use for SSL encryption	
--ssl-crl=file_name	ssl-crl	The path to a file that contains certificate revocation lists	
--ssl-crlpath=dir_name	ssl-crlpath	The path to a directory that contains certificate revocation list files	
--ssl-key=file_name	ssl-key	The name of the SSL key file to use for establishing a secure connection	
--ssl-verify-server-cert	ssl-verify-server-cert	The server's Common Name value in its certificate is verified against the host name used when connecting to the server	
--user=user_name,	user	MySQL user name to use when connecting to server	
--verbose		Verbose mode	
--version		Display version information and exit	

- `--help`, `-?`

  Display a help message and exit.

- `--auto-generate-sql`, `-a`

  Generate SQL statements automatically when they are not supplied in files or using command options.

- `--auto-generate-sql-add-autoincrement`

  Add an `AUTO_INCREMENT` column to automatically generated tables.

- `--auto-generate-sql-execute-number=N`

  Specify how many queries to generate automatically.

- `--auto-generate-sql-guid-primary`

Add a GUID-based primary key to automatically generated tables.

- --auto-generate-sql-load-type=*type*

  Specify the test load type. The permissible values are read (scan tables), write (insert into tables), key (read primary keys), update (update primary keys), or mixed (half inserts, half scanning selects). The default is mixed.

- --auto-generate-sql-secondary-indexes=*N*

  Specify how many secondary indexes to add to automatically generated tables. By default, none are added.

- --auto-generate-sql-unique-query-number=*N*

  How many different queries to generate for automatic tests. For example, if you run a key test that performs 1000 selects, you can use this option with a value of 1000 to run 1000 unique queries, or with a value of 50 to perform 50 different selects. The default is 10.

- --auto-generate-sql-unique-write-number=*N*

  How many different queries to generate for --auto-generate-sql-write-number. The default is 10.

- --auto-generate-sql-write-number=*N*

  How many row inserts to perform on each thread. The default is 100.

- --commit=*N*

  How many statements to execute before committing. The default is 0 (no commits are done).

- --compress, -C

  Compress all information sent between the client and the server if both support compression.

- --concurrency=*N*, -c *N*

  The number of clients to simulate when issuing the SELECT statement.

- --create=*value*

  The file or string containing the statement to use for creating the table.

- --create-schema=*value*

  The schema in which to run the tests.

  Note

  If the --auto-generate-sql option is also given, mysqlslap drops the schema at the end of the test run. To avoid this, use the --no-drop option as well.

- --csv[=*file_name*]

  Generate output in comma-separated values format. The output goes to the named file, or to the standard output if no file is given.

- `--debug[=debug_options]`, `-# [debug_options]`

  Write a debugging log. A typical `debug_options` string is `d:t:o,file_name`. The default is `d:t:o,/tmp/mysqlslap.trace`.

- `--debug-check`

  Print some debugging information when the program exits.

- `--debug-info`, `-T`

  Print debugging information and memory and CPU usage statistics when the program exits.

- `--default-auth=plugin`

  The client-side authentication plugin to use. See Section 6.3.8, “Pluggable Authentication”.

- `--defaults-extra-file=file_name`

  Read this option file after the global option file but (on Unix) before the user option file. If the file does not exist or is otherwise inaccessible, an error occurs. `file_name` is interpreted relative to the current directory if given as a relative path name rather than a full path name.

- `--defaults-file=file_name`

  Use only the given option file. If the file does not exist or is otherwise inaccessible, an error occurs. `file_name` is interpreted relative to the current directory if given as a relative path name rather than a full path name.

- `--defaults-group-suffix=str`

  Read not only the usual option groups, but also groups with the usual names and a suffix of `str`. For example, `mysqlslap` normally reads the `[client]` and `[mysqlslap]` groups. If the `--defaults-group-suffix=_other` option is given, `mysqlslap` also reads the `[client_other]` and `[mysqlslap_other]` groups.

- `--delimiter=str`, `-F str`

  The delimiter to use in SQL statements supplied in files or using command options.

- `--detach=N`

  Detach (close and reopen) each connection after each `N` statements. The default is 0 (connections are not detached).

- `--enable-cleartext-plugin`

  Enable the `mysql_clear_password` cleartext authentication plugin. (See Section 6.3.9.5, “The Cleartext Client-Side Authentication Plugin”.)

- `--engine=engine_name`, `-e engine_name`

  The storage engine to use for creating tables.

- `--host=host_name`, `-h host_name`

  Connect to the MySQL server on the given host.

- `--iterations=N`, `-i N`

The number of times to run the tests.

- --login-path=*name*

  Read options from the named login path in the .mylogin.cnf login file. A “login path” is an option group that permits only a limited set of options: host, user, and password. Think of a login path as a set of values that indicate the server host and the credentials for authenticating with the server. To create the login file, use the mysql_config_editor utility. See Section 4.6.6, “mysql_config_editor — MySQL Configuration Utility”.

- --no-drop

  Prevent mysqlslap from dropping any schema it creates during the test run.

- --no-defaults

  Do not read any option files. If program startup fails due to reading unknown options from an option file, --no-defaults can be used to prevent them from being read.

  The exception is that the .mylogin.cnf file, if it exists, is read in all cases. This permits passwords to be specified in a safer way than on the command line even when --no-defaults is used. (.mylogin.cnf is created by the mysql_config_editor utility. See Section 4.6.6, “mysql_config_editor — MySQL Configuration Utility”.)

- --number-char-cols=*N*, -x *N*

  The number of VARCHAR columns to use if --auto-generate-sql is specified.

- --number-int-cols=*N*, -y *N*

  The number of INT columns to use if --auto-generate-sql is specified.

- --number-of-queries=*N*

  Limit each client to approximately this many queries. Query counting takes into account the statement delimiter. For example, if you invoke mysqlslap as follows, the ; delimiter is recognized so that each instance of the query string counts as two queries. As a result, 5 rows (not 10) are inserted.

```
shell> mysqlslap --delimiter=";" --number-of-queries=10
         --query="use test;insert into t values(null)"
```

- --only-print

  Do not connect to databases. mysqlslap only prints what it would have done.

- --password[=*password*], -p[*password*]

  The password to use when connecting to the server. If you use the short option form (-p), you *cannot* have a space between the option and the password. If you omit the *password* value following the --password or -p option on the command line, mysqlslap prompts for one.

  Specifying a password on the command line should be considered insecure. See Section 6.1.2.1, “End-User Guidelines for Password Security”. You can use an option file to avoid giving the password on the command line.

- --pipe, -W

On Windows, connect to the server using a named pipe. This option applies only if the server supports named-pipe connections.

- `--plugin-dir=path`

  The directory in which to look for plugins. It may be necessary to specify this option if the `--default-auth` option is used to specify an authentication plugin but `mysqlslap` does not find it. See Section 6.3.8, “Pluggable Authentication”.

- `--port=port_num`, `-P port_num`

  The TCP/IP port number to use for the connection.

- `--post-query=value`

  The file or string containing the statement to execute after the tests have completed. This execution is not counted for timing purposes.

- `--post-system=str`

  The string to execute using `system()` after the tests have completed. This execution is not counted for timing purposes.

- `--pre-query=value`

  The file or string containing the statement to execute before running the tests. This execution is not counted for timing purposes.

- `--pre-system=str`

  The string to execute using `system()` before running the tests. This execution is not counted for timing purposes.

- `--print-defaults`

  Print the program name and all options that it gets from option files.

- `--protocol={TCP|SOCKET|PIPE|MEMORY}`

  The connection protocol to use for connecting to the server. It is useful when the other connection parameters normally would cause a protocol to be used other than the one you want. For details on the permissible values, see Section 4.2.2, “Connecting to the MySQL Server”.

- `--query=value`, `-q value`

  The file or string containing the `SELECT` statement to use for retrieving data.

- `--secure-auth`

  Do not send passwords to the server in old (pre-4.1) format. This prevents connections except for servers that use the newer password format. This option is enabled by default; use `--skip-secure-auth` to disable it. This option was added in MySQL 5.7.4.

  Note

  Passwords that use the pre-4.1 hashing method are less secure than passwords that use the native password hashing method and should be avoided. Pre-4.1 passwords are deprecated and support for them will be removed in a future

> MySQL release. For account upgrade instructions, see Section 6.3.9.3, “Migrating Away from Pre-4.1 Password Hashing and the mysql_old_password Plugin”.

- `--shared-memory-base-name=name`

  On Windows, the shared-memory name to use, for connections made using shared memory to a local server. This option applies only if the server supports shared-memory connections.

- `--silent`, `-s`

  Silent mode. No output.

- `--socket=path`, `-S path`

  For connections to `localhost`, the Unix socket file to use, or, on Windows, the name of the named pipe to use.

- `--ssl*`

  Options that begin with `--ssl` specify whether to connect to the server using SSL and indicate where to find SSL keys and certificates. See Section 6.3.11.4, “SSL Command Options”.

- `--user=user_name`, `-u user_name`

  The MySQL user name to use when connecting to the server.

- `--verbose`, `-v`

  Verbose mode. Print more information about what the program does. This option can be used multiple times to increase the amount of information.

- `--version`, `-V`

  Display version information and exit.

4.6 MySQL Administrative and Utility Programs

This section describes administrative programs and programs that perform miscellaneous utility operations.

4.6.1 innochecksum — Offline InnoDB File Checksum Utility

`innochecksum` prints checksums for `InnoDB` files. This tool reads an `InnoDB` tablespace file, calculates the checksum for each page, compares the calculated checksum to the stored checksum, and reports mismatches, which indicate damaged pages. It was originally developed to speed up verifying the integrity of tablespace files after power outages but can also be used after file copies. Because checksum mismatches will cause `InnoDB` to deliberately shut down a running server, it can be preferable to use this tool rather than waiting for a server in production usage to encounter the damaged pages. As of MySQL 5.7.2, `innochecksum` supports files greater than 2GB in size. Previously, `innochecksum` only supported files up to 2GB in size.

`innochecksum` cannot be used on tablespace files that the server already has open. For such files, you should use `CHECK TABLE` to check tables within the tablespace. Attempting to run `innochecksum` on a tablespace that the server already has open will result in an “`Unable to lock file`” error.

If checksum mismatches are found, you would normally restore the tablespace from backup or start the server and attempt to use `mysqldump` to make a backup of the tables within the tablespace.

Invoke `innochecksum` like this:

```
shell> innochecksum [options] file_name
```

innochecksum Options

innochecksum supports the following options. For options that refer to page numbers, the numbers are zero-based.

- --help, -?

  Displays command line help. Example usage:

  ```
  shell> innochecksum --help
  ```

- --info, -I

  Synonym for --help. Displays command line help. Example usage:

  ```
  shell> innochecksum --info
  ```

- --version, -V

  Displays version information. Example usage:

  ```
  shell> innochecksum --version
  ```

- --verbose, -v

  Verbose mode; prints a progress indicator to the log file every five seconds. In order for the progress indicator to be printed, the log file must be specified using the --log option. To turn on verbose mode, run:

  ```
  shell> innochecksum --verbose
  ```

  To turn off verbose mode, run:

  ```
  shell> innochecksum --verbose=FALSE
  ```

  The --verbose option and --log option can be specified at the same time. For example:

  ```
  shell> innochecksum --verbose --log=/var/lib/mysql/test/logtest.txt
  ```

  To locate the progress indicator information in the log file, you can perform the following search:

  ```
  shell> cat ./logtest.txt | grep -i "okay"
  ```

  The progress indicator information in the log file appears similar to the following:

  ```
  page 1663 okay: 2.863% done
  page 8447 okay: 14.537% done
  page 13695 okay: 23.568% done
  page 18815 okay: 32.379% done
  page 23039 okay: 39.648% done
  page 28351 okay: 48.789% done
  ```

```
page 33023 okay: 56.828% done
page 37951 okay: 65.308% done
page 44095 okay: 75.881% done
page 49407 okay: 85.022% done
page 54463 okay: 93.722% done
...
```

- `--count`, `-c`

  Print a count of the number of pages in the file and exit. Example usage:

  ```
  shell> innochecksum --count ../data/test/tab1.ibd
  ```

- `--start-page=num`, `-s num`

  Start at this page number. Example usage:

  ```
  shell> innochecksum --start-page=600 ../data/test/tab1.ibd
  ```

  or:

  ```
  shell> innochecksum -s 600 ../data/test/tab1.ibd
  ```

- `--end-page=num`, `-e num`

  End at this page number. Example usage:

  ```
  shell> innochecksum --end-page=700 ../data/test/tab1.ibd
  ```

  or:

  ```
  shell> innochecksum --p 700 ../data/test/tab1.ibd
  ```

- `--page=num`, `-p num`

  Check only this page number. Example usage:

  ```
  shell> innochecksum --page=701 ../data/test/tab1.ibd
  ```

- `--strict-check`, `-C`

  Specify a strict checksum algorithm. Options include `innodb`, `crc32`, and `none`.

  In this example, the `innodb` checksum algorithm is specified:

  ```
  shell> innochecksum --strict-check=innodb ../data/test/tab1.ibd
  ```

  In this example, the `crc32` checksum algorithm is specified:

  ```
  shell> innochecksum -C crc32 ../data/test/tab1.ibd
  ```

  The following conditions apply:

  - If you do not specify the `--strict-check` option, `innochecksum` validates against `innodb`, `crc32` and `none`.

  - If you specify the `none` option, only checksums generated by `none` are allowed.
  - If you specify the `innodb` option, only checksums generated by `innodb` are allowed.
  - If you specify the `crc32` option, only checksums generated by `crc32` are allowed.

- `--no-check`, `-n`

  Ignore the checksum verification when rewriting a checksum. This option may only be used with the `innochecksum --write` option. If the `--write` option is not specified, `innochecksum` will terminate.

  In this example, an `innodb` checksum is rewritten to replace an invalid checksum:

  ```
  shell> innochecksum --no-check --write innodb ../data/test/tab1.ibd
  ```

- `--allow-mismatches`, `-a`

  The maximum number of checksum mismatches allowed before `innochecksum` terminates. The default setting is 0. If `--allow-mismatches=`*`N`*, where *`N`*`>=0`, *`N`* mismatches are permitted and `innochecksum` terminates at *`N`*`+1`. When `--allow-mismatches` is set to 0, `innochecksum` terminates on the first checksum mismatch.

  In this example, an existing `innodb` checksum is rewritten to set `--allow-mismatches` to 1.

  ```
  shell> innochecksum --allow-mismatches=1 --write innodb ../data/test/tab1.ibd
  ```

  With `--allow-mismatches` set to 1, if there is a mismatch at page 600 and another at page 700 on a file with 1000 pages, the checksum is updated for pages 0-599 and 601-699. Because `--allow-mismatches` is set to 1, the checksum tolerates the first mismatch and terminates on the second mismatch, leaving page 600 and pages 700-999 unchanged.

- `--write=`*`name`*, `-w` *`num`*

  Rewrite a checksum. When rewriting an invalid checksum, the `--no-check` option must be used together with the `--write` option. The `--no-check` option tells `innochecksum` to ignore verification of the invalid checksum. You do not have to specify the `--no-check` option if the current checksum is valid.

  An algorithm must be specified when using the `--write` option. Possible values for the `--write` option are:

  - `innodb`: A checksum calculated in software, using the original algorithm from `InnoDB`.
  - `crc32`: A checksum calculated using the `crc32` algorithm, possibly done with a hardware assist.
  - `none`: A constant number.

  The `--write` option rewrites entire pages to disk. If the new checksum is identical to the existing checksum, the new checksum is not written to disk in order to minimize I/O.

  `innochecksum` obtains an exclusive lock when the `--write` option is used.

  In this example, a `crc32` checksum is written for `tab1.ibd`:

  ```
  shell> innochecksum -w crc32 ../data/test/tab1.ibd
  ```

In this example, a crc32 checksum is rewritten to replace an invalid crc32 checksum:

```
shell> innochecksum --no-check --write crc32 ../data/test/tab1.ibd
```

- --page-type-summary, -S

  Display a count of each page type in a tablespace. Example usage:

```
shell> innochecksum --page-type-summary ../data/test/tab1.ibd
```

  Sample output for --page-type-summary:

```
File::../data/test/tab1.ibd
================PAGE TYPE SUMMARY==============
#PAGE_COUNT PAGE_TYPE
===============================================
       2        Index page
       0        Undo log page
       1        Inode page
       0        Insert buffer free list page
       2        Freshly allocated page
       1        Insert buffer bitmap
       0        System page
       0        Transaction system page
       1        File Space Header
       0        Extent descriptor page
       0        BLOB page
       0        Compressed BLOB page
       0        Other type of page
===============================================
Additional information:
Undo page type: 0 insert, 0 update, 0 other
Undo page state: 0 active, 0 cached, 0 to_free, 0 to_purge, 0 prepared, 0 other
```

- --page-type-dump, -D

  Dump the page type information for each page in a tablespace to stderr or stdout. Example usage:

```
shell> innochecksum --page-type-dump=/tmp/a.txt ../data/test/tab1.ibd
```

- --log, -l

  Log output for the innochecksum tool. A log file name must be provided. Log output contains checksum values for each tablespace page. For uncompressed tables, LSN values are also provided. The --log replaces the --debug option, which was available in earlier releases. Example usage:

```
shell> innochecksum --log=/tmp/log.txt ../data/test/tab1.ibd
```

  or:

```
shell> innochecksum -l /tmp/log.txt ../data/test/tab1.ibd
```

- “-” option.

Specify the "-" option to read from standard input. If the "-" option is missing when "read from standard in" is expected, innochecksum will output innochecksum usage information indicating that the "-" option was omitted. Example usages:

```
shell> cat t1.ibd | innochecksum  -
```

In this example, innochecksum writes the crc32 checksum algorithm to a.ibd without changing the original t1.ibd file.

```
shell> cat t1.ibd | innochecksum --write=crc32 -  > a.ibd
```

Running innochecksum on Multiple User-defined Tablespace Files

The following examples demonstrate how to run innochecksum on multiple user-defined tablespace files (.ibd files).

Run innochecksum for all tablespace (.ibd) files in the "test" database:

```
shell> innochecksum ./data/test/*.ibd
```

Run innochecksum for all tablespace files (.ibd files) that have a file name starting with "t":

```
shell> innochecksum ./data/test/t*.ibd
```

Run innochecksum for all tablespace files (.ibd files) in the data directory:

```
shell> innochecksum ./data/*/*.ibd
```

Note

Running innochecksum on multiple user-defined tablespace files is not supported on Windows operating systems, as Windows shells such as cmd.exe do not support glob pattern expansion. On Windows systems, innochecksum must be run separately for each user-defined tablespace file. For example:

```
cmd> innochecksum.exe t1.ibd
cmd> innochecksum.exe t2.ibd
cmd> innochecksum.exe t3.ibd
```

Running innochecksum on Multiple System Tablespace Files

By default, there is only one InnoDB system tablespace file (ibdata1) but multiple files for the system tablespace can be defined using the innodb_data_file_path option. In the following example, three files for the system tablespace are defined using the innodb_data_file_path option: ibdata1, ibdata2, and ibdata3.

```
shell> ./bin/mysqld --no-defaults --innodb-data-file-path="ibdata1:10M;ibdata2:10M;ibdata3:10M:autoextend"
```

The three files (ibdata1, ibdata2, and ibdata3) form one logical system tablespace. To run innochecksum on multiple files that form one logical system tablespace, innochecksum requires the "-" option to read tablespace files in from standard input, which is equivalent to concatenating multiple files to create one single file. For the example provided above, the following innochecksum command would be used:

```
shell> cat ibdata* | innochecksum -
```

Refer to the innochecksum options information for more information about the “-” option.

Note

Running innochecksum on multiple files in the same tablespace is not supported on Windows operating systems, as Windows shells such as cmd.exe do not support glob pattern expansion. On Windows systems, innochecksum must be run separately for each system tablespace file. For example:

```
cmd> innochecksum.exe ibdata1
cmd> innochecksum.exe ibdata2
cmd> innochecksum.exe ibdata3
```

4.6.2 myisam_ftdump — Display Full-Text Index information

myisam_ftdump displays information about FULLTEXT indexes in MyISAM tables. It reads the MyISAM index file directly, so it must be run on the server host where the table is located. Before using myisam_ftdump, be sure to issue a FLUSH TABLES statement first if the server is running.

myisam_ftdump scans and dumps the entire index, which is not particularly fast. On the other hand, the distribution of words changes infrequently, so it need not be run often.

Invoke myisam_ftdump like this:

```
shell> myisam_ftdump [options] tbl_name index_num
```

The *tbl_name* argument should be the name of a MyISAM table. You can also specify a table by naming its index file (the file with the .MYI suffix). If you do not invoke myisam_ftdump in the directory where the table files are located, the table or index file name must be preceded by the path name to the table's database directory. Index numbers begin with 0.

Example: Suppose that the test database contains a table named mytexttable that has the following definition:

```
CREATE TABLE mytexttable
(
  id   INT NOT NULL,
  txt  TEXT NOT NULL,
  PRIMARY KEY (id),
  FULLTEXT (txt)
) ENGINE=MyISAM;
```

The index on id is index 0 and the FULLTEXT index on txt is index 1. If your working directory is the test database directory, invoke myisam_ftdump as follows:

```
shell> myisam_ftdump mytexttable 1
```

If the path name to the test database directory is /usr/local/mysql/data/test, you can also specify the table name argument using that path name. This is useful if you do not invoke myisam_ftdump in the database directory:

```
shell> myisam_ftdump /usr/local/mysql/data/test/mytexttable 1
```

You can use myisam_ftdump to generate a list of index entries in order of frequency of occurrence like this:

```
shell> myisam_ftdump -c mytexttable 1 | sort -r
```

myisam_ftdump supports the following options:

- --help, -h -?

  Display a help message and exit.

- --count, -c

  Calculate per-word statistics (counts and global weights).

- --dump, -d

  Dump the index, including data offsets and word weights.

- --length, -l

  Report the length distribution.

- --stats, -s

  Report global index statistics. This is the default operation if no other operation is specified.

- --verbose, -v

  Verbose mode. Print more output about what the program does.

4.6.3 myisamchk — MyISAM Table-Maintenance Utility

The myisamchk utility gets information about your database tables or checks, repairs, or optimizes them. myisamchk works with MyISAM tables (tables that have .MYD and .MYI files for storing data and indexes).

You can also use the CHECK TABLE and REPAIR TABLE statements to check and repair MyISAM tables. See Section 13.7.2.2, “CHECK TABLE Syntax”, and Section 13.7.2.5, “REPAIR TABLE Syntax”.

The use of myisamchk with partitioned tables is not supported.

Caution

It is best to make a backup of a table before performing a table repair operation; under some circumstances the operation might cause data loss. Possible causes include but are not limited to file system errors.

Invoke myisamchk like this:

```
shell> myisamchk [options] tbl_name ...
```

The *options* specify what you want myisamchk to do. They are described in the following sections. You can also get a list of options by invoking myisamchk --help.

With no options, myisamchk simply checks your table as the default operation. To get more information or to tell myisamchk to take corrective action, specify options as described in the following discussion.

tbl_name is the database table you want to check or repair. If you run myisamchk somewhere other than in the database directory, you must specify the path to the database directory, because myisamchk has no idea where the database is located. In fact, myisamchk does not actually care whether the files you are working on are located in a database directory. You can copy the files that correspond to a database table into some other location and perform recovery operations on them there.

You can name several tables on the myisamchk command line if you wish. You can also specify a table by naming its index file (the file with the .MYI suffix). This enables you to specify all tables in a directory by using the pattern *.MYI. For example, if you are in a database directory, you can check all the MyISAM tables in that directory like this:

```
shell> myisamchk *.MYI
```

If you are not in the database directory, you can check all the tables there by specifying the path to the directory:

```
shell> myisamchk /path/to/database_dir/*.MYI
```

You can even check all tables in all databases by specifying a wildcard with the path to the MySQL data directory:

```
shell> myisamchk /path/to/datadir/*/*.MYI
```

The recommended way to quickly check all MyISAM tables is:

```
shell> myisamchk --silent --fast /path/to/datadir/*/*.MYI
```

If you want to check all MyISAM tables and repair any that are corrupted, you can use the following command:

```
shell> myisamchk --silent --force --fast --update-state \
          --key_buffer_size=64M --myisam_sort_buffer_size=64M \
          --read_buffer_size=1M --write_buffer_size=1M \
          /path/to/datadir/*/*.MYI
```

This command assumes that you have more than 64MB free. For more information about memory allocation with myisamchk, see Section 4.6.3.6, “myisamchk Memory Usage”.

For additional information about using myisamchk, see Section 7.6, “MyISAM Table Maintenance and Crash Recovery”.

⚠ **Important**

You must ensure that no other program is using the tables while you are running myisamchk. The most effective means of doing so is to shut down the MySQL server while running myisamchk, or to lock all tables that myisamchk is being used on.

Otherwise, when you run myisamchk, it may display the following error message:

```
warning: clients are using or haven't closed the table properly
```

This means that you are trying to check a table that has been updated by another program (such as the mysqld server) that hasn't yet closed the file or that has died

> without closing the file properly, which can sometimes lead to the corruption of one or more MyISAM tables.
>
> If mysqld is running, you must force it to flush any table modifications that are still buffered in memory by using FLUSH TABLES. You should then ensure that no one is using the tables while you are running myisamchk
>
> However, the easiest way to avoid this problem is to use CHECK TABLE instead of myisamchk to check tables. See Section 13.7.2.2, "CHECK TABLE Syntax".

myisamchk supports the following options, which can be specified on the command line or in the [myisamchk] group of an option file. For information about option files, see Section 4.2.3.3, "Using Option Files".

Table 4.13 myisamchk Options

Format	Option File	Description
--analyze	analyze	Analyze the distribution of key values
--backup	backup	Make a backup of the .MYD file as file_name-time.BAK
--block-search=offset	block-search	Find the record that a block at the given offset belongs to
--check	check	Check the table for errors
--check-only-changed	check-only-changed	Check only tables that have changed since the last check
--correct-checksum	correct-checksum	Correct the checksum information for the table
--data-file-length=len	data-file-length	Maximum length of the data file (when re-creating data file when it is full)
--debug[=debug_options]	debug	Write a debugging log
decode_bits=#	decode_bits	Decode_bits
--defaults-extra-file=file_name		Read option file in addition to the usual option files
--defaults-file=file_name		Read only the given option file
--defaults-group-suffix=str		Option group suffix value
--description	description	Print some descriptive information about the table
--extend-check	extend-check	Do very thorough table check or repair that tries to recover every possible row from the data file
--fast	fast	Check only tables that haven't been closed properly
--force	force	Do a repair operation automatically if myisamchk finds any errors in the table
--force	force-recover	Overwrite old temporary files. For use with the -r or -o option
ft_max_word_len=#	ft_max_word_len	Maximum word length for FULLTEXT indexes
ft_min_word_len=#	ft_min_word_len	Minimum word length for FULLTEXT indexes
ft_stopword_file=value	ft_stopword_file	Use stopwords from this file instead of built-in list
--HELP		Display help message and exit
--help		Display help message and exit

Format	Option File	Description
--information	information	Print informational statistics about the table that is checked
key_buffer_size=#	key_buffer_size	The size of the buffer used for index blocks for MyISAM tables
--keys-used=val	keys-used	A bit-value that indicates which indexes to update
--max-record-length=len	max-record-length	Skip rows larger than the given length if myisamchk cannot allocate memory to hold them
--medium-check	medium-check	Do a check that is faster than an --extend-check operation
myisam_block_size=#	myisam_block_size	Block size to be used for MyISAM index pages
myisam_sort_buffer_size=#	myisam_sort_buffer_size	The buffer that is allocated when sorting the index when doing a REPAIR or when creating indexes with CREATE INDEX or ALTER TABLE
--no-defaults		Do not read any option files
--parallel-recover	parallel-recover	Uses the same technique as -r and -n, but creates all the keys in parallel, using different threads (beta)
--print-defaults		Print defaults
--quick	quick	Achieve a faster repair by not modifying the data file.
read_buffer_size=#	read_buffer_size	Each thread that does a sequential scan allocates a buffer of this size for each table it scans
--read-only	read-only	Don't mark the table as checked
--recover	recover	Do a repair that can fix almost any problem except unique keys that aren't unique
--safe-recover	safe-recover	Do a repair using an old recovery method that reads through all rows in order and updates all index trees based on the rows found
--set-auto-increment[=value]	set-auto-increment	Force AUTO_INCREMENT numbering for new records to start at the given value
--set-collation=name	set-collation	Specify the collation to use for sorting table indexes
--silent	silent	Silent mode
sort_buffer_size=#	sort_buffer_size	The buffer that is allocated when sorting the index when doing a REPAIR or when creating indexes with CREATE INDEX or ALTER TABLE
--sort-index	sort-index	Sort the index tree blocks in high-low order
sort_key_blocks=#	sort_key_blocks	sort_key_blocks
--sort-records=#	sort-records	Sort records according to a particular index
--sort-recover	sort-recover	Force myisamchk to use sorting to resolve the keys even if the temporary files would be very large
stats_method=value	stats_method	Specifies how MyISAM index statistics collection code should treat NULLs
--tmpdir=path	tmpdir	Path of the directory to be used for storing temporary files
--unpack	unpack	Unpack a table that was packed with myisampack
--update-state	update-state	Store information in the .MYI file to indicate when the table was checked and whether the table crashed

Format	Option File	Description
--verbose		Verbose mode
--version		Display version information and exit
write_buffer_size=#	write_buffer_size	Write buffer size

4.6.3.1 myisamchk General Options

The options described in this section can be used for any type of table maintenance operation performed by `myisamchk`. The sections following this one describe options that pertain only to specific operations, such as table checking or repairing.

- `--help`, `-?`

  Display a help message and exit. Options are grouped by type of operation.

- `--HELP`, `-H`

  Display a help message and exit. Options are presented in a single list.

- `--debug=debug_options`, `-# debug_options`

  Write a debugging log. A typical *`debug_options`* string is `d:t:o,file_name`. The default is `d:t:o,/tmp/myisamchk.trace`.

- `--defaults-extra-file=file_name`

  Read this option file after the global option file but (on Unix) before the user option file. If the file does not exist or is otherwise inaccessible, an error occurs. *`file_name`* is interpreted relative to the current directory if given as a relative path name rather than a full path name.

- `--defaults-file=file_name`

  Use only the given option file. If the file does not exist or is otherwise inaccessible, an error occurs. *`file_name`* is interpreted relative to the current directory if given as a relative path name rather than a full path name.

- `--defaults-group-suffix=str`

  Read not only the usual option groups, but also groups with the usual names and a suffix of *`str`*. For example, `myisamchk` normally reads the `[myisamchk]` group. If the `--defaults-group-suffix=_other` option is given, `myisamchk` also reads the `[myisamchk_other]` group.

- `--no-defaults`

  Do not read any option files. If program startup fails due to reading unknown options from an option file, `--no-defaults` can be used to prevent them from being read.

  The exception is that the `.mylogin.cnf` file, if it exists, is read in all cases. This permits passwords to be specified in a safer way than on the command line even when `--no-defaults` is used. (`.mylogin.cnf` is created by the `mysql_config_editor` utility. See Section 4.6.6, “`mysql_config_editor` — MySQL Configuration Utility”.)

- `--print-defaults`

  Print the program name and all options that it gets from option files.

- `--silent`, `-s`

Silent mode. Write output only when errors occur. You can use -s twice (-ss) to make myisamchk very silent.

- --verbose, -v

  Verbose mode. Print more information about what the program does. This can be used with -d and -e. Use -v multiple times (-vv, -vvv) for even more output.

- --version, -V

  Display version information and exit.

- --wait, -w

  Instead of terminating with an error if the table is locked, wait until the table is unlocked before continuing. If you are running mysqld with external locking disabled, the table can be locked only by another myisamchk command.

You can also set the following variables by using --*var_name*=*value* syntax:

Variable	Default Value
decode_bits	9
ft_max_word_len	version-dependent
ft_min_word_len	4
ft_stopword_file	built-in list
key_buffer_size	523264
myisam_block_size	1024
myisam_sort_key_blocks	16
read_buffer_size	262136
sort_buffer_size	2097144
sort_key_blocks	16
stats_method	nulls_unequal
write_buffer_size	262136

The possible myisamchk variables and their default values can be examined with myisamchk --help:

myisam_sort_buffer_size is used when the keys are repaired by sorting keys, which is the normal case when you use --recover. sort_buffer_size is a deprecated synonym for myisam_sort_buffer_size.

key_buffer_size is used when you are checking the table with --extend-check or when the keys are repaired by inserting keys row by row into the table (like when doing normal inserts). Repairing through the key buffer is used in the following cases:

- You use --safe-recover.
- The temporary files needed to sort the keys would be more than twice as big as when creating the key file directly. This is often the case when you have large key values for CHAR, VARCHAR, or TEXT columns, because the sort operation needs to store the complete key values as it proceeds. If you have lots of temporary space and you can force myisamchk to repair by sorting, you can use the --sort-recover option.

Repairing through the key buffer takes much less disk space than using sorting, but is also much slower.

If you want a faster repair, set the key_buffer_size and myisam_sort_buffer_size variables to about 25% of your available memory. You can set both variables to large values, because only one of them is used at a time.

myisam_block_size is the size used for index blocks.

stats_method influences how NULL values are treated for index statistics collection when the --analyze option is given. It acts like the myisam_stats_method system variable. For more information, see the description of myisam_stats_method in Section 5.1.4, "Server System Variables", and Section 8.3.7, "InnoDB and MyISAM Index Statistics Collection".

ft_min_word_len and ft_max_word_len indicate the minimum and maximum word length for FULLTEXT indexes on MyISAM tables. ft_stopword_file names the stopword file. These need to be set under the following circumstances.

If you use myisamchk to perform an operation that modifies table indexes (such as repair or analyze), the FULLTEXT indexes are rebuilt using the default full-text parameter values for minimum and maximum word length and the stopword file unless you specify otherwise. This can result in queries failing.

The problem occurs because these parameters are known only by the server. They are not stored in MyISAM index files. To avoid the problem if you have modified the minimum or maximum word length or the stopword file in the server, specify the same ft_min_word_len, ft_max_word_len, and ft_stopword_file values to myisamchk that you use for mysqld. For example, if you have set the minimum word length to 3, you can repair a table with myisamchk like this:

```
shell> myisamchk --recover --ft_min_word_len=3 tbl_name.MYI
```

To ensure that myisamchk and the server use the same values for full-text parameters, you can place each one in both the [mysqld] and [myisamchk] sections of an option file:

```
[mysqld]
ft_min_word_len=3

[myisamchk]
ft_min_word_len=3
```

An alternative to using myisamchk is to use the REPAIR TABLE, ANALYZE TABLE, OPTIMIZE TABLE, or ALTER TABLE. These statements are performed by the server, which knows the proper full-text parameter values to use.

4.6.3.2 myisamchk Check Options

myisamchk supports the following options for table checking operations:

- --check, -c

  Check the table for errors. This is the default operation if you specify no option that selects an operation type explicitly.

- --check-only-changed, -C

  Check only tables that have changed since the last check.

- --extend-check, -e

Check the table very thoroughly. This is quite slow if the table has many indexes. This option should only be used in extreme cases. Normally, myisamchk or myisamchk --medium-check should be able to determine whether there are any errors in the table.

If you are using --extend-check and have plenty of memory, setting the key_buffer_size variable to a large value helps the repair operation run faster.

See also the description of this option under table repair options.

For a description of the output format, see Section 4.6.3.5, “Obtaining Table Information with myisamchk”.

- --fast, -F

  Check only tables that haven't been closed properly.

- --force, -f

  Do a repair operation automatically if myisamchk finds any errors in the table. The repair type is the same as that specified with the --recover or -r option.

- --information, -i

  Print informational statistics about the table that is checked.

- --medium-check, -m

  Do a check that is faster than an --extend-check operation. This finds only 99.99% of all errors, which should be good enough in most cases.

- --read-only, -T

  Do not mark the table as checked. This is useful if you use myisamchk to check a table that is in use by some other application that does not use locking, such as mysqld when run with external locking disabled.

- --update-state, -U

  Store information in the .MYI file to indicate when the table was checked and whether the table crashed. This should be used to get full benefit of the --check-only-changed option, but you shouldn't use this option if the mysqld server is using the table and you are running it with external locking disabled.

4.6.3.3 myisamchk Repair Options

myisamchk supports the following options for table repair operations (operations performed when an option such as --recover or --safe-recover is given):

- --backup, -B

  Make a backup of the .MYD file as *file_name-time*.BAK

- --character-sets-dir=*path*

  The directory where character sets are installed. See Section 10.5, “Character Set Configuration”.

- --correct-checksum

  Correct the checksum information for the table.

- `--data-file-length=len`, `-D len`

  The maximum length of the data file (when re-creating data file when it is “full”).

- `--extend-check`, `-e`

  Do a repair that tries to recover every possible row from the data file. Normally, this also finds a lot of garbage rows. Do not use this option unless you are desperate.

  See also the description of this option under table checking options.

  For a description of the output format, see Section 4.6.3.5, “Obtaining Table Information with `myisamchk`”.

- `--force`, `-f`

  Overwrite old intermediate files (files with names like `tbl_name.TMD`) instead of aborting.

- `--keys-used=val`, `-k val`

  For `myisamchk`, the option value is a bit-value that indicates which indexes to update. Each binary bit of the option value corresponds to a table index, where the first index is bit 0. An option value of 0 disables updates to all indexes, which can be used to get faster inserts. Deactivated indexes can be reactivated by using `myisamchk -r`.

- `--no-symlinks`, `-l`

  Do not follow symbolic links. Normally `myisamchk` repairs the table that a symlink points to. This option does not exist as of MySQL 4.0 because versions from 4.0 on do not remove symlinks during repair operations.

- `--max-record-length=len`

  Skip rows larger than the given length if `myisamchk` cannot allocate memory to hold them.

- `--parallel-recover`, `-p`

  Use the same technique as `-r` and `-n`, but create all the keys in parallel, using different threads. *This is beta-quality code. Use at your own risk!*

- `--quick`, `-q`

  Achieve a faster repair by modifying only the index file, not the data file. You can specify this option twice to force `myisamchk` to modify the original data file in case of duplicate keys.

- `--recover`, `-r`

  Do a repair that can fix almost any problem except unique keys that are not unique (which is an extremely unlikely error with `MyISAM` tables). If you want to recover a table, this is the option to try first. You should try `--safe-recover` only if `myisamchk` reports that the table cannot be recovered using `--recover`. (In the unlikely case that `--recover` fails, the data file remains intact.)

  If you have lots of memory, you should increase the value of `myisam_sort_buffer_size`.

- `--safe-recover`, `-o`

  Do a repair using an old recovery method that reads through all rows in order and updates all index trees based on the rows found. This is an order of magnitude slower than `--recover`, but can handle a couple of very unlikely cases that --recover cannot. This recovery method also uses much less disk space than --recover. Normally, you should repair first using --recover, and then with --safe-recover only if --recover fails.

If you have lots of memory, you should increase the value of key_buffer_size.

- --set-character-set=*name*

  Change the character set used by the table indexes. This option was replaced by --set-collation in MySQL 5.0.3.

- --set-collation=*name*

  Specify the collation to use for sorting table indexes. The character set name is implied by the first part of the collation name.

- --sort-recover, -n

  Force myisamchk to use sorting to resolve the keys even if the temporary files would be very large.

- --tmpdir=*path*, -t *path*

  The path of the directory to be used for storing temporary files. If this is not set, myisamchk uses the value of the TMPDIR environment variable. --tmpdir can be set to a list of directory paths that are used successively in round-robin fashion for creating temporary files. The separator character between directory names is the colon (“:”) on Unix and the semicolon (“;”) on Windows.

- --unpack, -u

  Unpack a table that was packed with myisampack.

4.6.3.4 Other myisamchk Options

myisamchk supports the following options for actions other than table checks and repairs:

- --analyze, -a

  Analyze the distribution of key values. This improves join performance by enabling the join optimizer to better choose the order in which to join the tables and which indexes it should use. To obtain information about the key distribution, use a myisamchk --description --verbose *tbl_name* command or the SHOW INDEX FROM *tbl_name* statement.

- --block-search=*offset*, -b *offset*

  Find the record that a block at the given offset belongs to.

- --description, -d

  Print some descriptive information about the table. Specifying the --verbose option once or twice produces additional information. See Section 4.6.3.5, “Obtaining Table Information with myisamchk”.

- --set-auto-increment[=*value*], -A[*value*]

  Force AUTO_INCREMENT numbering for new records to start at the given value (or higher, if there are existing records with AUTO_INCREMENT values this large). If *value* is not specified, AUTO_INCREMENT numbers for new records begin with the largest value currently in the table, plus one.

- --sort-index, -S

Sort the index tree blocks in high-low order. This optimizes seeks and makes table scans that use indexes faster.

- `--sort-records=N`, `-R N`

  Sort records according to a particular index. This makes your data much more localized and may speed up range-based `SELECT` and `ORDER BY` operations that use this index. (The first time you use this option to sort a table, it may be very slow.) To determine a table's index numbers, use `SHOW INDEX`, which displays a table's indexes in the same order that `myisamchk` sees them. Indexes are numbered beginning with 1.

  If keys are not packed (`PACK_KEYS=0`), they have the same length, so when `myisamchk` sorts and moves records, it just overwrites record offsets in the index. If keys are packed (`PACK_KEYS=1`), `myisamchk` must unpack key blocks first, then re-create indexes and pack the key blocks again. (In this case, re-creating indexes is faster than updating offsets for each index.)

4.6.3.5 Obtaining Table Information with myisamchk

To obtain a description of a `MyISAM` table or statistics about it, use the commands shown here. The output from these commands is explained later in this section.

- `myisamchk -d tbl_name`

  Runs `myisamchk` in “describe mode” to produce a description of your table. If you start the MySQL server with external locking disabled, `myisamchk` may report an error for a table that is updated while it runs. However, because `myisamchk` does not change the table in describe mode, there is no risk of destroying data.

- `myisamchk -dv tbl_name`

  Adding `-v` runs `myisamchk` in verbose mode so that it produces more information about the table. Adding `-v` a second time produces even more information.

- `myisamchk -eis tbl_name`

  Shows only the most important information from a table. This operation is slow because it must read the entire table.

- `myisamchk -eiv tbl_name`

  This is like `-eis`, but tells you what is being done.

The *`tbl_name`* argument can be either the name of a `MyISAM` table or the name of its index file, as described in Section 4.6.3, “`myisamchk` — MyISAM Table-Maintenance Utility”. Multiple *`tbl_name`* arguments can be given.

Suppose that a table named `person` has the following structure. (The `MAX_ROWS` table option is included so that in the example output from `myisamchk` shown later, some values are smaller and fit the output format more easily.)

```
CREATE TABLE person
(
  id         INT NOT NULL AUTO_INCREMENT,
  last_name  VARCHAR(20) NOT NULL,
  first_name VARCHAR(20) NOT NULL,
  birth      DATE,
  death      DATE,
  PRIMARY KEY (id),
  INDEX (last_name, first_name),
  INDEX (birth)
) MAX_ROWS = 1000000;
```

Suppose also that the table has these data and index file sizes:

```
-rw-rw----  1 mysql  mysql  9347072 Aug 19 11:47 person.MYD
-rw-rw----  1 mysql  mysql  6066176 Aug 19 11:47 person.MYI
```

Example of `myisamchk -dvv` output:

```
MyISAM file:         person
Record format:       Packed
Character set:       latin1_swedish_ci (8)
File-version:        1
Creation time:       2009-08-19 16:47:41
Recover time:        2009-08-19 16:47:56
Status:              checked,analyzed,optimized keys
Auto increment key:              1  Last value:                306688
Data records:               306688  Deleted blocks:                 0
Datafile parts:             306688  Deleted data:                   0
Datafile pointer (bytes):        4  Keyfile pointer (bytes):        3
Datafile length:           9347072  Keyfile length:           6066176
Max datafile length:    4294967294  Max keyfile length:   17179868159
Recordlength:                   54

table description:
Key Start Len Index   Type                 Rec/key         Root  Blocksize
1   2     4   unique  long                       1        99328       1024
2   6     20  multip. varchar prefix           512      3563520       1024
    27    20          varchar                  512
3   48    3   multip. uint24 NULL           306688      6065152       1024

Field Start Length Nullpos Nullbit Type
1     1     1
2     2     4                      no zeros
3     6     21                     varchar
4     27    21                     varchar
5     48    3      1       1       no zeros
6     51    3      1       2       no zeros
```

Explanations for the types of information `myisamchk` produces are given here. “Keyfile” refers to the index file. “Record” and “row” are synonymous, as are “field” and “column.”

The initial part of the table description contains these values:

- `MyISAM file`

  Name of the `MyISAM` (index) file.

- `Record format`

  The format used to store table rows. The preceding examples use `Fixed length`. Other possible values are `Compressed` and `Packed`. (`Packed` corresponds to what `SHOW TABLE STATUS` reports as `Dynamic`.)

- `Chararacter set`

  The table default character set.

- `File-version`

  Version of `MyISAM` format. Currently always 1.

- Creation time

  When the data file was created.

- Recover time

  When the index/data file was last reconstructed.

- Status

  Table status flags. Possible values are crashed, open, changed, analyzed, optimized keys, and sorted index pages.

- Auto increment key, Last value

  The key number associated the table's AUTO_INCREMENT column, and the most recently generated value for this column. These fields do not appear if there is no such column.

- Data records

  The number of rows in the table.

- Deleted blocks

  How many deleted blocks still have reserved space. You can optimize your table to minimize this space. See Section 7.6.4, “MyISAM Table Optimization”.

- Datafile parts

  For dynamic-row format, this indicates how many data blocks there are. For an optimized table without fragmented rows, this is the same as Data records.

- Deleted data

  How many bytes of unreclaimed deleted data there are. You can optimize your table to minimize this space. See Section 7.6.4, “MyISAM Table Optimization”.

- Datafile pointer

  The size of the data file pointer, in bytes. It is usually 2, 3, 4, or 5 bytes. Most tables manage with 2 bytes, but this cannot be controlled from MySQL yet. For fixed tables, this is a row address. For dynamic tables, this is a byte address.

- Keyfile pointer

  The size of the index file pointer, in bytes. It is usually 1, 2, or 3 bytes. Most tables manage with 2 bytes, but this is calculated automatically by MySQL. It is always a block address.

- Max datafile length

  How long the table data file can become, in bytes.

- Max keyfile length

  How long the table index file can become, in bytes.

- Recordlength

  How much space each row takes, in bytes.

The table description part of the output includes a list of all keys in the table. For each key, myisamchk displays some low-level information:

- Key

  This key's number. This value is shown only for the first column of the key. If this value is missing, the line corresponds to the second or later column of a multiple-column key. For the table shown in the example, there are two table description lines for the second index. This indicates that it is a multiple-part index with two parts.

- Start

  Where in the row this portion of the index starts.

- Len

  How long this portion of the index is. For packed numbers, this should always be the full length of the column. For strings, it may be shorter than the full length of the indexed column, because you can index a prefix of a string column. The total length of a multiple-part key is the sum of the Len values for all key parts.

- Index

  Whether a key value can exist multiple times in the index. Possible values are unique or multip. (multiple).

- Type

  What data type this portion of the index has. This is a MyISAM data type with the possible values packed, stripped, or empty.

- Root

  Address of the root index block.

- Blocksize

  The size of each index block. By default this is 1024, but the value may be changed at compile time when MySQL is built from source.

- Rec/key

  This is a statistical value used by the optimizer. It tells how many rows there are per value for this index. A unique index always has a value of 1. This may be updated after a table is loaded (or greatly changed) with myisamchk -a. If this is not updated at all, a default value of 30 is given.

The last part of the output provides information about each column:

- Field

  The column number.

- Start

  The byte position of the column within table rows.

- Length

  The length of the column in bytes.

- Nullpos, Nullbit

  For columns that can be NULL, MyISAM stores NULL values as a flag in a byte. Depending on how many nullable columns there are, there can be one or more bytes used for this purpose. The Nullpos and Nullbit values, if nonempty, indicate which byte and bit contains that flag indicating whether the column is NULL.

  The position and number of bytes used to store NULL flags is shown in the line for field 1. This is why there are six Field lines for the person table even though it has only five columns.

- Type

  The data type. The value may contain any of the following descriptors:

  - constant

    All rows have the same value.

  - no endspace

    Do not store endspace.

  - no endspace, not_always

    Do not store endspace and do not do endspace compression for all values.

  - no endspace, no empty

    Do not store endspace. Do not store empty values.

  - table-lookup

    The column was converted to an ENUM.

  - zerofill(*N*)

    The most significant *N* bytes in the value are always 0 and are not stored.

  - no zeros

    Do not store zeros.

  - always zero

    Zero values are stored using one bit.

- Huff tree

  The number of the Huffman tree associated with the column.

- Bits

  The number of bits used in the Huffman tree.

The Huff tree and Bits fields are displayed if the table has been compressed with myisampack. See Section 4.6.5, "myisampack — Generate Compressed, Read-Only MyISAM Tables", for an example of this information.

Example of myisamchk -eiv output:

```
Checking MyISAM file: person
Data records:  306688   Deleted blocks:       0
- check file-size
- check record delete-chain
No recordlinks
- check key delete-chain
block_size 1024:
- check index reference
- check data record references index: 1
Key:  1:  Keyblocks used:  98%  Packed:    0%  Max levels:  3
- check data record references index: 2
Key:  2:  Keyblocks used:  99%  Packed:   97%  Max levels:  3
- check data record references index: 3
Key:  3:  Keyblocks used:  98%  Packed:  -14%  Max levels:  3
Total:    Keyblocks used:  98%  Packed:   89%

- check records and index references
*** LOTS OF ROW NUMBERS DELETED ***

Records:            306688  M.recordlength:       25  Packed:            83%
Recordspace used:       97% Empty space:           2% Blocks/Record:   1.00
Record blocks:      306688  Delete blocks:         0
Record data:       7934464  Deleted data:          0
Lost space:         256512  Linkdata:        1156096

User time 43.08, System time 1.68
Maximum resident set size 0, Integral resident set size 0
Non-physical pagefaults 0, Physical pagefaults 0, Swaps 0
Blocks in 0 out 7, Messages in 0 out 0, Signals 0
Voluntary context switches 0, Involuntary context switches 0
Maximum memory usage: 1046926 bytes (1023k)
```

myisamchk -eiv output includes the following information:

- Data records

  The number of rows in the table.

- Deleted blocks

  How many deleted blocks still have reserved space. You can optimize your table to minimize this space. See Section 7.6.4, “MyISAM Table Optimization”.

- Key

  The key number.

- Keyblocks used

  What percentage of the keyblocks are used. When a table has just been reorganized with myisamchk, the values are very high (very near theoretical maximum).

- Packed

  MySQL tries to pack key values that have a common suffix. This can only be used for indexes on CHAR and VARCHAR columns. For long indexed strings that have similar leftmost parts, this can significantly reduce the space used. In the preceding example, the second key is 40 bytes long and a 97% reduction in space is achieved.

- Max levels

  How deep the B-tree for this key is. Large tables with long key values get high values.

- `Records`

  How many rows are in the table.

- `M.recordlength`

  The average row length. This is the exact row length for tables with fixed-length rows, because all rows have the same length.

- `Packed`

  MySQL strips spaces from the end of strings. The `Packed` value indicates the percentage of savings achieved by doing this.

- `Recordspace used`

  What percentage of the data file is used.

- `Empty space`

  What percentage of the data file is unused.

- `Blocks/Record`

  Average number of blocks per row (that is, how many links a fragmented row is composed of). This is always 1.0 for fixed-format tables. This value should stay as close to 1.0 as possible. If it gets too large, you can reorganize the table. See Section 7.6.4, “`MyISAM` Table Optimization”.

- `Recordblocks`

  How many blocks (links) are used. For fixed-format tables, this is the same as the number of rows.

- `Deleteblocks`

  How many blocks (links) are deleted.

- `Recorddata`

  How many bytes in the data file are used.

- `Deleted data`

  How many bytes in the data file are deleted (unused).

- `Lost space`

  If a row is updated to a shorter length, some space is lost. This is the sum of all such losses, in bytes.

- `Linkdata`

  When the dynamic table format is used, row fragments are linked with pointers (4 to 7 bytes each). `Linkdata` is the sum of the amount of storage used by all such pointers.

4.6.3.6 `myisamchk` Memory Usage

Memory allocation is important when you run `myisamchk`. `myisamchk` uses no more memory than its memory-related variables are set to. If you are going to use `myisamchk` on very large tables, you should first decide how much memory you want it to use. The default is to use only about 3MB to perform repairs. By using larger values, you can get `myisamchk` to operate faster. For example, if you have more than

512MB RAM available, you could use options such as these (in addition to any other options you might specify):

```
shell> myisamchk --myisam_sort_buffer_size=256M \
           --key_buffer_size=512M \
           --read_buffer_size=64M \
           --write_buffer_size=64M ...
```

Using --myisam_sort_buffer_size=16M is probably enough for most cases.

Be aware that myisamchk uses temporary files in TMPDIR. If TMPDIR points to a memory file system, out of memory errors can easily occur. If this happens, run myisamchk with the --tmpdir=*path* option to specify a directory located on a file system that has more space.

When performing repair operations, myisamchk also needs a lot of disk space:

- Twice the size of the data file (the original file and a copy). This space is not needed if you do a repair with --quick; in this case, only the index file is re-created. *This space must be available on the same file system as the original data file*, as the copy is created in the same directory as the original.

- Space for the new index file that replaces the old one. The old index file is truncated at the start of the repair operation, so you usually ignore this space. This space must be available on the same file system as the original data file.

- When using --recover or --sort-recover (but not when using --safe-recover), you need space on disk for sorting. This space is allocated in the temporary directory (specified by TMPDIR or --tmpdir=*path*). The following formula yields the amount of space required:

  ```
  (largest_key + row_pointer_length) * number_of_rows * 2
  ```

  You can check the length of the keys and the *row_pointer_length* with myisamchk -dv *tbl_name* (see Section 4.6.3.5, “Obtaining Table Information with myisamchk”). The *row_pointer_length* and *number_of_rows* values are the Datafile pointer and Data records values in the table description. To determine the *largest_key* value, check the Key lines in the table description. The Len column indicates the number of bytes for each key part. For a multiple-column index, the key size is the sum of the Len values for all key parts.

If you have a problem with disk space during repair, you can try --safe-recover instead of --recover.

4.6.4 myisamlog — Display MyISAM Log File Contents

myisamlog processes the contents of a MyISAM log file. To create such a file, start the server with a --log-isam=log_file option.

Invoke myisamlog like this:

```
shell> myisamlog [options] [file_name [tbl_name] ...]
```

The default operation is update (-u). If a recovery is done (-r), all writes and possibly updates and deletes are done and errors are only counted. The default log file name is myisam.log if no *log_file* argument is given. If tables are named on the command line, only those tables are updated.

myisamlog supports the following options:

- -?, -I

  Display a help message and exit.

- `-c N`

  Execute only *N* commands.

- `-f N`

  Specify the maximum number of open files.

- `-i`

  Display extra information before exiting.

- `-o offset`

  Specify the starting offset.

- `-p N`

  Remove *N* components from path.

- `-r`

  Perform a recovery operation.

- `-R record_pos_file record_pos`

  Specify record position file and record position.

- `-u`

  Perform an update operation.

- `-v`

  Verbose mode. Print more output about what the program does. This option can be given multiple times to produce more and more output.

- `-w write_file`

  Specify the write file.

- `-V`

  Display version information.

4.6.5 `myisampack` — Generate Compressed, Read-Only MyISAM Tables

The `myisampack` utility compresses `MyISAM` tables. `myisampack` works by compressing each column in the table separately. Usually, `myisampack` packs the data file 40% to 70%.

When the table is used later, the server reads into memory the information needed to decompress columns. This results in much better performance when accessing individual rows, because you only have to uncompress exactly one row.

MySQL uses `mmap()` when possible to perform memory mapping on compressed tables. If `mmap()` does not work, MySQL falls back to normal read/write file operations.

Please note the following:

- If the mysqld server was invoked with external locking disabled, it is not a good idea to invoke myisampack if the table might be updated by the server during the packing process. It is safest to compress tables with the server stopped.

- After packing a table, it becomes read only. This is generally intended (such as when accessing packed tables on a CD).

- myisampack does not support partitioned tables.

Invoke myisampack like this:

```
shell> myisampack [options] file_name ...
```

Each file name argument should be the name of an index (.MYI) file. If you are not in the database directory, you should specify the path name to the file. It is permissible to omit the .MYI extension.

After you compress a table with myisampack, you should use myisamchk -rq to rebuild its indexes. Section 4.6.3, "myisamchk — MyISAM Table-Maintenance Utility".

myisampack supports the following options. It also reads option files and supports the options for processing them described at Section 4.2.3.4, "Command-Line Options that Affect Option-File Handling".

- --help, -?

  Display a help message and exit.

- --backup, -b

  Make a backup of each table's data file using the name *tbl_name*.OLD.

- --character-sets-dir=*path*

  The directory where character sets are installed. See Section 10.5, "Character Set Configuration".

- --debug[=*debug_options*], -# [*debug_options*]

  Write a debugging log. A typical *debug_options* string is d:t:o,*file_name*. The default is d:t:o.

- --force, -f

  Produce a packed table even if it becomes larger than the original or if the intermediate file from an earlier invocation of myisampack exists. (myisampack creates an intermediate file named *tbl_name*.TMD in the database directory while it compresses the table. If you kill myisampack, the .TMD file might not be deleted.) Normally, myisampack exits with an error if it finds that *tbl_name*.TMD exists. With --force, myisampack packs the table anyway.

- --join=*big_tbl_name*, -j *big_tbl_name*

  Join all tables named on the command line into a single packed table *big_tbl_name*. All tables that are to be combined *must* have identical structure (same column names and types, same indexes, and so forth).

  big_tbl_name must not exist prior to the join operation. All source tables named on the command line to be merged into *big_tbl_name* must exist. The source tables are read for the join operation but not modified. The join operation does not create a .frm file for *big_tbl_name*, so after the join operation finishes, copy the .frm file from one of the source tables and name it *big_tbl_name*.frm.

- --silent, -s

Silent mode. Write output only when errors occur.

- `--test`, `-t`

  Do not actually pack the table, just test packing it.

- `--tmpdir=path`, `-T path`

  Use the named directory as the location where `myisampack` creates temporary files.

- `--verbose`, `-v`

  Verbose mode. Write information about the progress of the packing operation and its result.

- `--version`, `-V`

  Display version information and exit.

- `--wait`, `-w`

  Wait and retry if the table is in use. If the `mysqld` server was invoked with external locking disabled, it is not a good idea to invoke `myisampack` if the table might be updated by the server during the packing process.

The following sequence of commands illustrates a typical table compression session:

```
shell> ls -l station.*
-rw-rw-r--   1 monty    my         994128 Apr 17 19:00 station.MYD
-rw-rw-r--   1 monty    my          53248 Apr 17 19:00 station.MYI
-rw-rw-r--   1 monty    my           5767 Apr 17 19:00 station.frm

shell> myisamchk -dvv station

MyISAM file:     station
Isam-version:  2
Creation time: 1996-03-13 10:08:58
Recover time:  1997-02-02  3:06:43
Data records:              1192  Deleted blocks:              0
Datafile parts:            1192  Deleted data:                0
Datafile pointer (bytes):     2  Keyfile pointer (bytes):     2
Max datafile length:   54657023  Max keyfile length:   33554431
Recordlength:               834
Record format: Fixed length

table description:
Key Start Len Index   Type                 Root  Blocksize    Rec/key
1   2     4   unique  unsigned long        1024       1024          1
2   32    30  multip. text                10240       1024          1

Field Start Length Type
1     1     1
2     2     4
3     6     4
4     10    1
5     11    20
6     31    1
7     32    30
8     62    35
9     97    35
10    132   35
11    167   4
12    171   16
```

```
13    187   35
14    222   4
15    226   16
16    242   20
17    262   20
18    282   20
19    302   30
20    332   4
21    336   4
22    340   1
23    341   8
24    349   8
25    357   8
26    365   2
27    367   2
28    369   4
29    373   4
30    377   1
31    378   2
32    380   8
33    388   4
34    392   4
35    396   4
36    400   4
37    404   1
38    405   4
39    409   4
40    413   4
41    417   4
42    421   4
43    425   4
44    429   20
45    449   30
46    479   1
47    480   1
48    481   79
49    560   79
50    639   79
51    718   79
52    797   8
53    805   1
54    806   1
55    807   20
56    827   4
57    831   4

shell> myisampack station.MYI
Compressing station.MYI: (1192 records)
- Calculating statistics

normal:     20  empty-space:   16  empty-zero:     12  empty-fill:  11
pre-space:   0  end-space:     12  table-lookups:   5  zero:         7
Original trees:  57  After join: 17
- Compressing file
87.14%
Remember to run myisamchk -rq on compressed tables

shell> ls -l station.*
-rw-rw-r--   1 monty    my         127874 Apr 17 19:00 station.MYD
-rw-rw-r--   1 monty    my          55296 Apr 17 19:04 station.MYI
-rw-rw-r--   1 monty    my           5767 Apr 17 19:00 station.frm

shell> myisamchk -dvv station

MyISAM file:     station
Isam-version:  2
Creation time: 1996-03-13 10:08:58
```

```
Recover time:  1997-04-17 19:04:26
Data records:               1192  Deleted blocks:              0
Datafile parts:             1192  Deleted data:                0
Datafile pointer (bytes):      3  Keyfile pointer (bytes):     1
Max datafile length:    16777215  Max keyfile length:     131071
Recordlength:                834
Record format: Compressed

table description:
Key Start Len Index   Type                 Root  Blocksize    Rec/key
1   2     4   unique  unsigned long       10240       1024          1
2   32    30  multip. text                54272       1024          1

Field Start Length Type                         Huff tree  Bits
1     1     1      constant                             1     0
2     2     4      zerofill(1)                          2     9
3     6     4      no zeros, zerofill(1)                2     9
4     10    1                                           3     9
5     11    20     table-lookup                         4     0
6     31    1                                           3     9
7     32    30     no endspace, not_always              5     9
8     62    35     no endspace, not_always, no empty    6     9
9     97    35     no empty                             7     9
10    132   35     no endspace, not_always, no empty    6     9
11    167   4      zerofill(1)                          2     9
12    171   16     no endspace, not_always, no empty    5     9
13    187   35     no endspace, not_always, no empty    6     9
14    222   4      zerofill(1)                          2     9
15    226   16     no endspace, not_always, no empty    5     9
16    242   20     no endspace, not_always              8     9
17    262   20     no endspace, no empty                8     9
18    282   20     no endspace, no empty                5     9
19    302   30     no endspace, no empty                6     9
20    332   4      always zero                          2     9
21    336   4      always zero                          2     9
22    340   1                                           3     9
23    341   8      table-lookup                         9     0
24    349   8      table-lookup                        10     0
25    357   8      always zero                          2     9
26    365   2                                           2     9
27    367   2      no zeros, zerofill(1)                2     9
28    369   4      no zeros, zerofill(1)                2     9
29    373   4      table-lookup                        11     0
30    377   1                                           3     9
31    378   2      no zeros, zerofill(1)                2     9
32    380   8      no zeros                             2     9
33    388   4      always zero                          2     9
34    392   4      table-lookup                        12     0
35    396   4      no zeros, zerofill(1)               13     9
36    400   4      no zeros, zerofill(1)                2     9
37    404   1                                           2     9
38    405   4      no zeros                             2     9
39    409   4      always zero                          2     9
40    413   4      no zeros                             2     9
41    417   4      always zero                          2     9
42    421   4      no zeros                             2     9
43    425   4      always zero                          2     9
44    429   20     no empty                             3     9
45    449   30     no empty                             3     9
46    479   1                                          14     4
47    480   1                                          14     4
48    481   79     no endspace, no empty               15     9
49    560   79     no empty                             2     9
50    639   79     no empty                             2     9
51    718   79     no endspace                         16     9
52    797   8      no empty                             2     9
53    805   1                                          17     1
```

```
54    806   1                                         3     9
55    807   20     no empty                           3     9
56    827   4      no zeros, zerofill(2)              2     9
57    831   4      no zeros, zerofill(1)              2     9
```

myisampack displays the following kinds of information:

- normal

  The number of columns for which no extra packing is used.

- empty-space

  The number of columns containing values that are only spaces. These occupy one bit.

- empty-zero

  The number of columns containing values that are only binary zeros. These occupy one bit.

- empty-fill

  The number of integer columns that do not occupy the full byte range of their type. These are changed to a smaller type. For example, a BIGINT column (eight bytes) can be stored as a TINYINT column (one byte) if all its values are in the range from -128 to 127.

- pre-space

  The number of decimal columns that are stored with leading spaces. In this case, each value contains a count for the number of leading spaces.

- end-space

  The number of columns that have a lot of trailing spaces. In this case, each value contains a count for the number of trailing spaces.

- table-lookup

  The column had only a small number of different values, which were converted to an ENUM before Huffman compression.

- zero

  The number of columns for which all values are zero.

- Original trees

  The initial number of Huffman trees.

- After join

  The number of distinct Huffman trees left after joining trees to save some header space.

After a table has been compressed, the Field lines displayed by myisamchk -dvv include additional information about each column:

- Type

  The data type. The value may contain any of the following descriptors:

  - constant

    All rows have the same value.

  - no endspace

    Do not store endspace.

  - no endspace, not_always

    Do not store endspace and do not do endspace compression for all values.

  - no endspace, no empty

    Do not store endspace. Do not store empty values.

  - table-lookup

    The column was converted to an ENUM.

  - zerofill(*N*)

    The most significant *N* bytes in the value are always 0 and are not stored.

  - no zeros

    Do not store zeros.

  - always zero

    Zero values are stored using one bit.

- Huff tree

  The number of the Huffman tree associated with the column.

- Bits

  The number of bits used in the Huffman tree.

After you run myisampack, you must run myisamchk to re-create any indexes. At this time, you can also sort the index blocks and create statistics needed for the MySQL optimizer to work more efficiently:

```
shell> myisamchk -rq --sort-index --analyze tbl_name.MYI
```

After you have installed the packed table into the MySQL database directory, you should execute mysqladmin flush-tables to force mysqld to start using the new table.

To unpack a packed table, use the --unpack option to myisamchk.

4.6.6 mysql_config_editor — MySQL Configuration Utility

The mysql_config_editor utility enables you to store authentication credentials in an encrypted login file named .mylogin.cnf. The file location is the %APPDATA%\MySQL directory on Windows and the current user's home directory on non-Windows systems. The file can be read later by MySQL client programs to obtain authentication credentials for connecting to MySQL Server.

To specify an alternate file name, set the MYSQL_TEST_LOGIN_FILE environment variable. This variable is used by the mysql-test-run.pl testing utility, but also is recognized by mysql_config_editor and by MySQL clients such as mysql, mysqladmin, and so forth.

mysql_config_editor encrypts the .mylogin.cnf file so it cannot be read as clear text, and its contents when decrypted by client programs are used only in memory. In this way, passwords can be stored in a file in non-cleartext format and used later without ever needing to be exposed on the command line or in an environment variable. mysql_config_editor provides a print command that enables the user to display the file contents, but even in this case, password values are masked so as never to appear in a way that other users can see them.

The encryption used by mysql_config_editor prevents passwords from appearing in .mylogin.cnf as clear text and provides a measure of security by preventing inadvertent password exposure. For example, if you display a regular unencrypted my.cnf option file on the screen, any passwords it contains are visible for anyone to see. With .mylogin.cnf, that is not true. But the encryption used will not deter a determined attacker and you should not consider it unbreakable. A user who can gain system administration privileges on your machine to access your files could decrypt the .mylogin.cnf file with some effort.

The login file must be readable and writable to the current user, and inaccessible to other users. Otherwise, mysql_config_editor ignores it, and the file is not used by client programs, either. On Windows, this constraint does not apply; instead, the user must have access to the %APPDATA%\MySQL directory.

The unencrypted format of the .mylogin.cnf login file consists of option groups, similar to other option files. Each option group in .mylogin.cnf is called a “login path,” which is a group that permits only a limited set of options: host, user, and password. Think of a login path as a set of values that indicate the server host and the credentials for authenticating with the server. Here is an example:

```
[myloginpath]
user = myname
password = mypass
host = 127.0.0.1
```

When you invoke a client program to connect to the server, .mylogin.cnf is used in conjunction with other option files. Its precedence is higher than other option files, but less than options specified explicitly on the client command line. For information about the order in which option files are used, see Section 4.2.3.3, “Using Option Files”.

Invoke mysql_config_editor like this:

```
shell> mysql_config_editor [program_options] command [command_options]
```

program_options consists of general mysql_config_editor options. command indicates what command to perform, and *command_options* indicates any additional options needed by the command.

The command indicates what action to perform on the .mylogin.cnf login file. For example, set writes a login path to the file, remove removes a login path, and print displays login path contents. Any options given provide information to the command, such as the login path name and the values to use in the login path.

The position of the command name within the set of program arguments is significant. For example, these command lines have the same arguments, but produce different results:

```
mysql_config_editor --help set
mysql_config_editor set --help
```

The first command line displays general mysql_config_editor help, and ignores the set command. The second command line displays help for the set command.

Suppose that you want to establish two login paths named local and remote for connecting to the local MySQL server and a server on the host remote.example.com. You want to authenticate to the local server with a user name and password of localuser and localpass, and to the remote server with a user name and password of remoteuser and remotepass. To set up the login paths in the .mylogin.cnf file, use the following set commands. Enter each command on a single line, then enter the appropriate password when prompted.

```
shell> mysql_config_editor set --login-path=local
         --host=localhost --user=localuser --password
Enter password: enter password "localpass" here
shell> mysql_config_editor set --login-path=remote
         --host=remote.example.com --user=remoteuser --password
Enter password: enter password "remotepass" here
```

To see what mysql_config_editor wrote to the .mylogin.cnf file, use the print command:

```
shell> mysql_config_editor print --all
[local]
user = localuser
password = *****
host = localhost
[remote]
user = remoteuser
password = *****
host = remote.example.com
```

The print command displays each login path as a set of lines beginning with a group header indicating the login path name in square brackets, followed by the option values for the login path. Password values are masked and do not appear as clear text.

As shown by the preceding examples, the .mylogin.cnf file can contain multiple login paths. In this way, mysql_config_editor makes it easy to set up multiple “personalities” for connecting to different MySQL servers. Any of these can be selected by name later using the --login-path option when you invoke a client program. For example, to connect to the local server, use this command:

```
shell> mysql --login-path=local
```

To connect to the remote server, use this command:

```
shell> mysql --login-path=remote
```

When you use the set command with mysql_config_editor to create a login path, you need not specify all three possible option values (host name, user name, and password). Only those values given are written to the path. Any missing values required later can be specified when you invoke a client path to connect to the MySQL server, either in other option files or on the command line. Also, any options specified on the command line override those in option files, including the .mylogin.cnf file. For example, if the credentials in the remote login path also apply for the host remote2.example.com, you can connect to the server on that host like this:

```
shell> mysql --login-path=remote --host=remote2.example.com
```

The .mylogin.cnf file, if it exists, is read in all cases, even when the --no-defaults option is used. This permits passwords to be specified in a safer way than on the command line even if --no-defaults is present.

mysql_config_editor Commands

This section describes the permitted mysql_config_editor commands, and the interpretation of options that have a command-specific meaning. In addition, mysql_config_editor takes other options that can be used with any command, such as --verbose to produce more information as mysql_config_editor executes. This option may be helpful in diagnosing problems if an operation does not have the effect you expect. For a list of supported options, see mysql_config_editor Options.

mysql_config_editor supports these commands:

- help

  Display a help message and exit.

- print [*options*]

  Print the contents of .mylogin.cnf in unencrypted form. Passwords are displayed as *****.

  The print command takes these options:

  - --all

    Print all login paths.

  - --login-path=*name*

    Print the named login path.

  If no login path is specified, the default path name is client. If both --all and --login-path are given, --all takes precedence.

- remove [*options*]

  Remove a login path from the .mylogin.cnf file.

  The remove command takes these options:

  - --host

    Remove the host name from the login path.

  - --login-path=*name*

    The login path to remove. If this option is not given, the default path name is client.

  - --password

    Remove the password from the login path.

  - --port

    Remove the TCP/IP port number from the login path.

  - --socket

  Remove the Unix socket file name from the login path.

  - --user

    Remove the user name from the login path.

  The --port and --socket options are supported for the remove command as of MySQL 5.7.1

  The remove command removes from the login path only such values as are specified with the --host, --password, --port, --socket, and --user options. If none of them is given, remove removes the entire login path. For example, this command removes only the user value from the client login path rather than the entire client login path:

  ```
  mysql_config_editor remove --login-path=client --user
  ```

- reset

  Empty the contents of the .mylogin.cnf file. The file is created if it does not exist.

- set [*options*]

  Write a login path to the .mylogin.cnf file.

  The set command takes these options:

  - --host=*host_name*

    The host name to write to the login path.

  - --login-path=*name*

    The login path to create. If this option is not given, the default path name is client.

  - --password

    Prompt for a password to write to the login path.

  - --port=*port_num*

    The TCP/IP port number to write to the login path.

  - --socket=*file_name*

    The Unix socket file to write to the login path.

  - --user=*user_name*

    The user name to write to the login path.

  The --port and --socket options are supported for the set command as of MySQL 5.7.1

  The set command writes to the login path only such values as are specified with the --host, --password, --port, --socket, and --user options. If none of those options are given, mysql_config_editor writes the login path as an empty group.

  To specify an empty password, use the set command with the --password option, then press Enter at the password prompt. The resulting login path written to .mylogin.cnf will include a line like this:

```
password =
```

If the login path already exists in `.mylogin.cnf`, the `set` command replaces it. To ensure that this is what the user wants, `mysql_config_editor` prints a warning and prompts for confirmation. To suppress the warning and prompt, use the `--skip-warn` option.

`mysql_config_editor` Options

`mysql_config_editor` supports the following options.

Table 4.14 `mysql_config_editor` Options

Format	Option File	Description	Introduced
--all		Print all login paths	
--debug[=debug_options]		Write a debugging log	
--help		Display help message and exit	
--host=host_name		Host to write to login file	
--login-path=name		Login path name	
--password		Solicit password to write to login file	
--port=port_num	port	The TCP/IP port number to write to login file	5.7.1
--socket=path	socket	The Unix socket file name to write to login file	5.7.1
--user=user_name		User name to write to login file	
--verbose		Verbose mode	
--version		Display version information and exit	
--warn		Warn and solicit confirmation for overwriting login path	

- `--help`, `-?`

  Display a help message and exit. If preceded by a command name such as `set` or `remove`, displays information about that command.

- `--all`

  For the `print` command, print all login paths in the login file.

- `--debug[=`*`debug_options`*`]`, `-#` *`debug_options`*

  Write a debugging log. A typical *`debug_options`* string is `d:t:o,`*`file_name`*. The default is `d:t:o`.

- `--host=`*`host_name`*, `-h` *`host_name`*

  For the `set` command, the host name to write to to the login path. For the `remove` command, removes the host name from the login path.

- `--login-path=`*`name`*, `-G` *`name`*

  For the `print`, `remove`, and `set` commands, the login path to use in the `.mylogin.cnf` login file.

  Client programs also support the `--login-path` option, to enable users to specify which login path to use for connecting to a MySQL server. For client programs, `--login-path` must be the first option given, which is not true for mysql_config_editor. See Section 4.2.3.4, "Command-Line Options that Affect Option-File Handling".

- --password, -p

  For the set command, cause mysql_config_editor to prompt for a password and write the value entered by the user to the login path. After mysql_config_editor starts and displays the prompt, the user should type the password and press Enter. To prevent other users from seeing the password, mysql_config_editor does not echo it.

  This option does not permit a password value following the option name. That is, with mysql_config_editor, you never enter a password on the command line where it might be seen by other users. This differs from most other MySQL programs, which permit the password to be given on the command line as --password=*pass_val* or -p*pass_val*. (That practice is insecure and should be avoided, however.)

  For the remove command, removes the password from the login path.

- --port=*port_num*, -P *port_num*

  For the set command, the TCP/IP port number to write to the login path. For the remove command, removes the port number from the login path.

- --socket=*file_name*, -S *file_name*

  For the set command, the Unix socket file name to write to the login path. For the remove command, removes the socket file from the login path.

- --user=*user_name*, -u *user_name*

  For the set command, the user name to write to the login path. For the remove command, removes the user name from the login path.

- --verbose, -v

  Verbose mode. Print more information about what the program does.

- --version, -V

  Display version information and exit.

- --warn, -w

  For the set command, warn and prompt the user for confirmation if the command attempts to overwrite an existing login path. This option is enabled by default; use --skip-warn to disable it.

4.6.7 mysqlbinlog — Utility for Processing Binary Log Files

The server's binary log consists of files containing "events" that describe modifications to database contents. The server writes these files in binary format. To display their contents in text format, use the mysqlbinlog utility. You can also use mysqlbinlog to display the contents of relay log files written by a slave server in a replication setup because relay logs have the same format as binary logs. The binary log and relay log are discussed further in Section 5.2.4, "The Binary Log", and Section 16.2.2, "Replication Relay and Status Logs".

Invoke mysqlbinlog like this:

```
shell> mysqlbinlog [options] log_file ...
```

For example, to display the contents of the binary log file named binlog.000003, use this command:

```
shell> mysqlbinlog binlog.0000003
```

The output includes events contained in binlog.000003. For statement-based logging, event information includes the SQL statement, the ID of the server on which it was executed, the timestamp when the statement was executed, how much time it took, and so forth. For row-based logging, the event indicates a row change rather than an SQL statement. See Section 16.1.2, “Replication Formats”, for information about logging modes.

Events are preceded by header comments that provide additional information. For example:

```
# at 141
#100309  9:28:36 server id 123  end_log_pos 245
  Query thread_id=3350  exec_time=11  error_code=0
```

In the first line, the number following at indicates the starting position of the event in the binary log file.

The second line starts with a date and time indicating when the statement started on the server where the event originated. For replication, this timestamp is propagated to slave servers. server id is the server_id value of the server where the event originated. end_log_pos indicates where the next event starts (that is, it is the end position of the current event + 1). thread_id indicates which thread executed the event. exec_time is the time spent executing the event, on a master server. On a slave, it is the difference of the end execution time on the slave minus the beginning execution time on the master. The difference serves as an indicator of how much replication lags behind the master. error_code indicates the result from executing the event. Zero means that no error occurred.

The output from mysqlbinlog can be re-executed (for example, by using it as input to mysql) to redo the statements in the log. This is useful for recovery operations after a server crash. For other usage examples, see the discussion later in this section and in Section 7.5, “Point-in-Time (Incremental) Recovery Using the Binary Log”.

Normally, you use mysqlbinlog to read binary log files directly and apply them to the local MySQL server. It is also possible to read binary logs from a remote server by using the --read-from-remote-server option. To read remote binary logs, the connection parameter options can be given to indicate how to connect to the server. These options are --host, --password, --port, --protocol, --socket, and --user; they are ignored except when you also use the --read-from-remote-server option.

mysqlbinlog supports the following options, which can be specified on the command line or in the [mysqlbinlog] and [client] groups of an option file. For information about option files, see Section 4.2.3.3, “Using Option Files”.

Table 4.15 mysqlbinlog Options

Format	Option File	Description	Introduced
--base64-output=value	base64-output	Print binary log entries using base-64 encoding	
--bind-address=ip_address	bind-address	Use the specified network interface to connect to the MySQL Server	
--binlog-row-event-max-size=#	binlog-row-event-max-size	Binary log max event size	
--character-sets-dir=path	character-sets-dir	The directory where character sets are installed	
--database=db_name	database	List entries for just this database	

Format	Option File	Description	Introduced
--debug[=debug_options]	debug	Write a debugging log	
--debug-check	debug-check	Print debugging information when the program exits	
--debug-info	debug-info	Print debugging information, memory and CPU statistics when the program exits	
--default-auth=plugin	default-auth=plugin	The authentication plugin to use	
--defaults-extra-file=file_name		Read option file in addition to the usual option files	
--defaults-file=file_name		Read only the given option file	
--defaults-group-suffix=str		Option group suffix value	
--disable-log-bin	disable-log-bin	Disable binary logging	
--exclude-gtids=gtid_set	exclude-gtids	Do not show any of the groups in the GTID set provided	
--force-if-open	force-if-open	Read binary log files even if open or not closed properly	
--force-read	force-read	If mysqlbinlog reads a binary log event that it does not recognize, it prints a warning	
--help		Display help message and exit	
--hexdump	hexdump	Display a hex dump of the log in comments	
--host=host_name	host	Connect to the MySQL server on the given host	
--idempotent	idempotent	Cause the server to use idempotent mode while processing binary log updates from this session only	5.7.0
--include-gtids=gtid_set	include-gtids	Show only the groups in the GTID set provided	
--local-load=path	local-load	Prepare local temporary files for LOAD DATA INFILE in the specified directory	
--login-path=name		Read login path options from .mylogin.cnf	
--no-defaults		Do not read any option files	
--offset=#	offset	Skip the first N entries in the log	
--password[=password]	password	The password to use when connecting to the server	
--plugin-dir=path	plugin-dir=path	The directory where plugins are located	
--port=port_num	port	The TCP/IP port number to use for the connection	
--print-defaults		Print defaults	
--protocol=type	protocol	The connection protocol to use	
--raw	raw	Write events in raw (binary) format to output files	
--read-from-remote-master=type	read-from-remote-master	Read the binary log from a MySQL master rather than reading a local log file	

Format	Option File	Description	Introduced
--read-from-remote-server	read-from-remote-server	Read binary log from MySQL server rather than local log file	
--result-file=name	result-file	Direct output to the given file	
--rewrite-db='oldname->newname'	rewrite-db	Create rewrite rules for databases when playing back from logs written in row-based format. Can be used multiple times.	5.7.1
--secure-auth	secure-auth	Do not send passwords to the server in old (pre-4.1.1) format	5.7.4
--server-id=id	server-id	Extract only those events created by the server having the given server ID	
--set-charset=charset_name	set-charset	Add a SET NAMES charset_name statement to the output	
--short-form	short-form	Display only the statements contained in the log	
--skip-gtids[=true|false]	skip-gtids	Do not print any GTIDs; use this when writing a dump file from binary logs containing GTIDs.	
--socket=path	socket	For connections to localhost	
--ssl[=TRUE|FALSE]		Enable an SSL connection to the server. This option is set to TRUE when any other SSL option is used, and so is normally not needed.	5.7.3
--ssl-ca=file_name		The path to a file that contains a list of trusted SSL CAs	5.7.3
--ssl-capath=dir_name		The path to a directory that contains trusted SSL CA certificates in PEM format	5.7.3
--ssl-cert=file_name		The name of the SSL certificate file to use for establishing a secure connection	5.7.3
--ssl-cipher=cipher_list		A list of allowable ciphers to use for SSL encryption	5.7.3
--ssl-crl=file_name		The path to a file that contains certificate revocation lists	
--ssl-crlpath=dir_name		The path to a directory that contains certificate revocation list files	
--ssl-key=file_name		The name of the SSL key file to use for establishing a secure connection	5.7.3
--ssl-verify-server-cert		The server's Common Name value in its certificate is verified against the host name used when connecting to the server	5.7.3
--start-datetime=datetime	start-datetime	Read binary log from first event with timestamp equal to or later than datetime argument	
--start-position=#	start-position	Read binary log from first event with position equal to or greater than argument	
--stop-datetime=datetime	stop-datetime	Stop reading binary log at first event with timestamp equal to or greater than datetime argument	
--stop-never	stop-never	Stay connected to server after reading last binary log file	

Format	Option File	Description	Introduced
--stop-never-slave-server-id=#	stop-never-slave-server-id	Slave server ID to report when connecting to server	
--stop-position=#	stop-position	Stop reading binary log at first event with position equal to or greater than argument	
--to-last-log	to-last-log	Do not stop at the end of requested binary log from a MySQL server, but rather continue printing to end of last binary log	
--user=user_name,	user	MySQL user name to use when connecting to server	
--verbose		Reconstruct row events as SQL statements	
--verify-binlog-checksum		Verify checksums in binary log	
--version		Display version information and exit	

- `--help`, `-?`

  Display a help message and exit.

- `--base64-output=value`

  This option determines when events should be displayed encoded as base-64 strings using `BINLOG` statements. The option has these permissible values (not case sensitive):

  - `AUTO` ("automatic") or `UNSPEC` ("unspecified") displays `BINLOG` statements automatically when necessary (that is, for format description events and row events). If no `--base64-output` option is given, the effect is the same as `--base64-output=AUTO`.

    Note

    Automatic `BINLOG` display is the only safe behavior if you intend to use the output of `mysqlbinlog` to re-execute binary log file contents. The other option values are intended only for debugging or testing purposes because they may produce output that does not include all events in executable form.

  - `NEVER` causes `BINLOG` statements not to be displayed. `mysqlbinlog` exits with an error if a row event is found that must be displayed using `BINLOG`.

  - `DECODE-ROWS` specifies to `mysqlbinlog` that you intend for row events to be decoded and displayed as commented SQL statements by also specifying the `--verbose` option. Like `NEVER`, `DECODE-ROWS` suppresses display of `BINLOG` statements, but unlike `NEVER`, it does not exit with an error if a row event is found.

  For examples that show the effect of `--base64-output` and `--verbose` on row event output, see Section 4.6.7.2, "`mysqlbinlog` Row Event Display".

- `--bind-address=ip_address`

  On a computer having multiple network interfaces, this option can be used to select which interface is employed when connecting to the MySQL server.

- `--binlog-row-event-max-size=N`

Command-Line Format	--binlog-row-event-max-size=#	
Option-File Format	binlog-row-event-max-size	
	Permitted Values	
	Platform Bit Size	64
	Type	numeric
	Default	4294967040
	Range	256 .. 18446744073709547520

Specify the maximum size of a row-based binary log event, in bytes. Rows are grouped into events smaller than this size if possible. The value should be a multiple of 256. The default is 4GB.

- --character-sets-dir=*path*

  The directory where character sets are installed. See Section 10.5, “Character Set Configuration”.

- --database=*db_name*, -d *db_name*

  This option causes mysqlbinlog to output entries from the binary log (local log only) that occur while *db_name* is been selected as the default database by USE.

  The --database option for mysqlbinlog is similar to the --binlog-do-db option for mysqld, but can be used to specify only one database. If --database is given multiple times, only the last instance is used.

  The effects of this option depend on whether the statement-based or row-based logging format is in use, in the same way that the effects of --binlog-do-db depend on whether statement-based or row-based logging is in use.

  Statement-based logging. The --database option works as follows:

  - While *db_name* is the default database, statements are output whether they modify tables in *db_name* or a different database.

  - Unless *db_name* is selected as the default database, statements are not output, even if they modify tables in *db_name*.

  - There is an exception for CREATE DATABASE, ALTER DATABASE, and DROP DATABASE. The database being *created, altered, or dropped* is considered to be the default database when determining whether to output the statement.

  Suppose that the binary log was created by executing these statements using statement-based-logging:

```
INSERT INTO test.t1 (i) VALUES(100);
INSERT INTO db2.t2 (j)  VALUES(200);
USE test;
INSERT INTO test.t1 (i) VALUES(101);
INSERT INTO t1 (i)      VALUES(102);
INSERT INTO db2.t2 (j)  VALUES(201);
USE db2;
INSERT INTO test.t1 (i) VALUES(103);
INSERT INTO db2.t2 (j)  VALUES(202);
INSERT INTO t2 (j)      VALUES(203);
```

mysqlbinlog --database=test does not output the first two INSERT statements because there is no default database. It outputs the three INSERT statements following USE test, but not the three INSERT statements following USE db2.

mysqlbinlog --database=db2 does not output the first two INSERT statements because there is no default database. It does not output the three INSERT statements following USE test, but does output the three INSERT statements following USE db2.

Row-based logging. mysqlbinlog outputs only entries that change tables belonging to *db_name*. The default database has no effect on this. Suppose that the binary log just described was created using row-based logging rather than statement-based logging. mysqlbinlog --database=test outputs only those entries that modify t1 in the test database, regardless of whether USE was issued or what the default database is.

If a server is running with binlog_format set to MIXED and you want it to be possible to use mysqlbinlog with the --database option, you must ensure that tables that are modified are in the database selected by USE. (In particular, no cross-database updates should be used.)

Prior to MySQL 5.7.1, the --database option did not work correctly with a log written by a GTID-enabled MySQL server. (Bug #15912728)

When used together with the --rewrite-db option (available in MySQL 5.7.1 and later), the --rewrite-db option is applied first; then the --database option is applied, using the rewritten database name. The order in which the options are provided makes no difference in this regard.

- --debug[=*debug_options*], -# [*debug_options*]

  Write a debugging log. A typical *debug_options* string is d:t:o,*file_name*. The default is d:t:o,/tmp/mysqlbinlog.trace.

- --debug-check

  Print some debugging information when the program exits.

- --debug-info

  Print debugging information and memory and CPU usage statistics when the program exits.

- --default-auth=*plugin*

  The client-side authentication plugin to use. See Section 6.3.8, “Pluggable Authentication”.

- --defaults-extra-file=*file_name*

  Read this option file after the global option file but (on Unix) before the user option file. If the file does not exist or is otherwise inaccessible, an error occurs. *file_name* is interpreted relative to the current directory if given as a relative path name rather than a full path name.

- --defaults-file=*file_name*

  Use only the given option file. If the file does not exist or is otherwise inaccessible, an error occurs. *file_name* is interpreted relative to the current directory if given as a relative path name rather than a full path name.

- --defaults-group-suffix=*str*

Read not only the usual option groups, but also groups with the usual names and a suffix of *str*. For example, mysqlbinlog normally reads the [client] and [mysqlbinlog] groups. If the --defaults-group-suffix=_other option is given, mysqlbinlog also reads the [client_other] and [mysqlbinlog_other] groups.

- --disable-log-bin, -D

  Disable binary logging. This is useful for avoiding an endless loop if you use the --to-last-log option and are sending the output to the same MySQL server. This option also is useful when restoring after a crash to avoid duplication of the statements you have logged.

  This option requires that you have the SUPER privilege. It causes mysqlbinlog to include a SET sql_log_bin = 0 statement in its output to disable binary logging of the remaining output. The SET statement is ineffective unless you have the SUPER privilege.

- --exclude-gtids=*gtid_set*

  Do not display any of the groups listed in the *gtid_set*.

- --force-if-open, -F

  Read binary log files even if they are open or were not closed properly.

- --force-read, -f

  With this option, if mysqlbinlog reads a binary log event that it does not recognize, it prints a warning, ignores the event, and continues. Without this option, mysqlbinlog stops if it reads such an event.

- --hexdump, -H

  Display a hex dump of the log in comments, as described in Section 4.6.7.1, “mysqlbinlog Hex Dump Format”. The hex output can be helpful for replication debugging.

- --host=*host_name*, -h *host_name*

  Get the binary log from the MySQL server on the given host.

- --idempotent

  Tell the MySQL Server to use idempotent mode while processing updates; this causes suppression of any duplicate-key or key-not-found errors that the server encounters in the current session while processing updates. This option may prove useful whenever it is desirable or necessary to replay one or more binary logs to a MySQL Server which may not contain all of the data to which the logs refer.

  The scope of effect for this option includes the current mysqlbinlog client and session only.

  The --idempotent option was introduced in MySQL 5.7.0.

- --include-gtids=*gtid_set*

  Display only the groups listed in the *gtid_set*.

- --local-load=*path*, -l *path*

  Prepare local temporary files for LOAD DATA INFILE in the specified directory.

> ⚠ **Important**
>
> These temporary files are not automatically removed by mysqlbinlog or any other MySQL program.

- `--login-path=name`

  Read options from the named login path in the `.mylogin.cnf` login file. A “login path” is an option group that permits only a limited set of options: `host`, `user`, and `password`. Think of a login path as a set of values that indicate the server host and the credentials for authenticating with the server. To create the login file, use the `mysql_config_editor` utility. See Section 4.6.6, “mysql_config_editor — MySQL Configuration Utility”.

- `--no-defaults`

  Do not read any option files. If program startup fails due to reading unknown options from an option file, `--no-defaults` can be used to prevent them from being read.

  The exception is that the `.mylogin.cnf` file, if it exists, is read in all cases. This permits passwords to be specified in a safer way than on the command line even when `--no-defaults` is used. (`.mylogin.cnf` is created by the `mysql_config_editor` utility. See Section 4.6.6, “mysql_config_editor — MySQL Configuration Utility”.)

- `--offset=N`, `-o N`

  Skip the first *N* entries in the log.

- `--password[=password]`, `-p[password]`

  The password to use when connecting to the server. If you use the short option form (`-p`), you *cannot* have a space between the option and the password. If you omit the *password* value following the `--password` or `-p` option on the command line, mysqlbinlog prompts for one.

  Specifying a password on the command line should be considered insecure. See Section 6.1.2.1, “End-User Guidelines for Password Security”. You can use an option file to avoid giving the password on the command line.

- `--plugin-dir=path`

  The directory in which to look for plugins. It may be necessary to specify this option if the `--default-auth` option is used to specify an authentication plugin but mysqlbinlog does not find it. See Section 6.3.8, “Pluggable Authentication”.

- `--port=port_num`, `-P port_num`

  The TCP/IP port number to use for connecting to a remote server.

- `--print-defaults`

  Print the program name and all options that it gets from option files.

- `--protocol={TCP|SOCKET|PIPE|MEMORY}`

  The connection protocol to use for connecting to the server. It is useful when the other connection parameters normally would cause a protocol to be used other than the one you want. For details on the permissible values, see Section 4.2.2, “Connecting to the MySQL Server”.

- `--raw`

By default, mysqlbinlog reads binary log files and writes events in text format. The --raw option tells mysqlbinlog to write them in their original binary format. Its use requires that --read-from-remote-server also be used because the files are requested from a server. mysqlbinlog writes one output file for each file read from the server. The --raw option can be used to make a backup of a server's binary log. With the --stop-never option, the backup is "live" because mysqlbinlog stays connected to the server. By default, output files are written in the current directory with the same names as the original log files. Output file names can be modified using the --result-file option. For more information, see Section 4.6.7.3, "Using mysqlbinlog to Back Up Binary Log Files".

- --read-from-remote-master=*type*

  Read binary logs from a MySQL server with the COM_BINLOG_DUMP or COM_BINLOG_DUMP_GTID commands by setting the option value to either BINLOG-DUMP-NON-GTIDS or BINLOG-DUMP-GTIDS, respectively. If --read-from-remote-master=BINLOG-DUMP-GTIDS is combined with --exclude-gtids, transactions can be filtered out on the master, avoiding unnecessary network traffic.

  See also the description for --read-from-remote-server.

- --read-from-remote-server, -R

  Read the binary log from a MySQL server rather than reading a local log file. Any connection parameter options are ignored unless this option is given as well. These options are --host, --password, --port, --protocol, --socket, and --user.

  This option requires that the remote server be running. It works only for binary log files on the remote server, not relay log files.

  This option is like --read-from-remote-master=BINLOG-DUMP-NON-GTIDS.

- --result-file=*name*, -r *name*

  Without the --raw option, this option indicates the file to which mysqlbinlog writes text output. With --raw, mysqlbinlog writes one binary output file for each log file transferred from the server, writing them by default in the current directory using the same names as the original log file. In this case, the --result-file option value is treated as a prefix that modifies output file names.

- --rewrite-db='*dboldname*->*dbnewname*'

  When reading from a row-based log, rewrite all occurrences of *dboldname* to *dbnewname*. For use when restoring tables logged using the row-based format to a database having a different name from the original database.

  The rewrite rule employed as a value for this option is a string having the form '*dboldname*->*dbnewname*', as shown previously, and for this reason must be enclosed by quotation marks. These can be single or double quotation marks (' or ").

  To employ multiple rewrite rules, specify the option multiple times, as shown here:

  ```
  shell> mysqlbinlog --rewrite-db='dbcurrent->dbold' --rewrite-db='dbtest->dbcurrent' \
                     binlog.00001 > /tmp/statements.sql
  ```

  When used together with the --database option, the --rewrite-db option is applied first; then --database option is applied, using the rewritten database name. The order in which the options are provided makes no difference in this regard.

This means that, for example, if mysqlbinlog is started with --rewrite-db='mydb->yourdb' --database=yourdb, then all updates to any tables in databases mydb and yourdb are included in the output. On the other hand, if it is started with --rewrite-db='mydb->yourdb' --database=mydb, then mysqlbinlog outputs no statements at all: since all updates to mydb are first rewritten as updates to yourdb before applying the --database option, there remain no updates that match --database=mydb.

This option was added in MySQL 5.7.1.

- --secure-auth

  Do not send passwords to the server in old (pre-4.1) format. This prevents connections except for servers that use the newer password format. This option is enabled by default; use --skip-secure-auth to disable it. This option was added in MySQL 5.7.4.

  Note

  Passwords that use the pre-4.1 hashing method are less secure than passwords that use the native password hashing method and should be avoided. Pre-4.1 passwords are deprecated and support for them will be removed in a future MySQL release. For account upgrade instructions, see Section 6.3.9.3, “Migrating Away from Pre-4.1 Password Hashing and the mysql_old_password Plugin”.

- --server-id=*id*

  Display only those events created by the server having the given server ID.

- --set-charset=*charset_name*

  Add a SET NAMES *charset_name* statement to the output to specify the character set to be used for processing log files.

- --short-form, -s

  Display only the statements contained in the log, without any extra information or row-based events. This is for testing only, and should not be used in production systems.

- --skip-gtids[=(true|false)]

  Do not display any GTIDs in the output. This is needed when writing to a dump file from one or more binary logs containing GTIDs, as shown in this example:

```
shell> mysqlbinlog --skip-gtids binlog.000001 >  /tmp/dump.sql
shell> mysqlbinlog --skip-gtids binlog.000002 >> /tmp/dump.sql
shell> mysql -u root -p -e "source /tmp/dump.sql"
```

  The use of this option is otherwise not normally recommended in production.

- --socket=*path*, -S *path*

  For connections to localhost, the Unix socket file to use, or, on Windows, the name of the named pipe to use.

- --ssl*

  Options that begin with --ssl specify whether to connect to the server using SSL and indicate where to find SSL keys and certificates. See Section 6.3.11.4, “SSL Command Options”.

- `--start-datetime=datetime`

  Start reading the binary log at the first event having a timestamp equal to or later than the *datetime* argument. The *datetime* value is relative to the local time zone on the machine where you run `mysqlbinlog`. The value should be in a format accepted for the `DATETIME` or `TIMESTAMP` data types. For example:

  ```
  shell> mysqlbinlog --start-datetime="2005-12-25 11:25:56" binlog.000003
  ```

  This option is useful for point-in-time recovery. See Section 7.3, “Example Backup and Recovery Strategy”.

- `--start-position=N`, `-j N`

  Start reading the binary log at the first event having a position equal to or greater than *N*. This option applies to the first log file named on the command line.

  This option is useful for point-in-time recovery. See Section 7.3, “Example Backup and Recovery Strategy”.

- `--stop-datetime=datetime`

  Stop reading the binary log at the first event having a timestamp equal to or later than the *datetime* argument. This option is useful for point-in-time recovery. See the description of the `--start-datetime` option for information about the *datetime* value.

  This option is useful for point-in-time recovery. See Section 7.3, “Example Backup and Recovery Strategy”.

- `--stop-never`

  This option is used with `--read-from-remote-server`. It tells `mysqlbinlog` to remain connected to the server. Otherwise `mysqlbinlog` exits when the last log file has been transferred from the server. `--stop-never` implies `--to-last-log`, so only the first log file to transfer need be named on the command line.

  `--stop-never` is commonly used with `--raw` to make a live binary log backup, but also can be used without `--raw` to maintain a continuous text display of log events as the server generates them.

- `--stop-never-slave-server-id=id`

  With `--stop-never`, `mysqlbinlog` reports a server ID of 65535 when it connects to the server. `--stop-never-slave-server-id` explicitly specifies the server ID to report. It can be used to avoid a conflict with the ID of a slave server or another `mysqlbinlog` process. See Section 4.6.7.4, “Specifying the `mysqlbinlog` Server ID”.

- `--stop-position=N`

  Stop reading the binary log at the first event having a position equal to or greater than *N*. This option applies to the last log file named on the command line.

  This option is useful for point-in-time recovery. See Section 7.3, “Example Backup and Recovery Strategy”.

- `--to-last-log`, `-t`

Do not stop at the end of the requested binary log from a MySQL server, but rather continue printing until the end of the last binary log. If you send the output to the same MySQL server, this may lead to an endless loop. This option requires `--read-from-remote-server`.

- `--user=user_name`, `-u user_name`

  The MySQL user name to use when connecting to a remote server.

- `--verbose`, `-v`

  Reconstruct row events and display them as commented SQL statements. If this option is given twice, the output includes comments to indicate column data types and some metadata.

  For examples that show the effect of `--base64-output` and `--verbose` on row event output, see Section 4.6.7.2, “`mysqlbinlog` Row Event Display”.

- `--verify-binlog-checksum`, `-c`

  Verify checksums in binary log files.

- `--version`, `-V`

  Display version information and exit.

  In MySQL 5.7.1 and later, the `mysqlbinlog` version number shown when using this option is 3.4. (Bug #15894381, Bug #67643)

You can also set the following variable by using `--var_name=value` syntax:

- `open_files_limit`

  Specify the number of open file descriptors to reserve.

You can pipe the output of `mysqlbinlog` into the `mysql` client to execute the events contained in the binary log. This technique is used to recover from a crash when you have an old backup (see Section 7.5, “Point-in-Time (Incremental) Recovery Using the Binary Log”). For example:

```
shell> mysqlbinlog binlog.000001 | mysql -u root -p
```

Or:

```
shell> mysqlbinlog binlog.[0-9]* | mysql -u root -p
```

If the statements produced by `mysqlbinlog` may contain `BLOB` values, these may cause problems when `mysql` processes them. In this case, invoke `mysql` with the `--binary-mode` option.

You can also redirect the output of `mysqlbinlog` to a text file instead, if you need to modify the statement log first (for example, to remove statements that you do not want to execute for some reason). After editing the file, execute the statements that it contains by using it as input to the `mysql` program:

```
shell> mysqlbinlog binlog.000001 > tmpfile
shell> ... edit tmpfile ...
shell> mysql -u root -p < tmpfile
```

When `mysqlbinlog` is invoked with the `--start-position` option, it displays only those events with an offset in the binary log greater than or equal to a given position (the given position must match the start of one event). It also has options to stop and start when it sees an event with a given date and time. This enables you to perform point-in-time recovery using the --stop-datetime option (to be able to say, for example, “roll forward my databases to how they were today at 10:30 a.m.”).

If you have more than one binary log to execute on the MySQL server, the safe method is to process them all using a single connection to the server. Here is an example that demonstrates what may be *unsafe*:

```
shell> mysqlbinlog binlog.000001 | mysql -u root -p # DANGER!!
shell> mysqlbinlog binlog.000002 | mysql -u root -p # DANGER!!
```

Processing binary logs this way using multiple connections to the server causes problems if the first log file contains a CREATE TEMPORARY TABLE statement and the second log contains a statement that uses the temporary table. When the first mysql process terminates, the server drops the temporary table. When the second mysql process attempts to use the table, the server reports “unknown table.”

To avoid problems like this, use a *single* mysql process to execute the contents of all binary logs that you want to process. Here is one way to do so:

```
shell> mysqlbinlog binlog.000001 binlog.000002 | mysql -u root -p
```

Another approach is to write all the logs to a single file and then process the file:

```
shell> mysqlbinlog binlog.000001 >  /tmp/statements.sql
shell> mysqlbinlog binlog.000002 >> /tmp/statements.sql
shell> mysql -u root -p -e "source /tmp/statements.sql"
```

mysqlbinlog can produce output that reproduces a LOAD DATA INFILE operation without the original data file. mysqlbinlog copies the data to a temporary file and writes a LOAD DATA LOCAL INFILE statement that refers to the file. The default location of the directory where these files are written is system-specific. To specify a directory explicitly, use the --local-load option.

Because mysqlbinlog converts LOAD DATA INFILE statements to LOAD DATA LOCAL INFILE statements (that is, it adds LOCAL), both the client and the server that you use to process the statements must be configured with the LOCAL capability enabled. See Section 6.1.6, “Security Issues with LOAD DATA LOCAL”.

Warning

The temporary files created for LOAD DATA LOCAL statements are *not* automatically deleted because they are needed until you actually execute those statements. You should delete the temporary files yourself after you no longer need the statement log. The files can be found in the temporary file directory and have names like *original_file_name-#-#*.

4.6.7.1 mysqlbinlog Hex Dump Format

The --hexdump option causes mysqlbinlog to produce a hex dump of the binary log contents:

```
shell> mysqlbinlog --hexdump master-bin.000001
```

The hex output consists of comment lines beginning with #, so the output might look like this for the preceding command:

```
/*!40019 SET @@session.max_insert_delayed_threads=0*/;
/*!50003 SET @OLD_COMPLETION_TYPE=@@COMPLETION_TYPE,COMPLETION_TYPE=0*/;
# at 4
```

```
#051024 17:24:13 server id 1  end_log_pos 98
# Position  Timestamp   Type   Master ID        Size      Master Pos    Flags
# 00000004 9d fc 5c 43   0f   01 00 00 00   5e 00 00 00   62 00 00 00   00 00
# 00000017 04 00 35 2e 30 2e 31 35  2d 64 65 62 75 67 2d 6c |..5.0.15.debug.l|
# 00000027 6f 67 00 00 00 00 00 00  00 00 00 00 00 00 00 00 |og..............|
# 00000037 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 |................|
# 00000047 00 00 00 00 9d fc 5c 43  13 38 0d 00 08 00 12 00 |.......C.8......|
# 00000057 04 04 04 04 12 00 00 4b  00 04 1a                |.......K...|
#       Start: binlog v 4, server v 5.0.15-debug-log created 051024 17:24:13
#       at startup
ROLLBACK;
```

Hex dump output currently contains the elements in the following list. This format is subject to change. (For more information about binary log format, see MySQL Internals: The Binary Log.

- Position: The byte position within the log file.
- Timestamp: The event timestamp. In the example shown, '9d fc 5c 43' is the representation of '051024 17:24:13' in hexadecimal.
- Type: The event type code. In the example shown, '0f' indicates a FORMAT_DESCRIPTION_EVENT. The following table lists the possible type codes.

Type	Name	Meaning
00	UNKNOWN_EVENT	This event should never be present in the log.
01	START_EVENT_V3	This indicates the start of a log file written by MySQL 4 or earlier.
02	QUERY_EVENT	The most common type of events. These contain statements executed on the master.
03	STOP_EVENT	Indicates that master has stopped.
04	ROTATE_EVENT	Written when the master switches to a new log file.
05	INTVAR_EVENT	Used for AUTO_INCREMENT values or when the LAST_INSERT_ID() function is used in the statement.
06	LOAD_EVENT	Used for LOAD DATA INFILE in MySQL 3.23.
07	SLAVE_EVENT	Reserved for future use.
08	CREATE_FILE_EVENT	Used for LOAD DATA INFILE statements. This indicates the start of execution of such a statement. A temporary file is created on the slave. Used in MySQL 4 only.
09	APPEND_BLOCK_EVENT	Contains data for use in a LOAD DATA INFILE statement. The data is stored in the temporary file on the slave.
0a	EXEC_LOAD_EVENT	Used for LOAD DATA INFILE statements. The contents of the temporary file is stored in the table on the slave. Used in MySQL 4 only.
0b	DELETE_FILE_EVENT	Rollback of a LOAD DATA INFILE statement. The temporary file should be deleted on the slave.
0c	NEW_LOAD_EVENT	Used for LOAD DATA INFILE in MySQL 4 and earlier.
0d	RAND_EVENT	Used to send information about random values if the RAND() function is used in the statement.
0e	USER_VAR_EVENT	Used to replicate user variables.
0f	FORMAT_DESCRIPTION_EVENT	This indicates the start of a log file written by MySQL 5 or later.

Type	Name	Meaning
10	XID_EVENT	Event indicating commit of an XA transaction.
11	BEGIN_LOAD_QUERY_EVENT	Used for LOAD DATA INFILE statements in MySQL 5 and later.
12	EXECUTE_LOAD_QUERY_EVENT	Used for LOAD DATA INFILE statements in MySQL 5 and later.
13	TABLE_MAP_EVENT	Information about a table definition. Used in MySQL 5.1.5 and later.
14	PRE_GA_WRITE_ROWS_EVENT	Row data for a single table that should be created. Used in MySQL 5.1.5 to 5.1.17.
15	PRE_GA_UPDATE_ROWS_EVENT	Row data for a single table that needs to be updated. Used in MySQL 5.1.5 to 5.1.17.
16	PRE_GA_DELETE_ROWS_EVENT	Row data for a single table that should be deleted. Used in MySQL 5.1.5 to 5.1.17.
17	WRITE_ROWS_EVENT	Row data for a single table that should be created. Used in MySQL 5.1.18 and later.
18	UPDATE_ROWS_EVENT	Row data for a single table that needs to be updated. Used in MySQL 5.1.18 and later.
19	DELETE_ROWS_EVENT	Row data for a single table that should be deleted. Used in MySQL 5.1.18 and later.
1a	INCIDENT_EVENT	Something out of the ordinary happened. Added in MySQL 5.1.18.

- Master ID: The server ID of the master that created the event.
- Size: The size in bytes of the event.
- Master Pos: The position of the next event in the original master log file.
- Flags: 16 flags. Currently, the following flags are used. The others are reserved for future use.

Flag	Name	Meaning
01	LOG_EVENT_BINLOG_IN_USE_F	Log file correctly closed. (Used only in FORMAT_DESCRIPTION_EVENT.) If this flag is set (if the flags are, for example, '01 00') in a FORMAT_DESCRIPTION_EVENT, the log file has not been properly closed. Most probably this is because of a master crash (for example, due to power failure).
02		Reserved for future use.
04	LOG_EVENT_THREAD_SPECIFIC_F	Set if the event is dependent on the connection it was executed in (for example, '04 00'), for example, if the event uses temporary tables.
08	LOG_EVENT_SUPPRESS_USE_F	Set in some circumstances when the event is not dependent on the default database.

4.6.7.2 mysqlbinlog Row Event Display

The following examples illustrate how mysqlbinlog displays row events that specify data modifications. These correspond to events with the WRITE_ROWS_EVENT, UPDATE_ROWS_EVENT, and

DELETE_ROWS_EVENT type codes. The --base64-output=DECODE-ROWS and --verbose options may be used to affect row event output.

Suppose that the server is using row-based binary logging and that you execute the following sequence of statements:

```
CREATE TABLE t
(
  id   INT NOT NULL,
  name VARCHAR(20) NOT NULL,
  date DATE NULL
) ENGINE = InnoDB;

START TRANSACTION;
INSERT INTO t VALUES(1, 'apple', NULL);
UPDATE t SET name = 'pear', date = '2009-01-01' WHERE id = 1;
DELETE FROM t WHERE id = 1;
COMMIT;
```

By default, mysqlbinlog displays row events encoded as base-64 strings using BINLOG statements. Omitting extraneous lines, the output for the row events produced by the preceding statement sequence looks like this:

```
shell> mysqlbinlog log_file
...
# at 218
#080828 15:03:08 server id 1  end_log_pos 258   Write_rows: table id 17 flags: STMT_END_F

BINLOG '
fAS3SBMBAAAALAAAANoAAAAAABEAAAAAAAAABHRlc3QAAXQAAwMPCgIUAAQ=
fAS3SBcBAAAAKAAAAAIBAAAQABEAAAAAAAEAA//8AQAAAAVhcHBsZQ==
'/*!*/;
...
# at 302
#080828 15:03:08 server id 1  end_log_pos 356   Update_rows: table id 17 flags: STMT_END_F

BINLOG '
fAS3SBMBAAAALAAAAC4BAAAAABEAAAAAAAAABHRlc3QAAXQAAwMPCgIUAAQ=
fAS3SBgBAAAANgAAAGQBAAAQABEAAAAAAAEAA////AEAAAAFYXBwbGX4AQAAAARwZWFyIbIP
'/*!*/;
...
# at 400
#080828 15:03:08 server id 1  end_log_pos 442   Delete_rows: table id 17 flags: STMT_END_F

BINLOG '
fAS3SBMBAAAALAAAAJABAAAAABEAAAAAAAAABHRlc3QAAXQAAwMPCgIUAAQ=
fAS3SBkBAAAAKgAAALoBAAAQABEAAAAAAAEAA//4AQAAAARwZWFyIbIP
'/*!*/;
```

To see the row events as comments in the form of “pseudo-SQL” statements, run mysqlbinlog with the --verbose or -v option. The output will contain lines beginning with ###:

```
shell> mysqlbinlog -v log_file
...
# at 218
#080828 15:03:08 server id 1  end_log_pos 258   Write_rows: table id 17 flags: STMT_END_F

BINLOG '
fAS3SBMBAAAALAAAANoAAAAAABEAAAAAAAAABHRlc3QAAXQAAwMPCgIUAAQ=
fAS3SBcBAAAAKAAAAAIBAAAQABEAAAAAAAEAA//8AQAAAAVhcHBsZQ==
'/*!*/;
### INSERT INTO test.t
### SET
###   @1=1
###   @2='apple'
###   @3=NULL
...
# at 302
#080828 15:03:08 server id 1  end_log_pos 356   Update_rows: table id 17 flags: STMT_END_F

BINLOG '
fAS3SBMBAAAALAAAAC4BAAAAABEAAAAAAAAABHRlc3QAAXQAAwMPCgIUAAQ=
fAS3SBgBAAAANgAAAGQBAAAQABEAAAAAAAEAA////AEAAAAFYXBwbGX4AQAAAARwZWFyIbIP
'/*!*/;
### UPDATE test.t
### WHERE
###   @1=1
###   @2='apple'
###   @3=NULL
### SET
###   @1=1
###   @2='pear'
###   @3='2009:01:01'
...
# at 400
#080828 15:03:08 server id 1  end_log_pos 442   Delete_rows: table id 17 flags: STMT_END_F

BINLOG '
fAS3SBMBAAAALAAAAJABAAAAABEAAAAAAAAABHRlc3QAAXQAAwMPCgIUAAQ=
fAS3SBkBAAAAKgAAALoBAAAQABEAAAAAAAEAA//4AQAAAARwZWFyIbIP
'/*!*/;
### DELETE FROM test.t
### WHERE
###   @1=1
###   @2='pear'
###   @3='2009:01:01'
```

Specify --verbose or -v twice to also display data types and some metadata for each column. The output will contain an additional comment following each column change:

```
shell> mysqlbinlog -vv log_file
...
# at 218
#080828 15:03:08 server id 1  end_log_pos 258   Write_rows: table id 17 flags: STMT_END_F

BINLOG '
fAS3SBMBAAAALAAAANoAAAAAABEAAAAAAAAABHRlc3QAAXQAAwMPCgIUAAQ=
fAS3SBcBAAAAKAAAAAIBAAAQABEAAAAAAAEAA//8AQAAAAVhcHBsZQ==
'/*!*/;
### INSERT INTO test.t
### SET
###   @1=1 /* INT meta=0 nullable=0 is_null=0 */
###   @2='apple' /* VARSTRING(20) meta=20 nullable=0 is_null=0 */
###   @3=NULL /* VARSTRING(20) meta=0 nullable=1 is_null=1 */
...
# at 302
#080828 15:03:08 server id 1  end_log_pos 356   Update_rows: table id 17 flags: STMT_END_F

BINLOG '
fAS3SBMBAAAALAAAAC4BAAAAABEAAAAAAAAABHRlc3QAAXQAAwMPCgIUAAQ=
fAS3SBgBAAAANgAAAGQBAAAQABEAAAAAAAEAA////AEAAAAFYXBwbGX4AQAAAARwZWFyIbIP
'/*!*/;
### UPDATE test.t
### WHERE
###   @1=1 /* INT meta=0 nullable=0 is_null=0 */
###   @2='apple' /* VARSTRING(20) meta=20 nullable=0 is_null=0 */
###   @3=NULL /* VARSTRING(20) meta=0 nullable=1 is_null=1 */
### SET
###   @1=1 /* INT meta=0 nullable=0 is_null=0 */
###   @2='pear' /* VARSTRING(20) meta=20 nullable=0 is_null=0 */
###   @3='2009:01:01' /* DATE meta=0 nullable=1 is_null=0 */
...
# at 400
#080828 15:03:08 server id 1  end_log_pos 442   Delete_rows: table id 17 flags: STMT_END_F

BINLOG '
fAS3SBMBAAAALAAAAJABAAAAABEAAAAAAAAAABHRlc3QAAXQAAwMPCgIUAAQ=
fAS3SBkBAAAAKgAAALoBAAAQABEAAAAAAAEAA//4AQAAAARwZWFyIbIP
'/*!*/;
### DELETE FROM test.t
### WHERE
###   @1=1 /* INT meta=0 nullable=0 is_null=0 */
###   @2='pear' /* VARSTRING(20) meta=20 nullable=0 is_null=0 */
###   @3='2009:01:01' /* DATE meta=0 nullable=1 is_null=0 */
```

You can tell mysqlbinlog to suppress the BINLOG statements for row events by using the --base64-output=DECODE-ROWS option. This is similar to --base64-output=NEVER but does not exit with an error if a row event is found. The combination of --base64-output=DECODE-ROWS and --verbose provides a convenient way to see row events only as SQL statements:

```
shell> mysqlbinlog -v --base64-output=DECODE-ROWS log_file
...
# at 218
#080828 15:03:08 server id 1  end_log_pos 258   Write_rows: table id 17 flags: STMT_END_F
### INSERT INTO test.t
### SET
###   @1=1
###   @2='apple'
###   @3=NULL
...
# at 302
#080828 15:03:08 server id 1  end_log_pos 356   Update_rows: table id 17 flags: STMT_END_F
### UPDATE test.t
### WHERE
###   @1=1
###   @2='apple'
###   @3=NULL
### SET
###   @1=1
###   @2='pear'
###   @3='2009:01:01'
...
# at 400
#080828 15:03:08 server id 1  end_log_pos 442   Delete_rows: table id 17 flags: STMT_END_F
### DELETE FROM test.t
### WHERE
###   @1=1
###   @2='pear'
###   @3='2009:01:01'
```

Note

You should not suppress BINLOG statements if you intend to re-execute mysqlbinlog output.

The SQL statements produced by --verbose for row events are much more readable than the corresponding BINLOG statements. However, they do not correspond exactly to the original SQL statements that generated the events. The following limitations apply:

- The original column names are lost and replaced by *@N*, where *N* is a column number.
- Character set information is not available in the binary log, which affects string column display:

- There is no distinction made between corresponding binary and nonbinary string types (BINARY and CHAR, VARBINARY and VARCHAR, BLOB and TEXT). The output uses a data type of STRING for fixed-length strings and VARSTRING for variable-length strings.
- For multi-byte character sets, the maximum number of bytes per character is not present in the binary log, so the length for string types is displayed in bytes rather than in characters. For example, STRING(4) will be used as the data type for values from either of these column types:

  ```
  CHAR(4) CHARACTER SET latin1
  CHAR(2) CHARACTER SET ucs2
  ```

- Due to the storage format for events of type UPDATE_ROWS_EVENT, UPDATE statements are displayed with the WHERE clause preceding the SET clause.

Proper interpretation of row events requires the information from the format description event at the beginning of the binary log. Because mysqlbinlog does not know in advance whether the rest of the log contains row events, by default it displays the format description event using a BINLOG statement in the initial part of the output.

If the binary log is known not to contain any events requiring a BINLOG statement (that is, no row events), the --base64-output=NEVER option can be used to prevent this header from being written.

4.6.7.3 Using mysqlbinlog to Back Up Binary Log Files

By default, mysqlbinlog reads binary log files and displays their contents in text format. This enables you to examine events within the files more easily and to re-execute them (for example, by using the output as input to mysql). mysqlbinlog can read log files directly from the local file system, or, with the --read-from-remote-server option, it can connect to a server and request binary log contents from that server. mysqlbinlog writes text output to its standard output, or to the file named as the value of the --result-file=*file_name* option if that option is given.

mysqlbinlog can read binary log files and write new files containing the same content—that is, in binary format rather than text format. This capability enables you to easily back up a binary log in its original format. mysqlbinlog can make a static backup, backing up a set of log files and stopping when the end of the last file is reached. It can also make a continuous (“live”) backup, staying connected to the server when it reaches the end of the last log file and continuing to copy new events as they are generated. In continuous-backup operation, mysqlbinlog runs until the connection ends (for example, when the server exits) or mysqlbinlog is forcibly terminated. When the connection ends, mysqlbinlog does not wait and retry the connection, unlike a slave replication server. To continue a live backup after the server has been restarted, you must also restart mysqlbinlog.

Binary log backup requires that you invoke mysqlbinlog with two options at minimum:

- The --read-from-remote-server (or -R) option tells mysqlbinlog to connect to a server and request its binary log. (This is similar to a slave replication server connecting to its master server.)
- The --raw option tells mysqlbinlog to write raw (binary) output, not text output.

Along with --read-from-remote-server, it is common to specify other options: --host indicates where the server is running, and you may also need to specify connection options such as --user and --password.

Several other options are useful in conjunction with --raw:

- --stop-never: Stay connected to the server after reaching the end of the last log file and continue to read new events.

- `--stop-never-slave-server-id=id`: The server ID that mysqlbinlog reports to the server when `--stop-never` is used. The default is 65535. This can be used to avoid a conflict with the ID of a slave server or another mysqlbinlog process. See Section 4.6.7.4, “Specifying the mysqlbinlog Server ID”.
- `--result-file`: A prefix for output file names, as described later.

To back up a server's binary log files with mysqlbinlog, you must specify file names that actually exist on the server. If you do not know the names, connect to the server and use the `SHOW BINARY LOGS` statement to see the current names. Suppose that the statement produces this output:

```
mysql> SHOW BINARY LOGS;
+---------------+-----------+
| Log_name      | File_size |
+---------------+-----------+
| binlog.000130 |     27459 |
| binlog.000131 |     13719 |
| binlog.000132 |     43268 |
+---------------+-----------+
```

With that information, you can use mysqlbinlog to back up the binary log to the current directory as follows (enter each command on a single line):

- To make a static backup of `binlog.000130` through `binlog.000132`, use either of these commands:

  ```
  mysqlbinlog --read-from-remote-server --host=host_name --raw
    binlog.000130 binlog.000131 binlog.000132

  mysqlbinlog --read-from-remote-server --host=host_name --raw
    --to-last-log binlog.000130
  ```

  The first command specifies every file name explicitly. The second names only the first file and uses `--to-last-log` to read through the last. A difference between these commands is that if the server happens to open `binlog.000133` before mysqlbinlog reaches the end of `binlog.000132`, the first command will not read it, but the second command will.

- To make a live backup in which mysqlbinlog starts with `binlog.000130` to copy existing log files, then stays connected to copy new events as the server generates them:

  ```
  mysqlbinlog --read-from-remote-server --host=host_name --raw
    --stop-never binlog.000130
  ```

  With `--stop-never`, it is not necessary to specify `--to-last-log` to read to the last log file because that option is implied.

Output File Naming

Without `--raw`, mysqlbinlog produces text output and the `--result-file` option, if given, specifies the name of the single file to which all output is written. With `--raw`, mysqlbinlog writes one binary output file for each log file transferred from the server. By default, mysqlbinlog writes the files in the current directory with the same names as the original log files. To modify the output file names, use the `--result-file` option. In conjunction with `--raw`, the `--result-file` option value is treated as a prefix that modifies the output file names.

Suppose that a server currently has binary log files named `binlog.000999` and up. If you use `mysqlbinlog --raw` to back up the files, the `--result-file` option produces output file names as shown in the following table. You can write the files to a specific directory by beginning the `--result-`

file value with the directory path. If the --result-file value consists only of a directory name, the value must end with the pathname separator character. Output files are overwritten if they exist.

--result-file Option	**Output File Names**
--result-file=x	xbinlog.000999 and up
--result-file=/tmp/	/tmp/binlog.000999 and up
--result-file=/tmp/x	/tmp/xbinlog.000999 and up

Example: mysqldump + mysqlbinlog for Backup and Restore

The following example describes a simple scenario that shows how to use mysqldump and mysqlbinlog together to back up a server's data and binary log, and how to use the backup to restore the server if data loss occurs. The example assumes that the server is running on host *host_name* and its first binary log file is named binlog.000999. Enter each command on a single line.

Use mysqlbinlog to make a continuous backup of the binary log:

```
mysqlbinlog --read-from-remote-server --host=host_name --raw
  --stop-never binlog.000999
```

Use mysqldump to create a dump file as a snapshot of the server's data. Use --all-databases, --events, and --routines to back up all data, and --master-data=2 to include the current binary log coordinates in the dump file.

```
mysqldump --host=host_name --all-databases --events --routines --master-data=2> dump_file
```

Execute the mysqldump command periodically to create newer snapshots as desired.

If data loss occurs (for example, if the server crashes), use the most recent dump file to restore the data:

```
mysql --host=host_name -u root -p < dump_file
```

Then use the binary log backup to re-execute events that were written after the coordinates listed in the dump file. Suppose that the coordinates in the file look like this:

```
-- CHANGE MASTER TO MASTER_LOG_FILE='binlog.001002', MASTER_LOG_POS=27284;
```

If the most recent backed-up log file is named binlog.001004, re-execute the log events like this:

```
mysqlbinlog --start-position=27284 binlog.001002 binlog.001003 binlog.001004
  | mysql --host=host_name -u root -p
```

You might find it easier to copy the backup files (dump file and binary log files) to the server host to make it easier to perform the restore operation, or if MySQL does not allow remote root access.

4.6.7.4 Specifying the mysqlbinlog Server ID

When invoked with the --read-from-remote-server option, mysqlbinlog connects to a MySQL server, specifies a server ID to identify itself, and requests binary log files from the server. You can use mysqlbinlog to request log files from a server in several ways:

- Specify an explicitly named set of files: For each file, mysqlbinlog connects and issues a Binlog dump command. The server sends the file and disconnects. There is one connection per file.

- Specify the beginning file and --to-last-log: mysqlbinlog connects and issues a Binlog dump command for all files. The server sends all files and disconnects.
- Specify the beginning file and --stop-never (which implies --to-last-log): mysqlbinlog connects and issues a Binlog dump command for all files. The server sends all files, but does not disconnect after sending the last one.

With --read-from-remote-server only, mysqlbinlog connects using a server ID of 0, which tells the server to disconnect after sending the last requested log file.

With --read-from-remote-server and --stop-never, mysqlbinlog connects using a nonzero server ID, so the server does not disconnect after sending the last log file. The server ID is 65535 by default, but this can be changed with --stop-never-slave-server-id.

Thus, for the first two ways of requesting files, the server disconnects because mysqlbinlog specifies a server ID of 0. It does not disconnect if --stop-never is given because mysqlbinlog specifies a nonzero server ID.

4.6.8 mysqldumpslow — Summarize Slow Query Log Files

The MySQL slow query log contains information about queries that take a long time to execute (see Section 5.2.5, “The Slow Query Log”). mysqldumpslow parses MySQL slow query log files and prints a summary of their contents.

Normally, mysqldumpslow groups queries that are similar except for the particular values of number and string data values. It “abstracts” these values to N and 'S' when displaying summary output. The -a and -n options can be used to modify value abstracting behavior.

Invoke mysqldumpslow like this:

```
shell> mysqldumpslow [options] [log_file ...]
```

mysqldumpslow supports the following options.

Table 4.16 mysqldumpslow Options

Format	Option File	Description
-a		Do not abstract all numbers to N and strings to S
-n num		Abstract numbers with at least the specified digits
--debug	debug	Write debugging information
-g pattern		Only consider statements that match the pattern
--help		Display help message and exit
-h name		Host name of the server in the log file name
-i name		Name of the server instance
-l		Do not subtract lock time from total time
-r		Reverse the sort order
-s value		How to sort output
-t num		Display only first num queries
--verbose	verbose	Verbose mode

- --help

Display a help message and exit.

- -a

  Do not abstract all numbers to N and strings to 'S'.

- --debug, -d

  Run in debug mode.

- -g *pattern*

  Consider only queries that match the (grep-style) pattern.

- -h *host_name*

  Host name of MySQL server for *-slow.log file name. The value can contain a wildcard. The default is * (match all).

- -i *name*

  Name of server instance (if using mysql.server startup script).

- -l

  Do not subtract lock time from total time.

- -n *N*

  Abstract numbers with at least *N* digits within names.

- -r

  Reverse the sort order.

- -s *sort_type*

  How to sort the output. The value of *sort_type* should be chosen from the following list:

  - t, at: Sort by query time or average query time
  - l, al: Sort by lock time or average lock time
  - r, ar: Sort by rows sent or average rows sent
  - c: Sort by count

  By default, mysqldumpslow sorts by average query time (equivalent to -s at).

- -t *N*

  Display only the first *N* queries in the output.

- --verbose, -v

  Verbose mode. Print more information about what the program does.

Example of usage:

```
shell> mysqldumpslow

Reading mysql slow query log from /usr/local/mysql/data/mysqld51-apple-slow.log
Count: 1  Time=4.32s (4s)  Lock=0.00s (0s)  Rows=0.0 (0), root[root]@localhost
 insert into t2 select * from t1

Count: 3  Time=2.53s (7s)  Lock=0.00s (0s)  Rows=0.0 (0), root[root]@localhost
 insert into t2 select * from t1 limit N

Count: 3  Time=2.13s (6s)  Lock=0.00s (0s)  Rows=0.0 (0), root[root]@localhost
 insert into t1 select * from t1
```

4.6.9 mysqlhotcopy — A Database Backup Program

mysqlhotcopy is a Perl script that was originally written and contributed by Tim Bunce. It uses FLUSH TABLES, LOCK TABLES, and cp or scp to make a database backup. It is a fast way to make a backup of the database or single tables, but it can be run only on the same machine where the database directories are located. mysqlhotcopy works only for backing up MyISAM and ARCHIVE tables. It runs on Unix.

To use mysqlhotcopy, you must have read access to the files for the tables that you are backing up, the SELECT privilege for those tables, the RELOAD privilege (to be able to execute FLUSH TABLES), and the LOCK TABLES privilege (to be able to lock the tables).

```
shell> mysqlhotcopy db_name [/path/to/new_directory]
```

```
shell> mysqlhotcopy db_name_1 ... db_name_n /path/to/new_directory
```

Back up tables in the given database that match a regular expression:

```
shell> mysqlhotcopy db_name./regex/
```

The regular expression for the table name can be negated by prefixing it with a tilde (“~”):

```
shell> mysqlhotcopy db_name./~regex/
```

mysqlhotcopy supports the following options, which can be specified on the command line or in the [mysqlhotcopy] and [client] groups of an option file. For information about option files, see Section 4.2.3.3, “Using Option Files”.

Table 4.17 mysqlhotcopy Options

Format	Option File	Description
--addtodest	addtodest	Do not rename target directory (if it exists); merely add files to it
--allowold	allowold	Do not abort if a target exists; rename it by adding an _old suffix
--checkpoint=db_name.tbl_name	checkpoint	Insert checkpoint entries
--chroot=path	chroot	Base directory of the chroot jail in which mysqld operates
--debug	debug	Write a debugging log
--dryrun	dryrun	Report actions without performing them

Format	Option File	Description
--flushlog	flushlog	Flush logs after all tables are locked
--help		Display help message and exit
--host=host_name	host	Connect to the MySQL server on the given host
--keepold	keepold	Do not delete previous (renamed) target when done
--method	method	The method for copying files
--noindices	noindices	Do not include full index files in the backup
--old_server	old_server	Connect to server that does not support FLUSH TABLES tbl_list WITH READ LOCK
--password[=password]	password	The password to use when connecting to the server
--port=port_num	port	The TCP/IP port number to use for the connection
--quiet	quiet	Be silent except for errors
--regexp	regexp	Copy all databases with names that match the given regular expression
--resetmaster	resetmaster	Reset the binary log after locking all the tables
--resetslave	resetslave	Reset the master.info file after locking all the tables
--socket=path	socket	For connections to localhost
--tmpdir=path	tmpdir	The temporary directory
--user=user_name,	user	MySQL user name to use when connecting to server

- `--help`, `-?`

  Display a help message and exit.

- `--addtodest`

  Do not rename target directory (if it exists); merely add files to it.

- `--allowold`

  Do not abort if a target exists; rename it by adding an `_old` suffix.

- `--checkpoint=`*`db_name.tbl_name`*

  Insert checkpoint entries into the specified database *`db_name`* and table *`tbl_name`*.

- `--chroot=`*`path`*

  Base directory of the `chroot` jail in which `mysqld` operates. The *`path`* value should match that of the `--chroot` option given to `mysqld`.

- `--debug`

  Enable debug output.

- `--dryrun`, `-n`

  Report actions without performing them.

- `--flushlog`

Flush logs after all tables are locked.

- `--host=host_name`, `-h host_name`

  The host name of the local host to use for making a TCP/IP connection to the local server. By default, the connection is made to `localhost` using a Unix socket file.

- `--keepold`

  Do not delete previous (renamed) target when done.

- `--method=command`

  The method for copying files (`cp` or `scp`). The default is `cp`.

- `--noindices`

  Do not include full index files for `MyISAM` tables in the backup. This makes the backup smaller and faster. The indexes for reloaded tables can be reconstructed later with `myisamchk -rq`.

- `--password=password`, `-ppassword`

  The password to use when connecting to the server. The password value is not optional for this option, unlike for other MySQL programs.

  Specifying a password on the command line should be considered insecure. See Section 6.1.2.1, “End-User Guidelines for Password Security”. You can use an option file to avoid giving the password on the command line.

- `--port=port_num`, `-P port_num`

  The TCP/IP port number to use when connecting to the local server.

- `--old_server`

  In MySQL 5.7, `mysqlhotcopy` uses `FLUSH TABLES tbl_list WITH READ LOCK` to flush and lock tables. Use the `--old_server` option if the server is older than 5.5.3, which is when that statement was introduced.

- `--quiet`, `-q`

  Be silent except for errors.

- `--record_log_pos=db_name.tbl_name`

  Record master and slave status in the specified database *`db_name`* and table *`tbl_name`*.

- `--regexp=expr`

  Copy all databases with names that match the given regular expression.

- `--resetmaster`

  Reset the binary log after locking all the tables.

- `--resetslave`

  Reset the master info repository file or table after locking all the tables.

- --socket=*path*, -S *path*

  The Unix socket file to use for connections to localhost.

- --suffix=*str*

  The suffix to use for names of copied databases.

- --tmpdir=*path*

  The temporary directory. The default is /tmp.

- --user=*user_name*, -u *user_name*

  The MySQL user name to use when connecting to the server.

Use perldoc for additional mysqlhotcopy documentation, including information about the structure of the tables needed for the --checkpoint and --record_log_pos options:

```
shell> perldoc mysqlhotcopy
```

4.6.10 mysql_waitpid — Kill Process and Wait for Its Termination

mysql_waitpid signals a process to terminate and waits for the process to exit. It uses the kill() system call and Unix signals, so it runs on Unix and Unix-like systems.

Invoke mysql_waitpid like this:

```
shell> mysql_waitpid [options] pid wait_time
```

mysql_waitpid sends signal 0 to the process identified by *pid* and waits up to *wait_time* seconds for the process to terminate. *pid* and *wait_time* must be positive integers.

If process termination occurs within the wait time or the process does not exist, mysql_waitpid returns 0. Otherwise, it returns 1.

If the kill() system call cannot handle signal 0, mysql_waitpid() uses signal 1 instead.

mysql_waitpid supports the following options:

- --help, -?, -I

  Display a help message and exit.

- --verbose, -v

  Verbose mode. Display a warning if signal 0 could not be used and signal 1 is used instead.

- --version, -V

  Display version information and exit.

4.6.11 mysql_zap — Kill Processes That Match a Pattern

mysql_zap kills processes that match a pattern. It uses the ps command and Unix signals, so it runs on Unix and Unix-like systems.

Invoke mysql_zap like this:

```
shell> mysql_zap [-signal] [-?Ift] pattern
```

A process matches if its output line from the ps command contains the pattern. By default, mysql_zap asks for confirmation for each process. Respond y to kill the process, or q to exit mysql_zap. For any other response, mysql_zap does not attempt to kill the process.

If the -*signal* option is given, it specifies the name or number of the signal to send to each process. Otherwise, mysql_zap tries first with TERM (signal 15) and then with KILL (signal 9).

mysql_zap supports the following additional options:

- --help, -?, -I

  Display a help message and exit.

- -f

  Force mode. mysql_zap attempts to kill each process without confirmation.

- -t

  Test mode. Display information about each process but do not kill it.

4.7 MySQL Program Development Utilities

This section describes some utilities that you may find useful when developing MySQL programs.

In shell scripts, you can use the my_print_defaults program to parse option files and see what options would be used by a given program. The following example shows the output that my_print_defaults might produce when asked to show the options found in the [client] and [mysql] groups:

```
shell> my_print_defaults client mysql
--port=3306
--socket=/tmp/mysql.sock
--no-auto-rehash
```

Note for developers: Option file handling is implemented in the C client library simply by processing all options in the appropriate group or groups before any command-line arguments. This works well for programs that use the last instance of an option that is specified multiple times. If you have a C or C++ program that handles multiply specified options this way but that doesn't read option files, you need add only two lines to give it that capability. Check the source code of any of the standard MySQL clients to see how to do this.

Several other language interfaces to MySQL are based on the C client library, and some of them provide a way to access option file contents. These include Perl and Python. For details, see the documentation for your preferred interface.

4.7.1 mysql_config — Display Options for Compiling Clients

mysql_config provides you with useful information for compiling your MySQL client and connecting it to MySQL. It is a shell script, so it is available only on Unix and Unix-like systems.

Note

As of MySQL 5.7.4, for binary distributions for Solaris, mysql_config does not provide arguments for linking with the embedded library. To get linking arguments for the embedded library, use the mysql_server_config script instead.

mysql_config supports the following options.

- --cflags

  C Compiler flags to find include files and critical compiler flags and defines used when compiling the libmysqlclient library. The options returned are tied to the specific compiler that was used when the library was created and might clash with the settings for your own compiler. Use --include for more portable options that contain only include paths.

- --cxxflags

  Like --cflags, but for C++ compiler flags.

- --include

  Compiler options to find MySQL include files.

- --libmysqld-libs, --embedded

  Libraries and options required to link with the MySQL embedded server.

- --libs

  Libraries and options required to link with the MySQL client library.

- --libs_r

  Libraries and options required to link with the thread-safe MySQL client library. In MySQL 5.7, all client libraries are thread-safe, so this option need not be used. The --libs option can be used in all cases.

- --plugindir

  The default plugin directory path name, defined when configuring MySQL.

- --port

  The default TCP/IP port number, defined when configuring MySQL.

- --socket

  The default Unix socket file, defined when configuring MySQL.

- --version

  Version number for the MySQL distribution.

If you invoke mysql_config with no options, it displays a list of all options that it supports, and their values:

```
shell> mysql_config
Usage: /usr/local/mysql/bin/mysql_config [options]
Options:
  --cflags         [-I/usr/local/mysql/include/mysql -mcpu=pentiumpro]
  --include        [-I/usr/local/mysql/include/mysql]
  --libs           [-L/usr/local/mysql/lib/mysql -lmysqlclient
                    -lpthread -lm -lrt -lssl -lcrypto -ldl]
  --libs_r         [-L/usr/local/mysql/lib/mysql -lmysqlclient_r
                    -lpthread -lm -lrt -lssl -lcrypto -ldl]
  --socket         [/tmp/mysql.sock]
  --port            [3306]
  --version         [5.7.1]
  --libmysqld-libs [-L/usr/local/mysql/lib/mysql -lmysqld
                    -lpthread -lm -lrt -lssl -lcrypto -ldl -lcrypt]
```

You can use mysql_config within a command line using backticks to include the output that it produces for a particular option. For example, to compile and link a MySQL client program, use mysql_config as follows:

```
shell> gcc -c `mysql_config --cflags` progname.c
shell> gcc -o progname progname.o `mysql_config --libs`
```

4.7.2 my_print_defaults — Display Options from Option Files

my_print_defaults displays the options that are present in option groups of option files. The output indicates what options will be used by programs that read the specified option groups. For example, the mysqlcheck program reads the [mysqlcheck] and [client] option groups. To see what options are present in those groups in the standard option files, invoke my_print_defaults like this:

```
shell> my_print_defaults mysqlcheck client
--user=myusername
--password=secret
--host=localhost
```

The output consists of options, one per line, in the form that they would be specified on the command line.

my_print_defaults supports the following options.

- --help, -?

  Display a help message and exit.

- --config-file=*file_name*, --defaults-file=*file_name*, -c *file_name*

  Read only the given option file.

- --debug=*debug_options*, -# *debug_options*

  Write a debugging log. A typical *debug_options* string is d:t:o,*file_name*. The default is d:t:o,/tmp/my_print_defaults.trace.

- --defaults-extra-file=*file_name*, --extra-file=*file_name*, -e *file_name*

  Read this option file after the global option file but (on Unix) before the user option file.

- --defaults-group-suffix=*suffix*, -g *suffix*

  In addition to the groups named on the command line, read groups that have the given suffix.

- --login-path=*name*, -l *name*

  Read options from the named login path in the .mylogin.cnf login file. A “login path” is an option group that permits only a limited set of options: host, user, and password. Think of a login path as a set of values that indicate the server host and the credentials for authenticating with the server. To create the login file, use the mysql_config_editor utility. See Section 4.6.6, “mysql_config_editor — MySQL Configuration Utility”.

- --no-defaults, -n

Return an empty string.

- --verbose, -v

  Verbose mode. Print more information about what the program does.

- --version, -V

  Display version information and exit.

4.7.3 resolve_stack_dump — Resolve Numeric Stack Trace Dump to Symbols

resolve_stack_dump resolves a numeric stack dump to symbols.

Invoke resolve_stack_dump like this:

```
shell> resolve_stack_dump [options] symbols_file [numeric_dump_file]
```

The symbols file should include the output from the nm --numeric-sort mysqld command. The numeric dump file should contain a numeric stack track from mysqld. If no numeric dump file is named on the command line, the stack trace is read from the standard input.

resolve_stack_dump supports the following options.

- --help, -h

  Display a help message and exit.

- --numeric-dump-file=*file_name*, -n *file_name*

  Read the stack trace from the given file.

- --symbols-file=*file_name*, -s *file_name*

  Use the given symbols file.

- --version, -V

  Display version information and exit.

For more information, see Section 22.4.1.5, “Using a Stack Trace”.

4.8 Miscellaneous Programs

4.8.1 perror — Explain Error Codes

For most system errors, MySQL displays, in addition to an internal text message, the system error code in one of the following styles:

```
message ... (errno: #)
message ... (Errcode: #)
```

You can find out what the error code means by examining the documentation for your system or by using the perror utility.

perror prints a description for a system error code or for a storage engine (table handler) error code.

Invoke perror like this:

```
shell> perror [options] errorcode ...
```

Example:

```
shell> perror 13 64
OS error code  13:  Permission denied
OS error code  64:  Machine is not on the network
```

To obtain the error message for a MySQL Cluster error code, invoke perror with the --ndb option:

```
shell> perror --ndb errorcode
```

Note that the meaning of system error messages may be dependent on your operating system. A given error code may mean different things on different operating systems.

perror supports the following options.

- --help, --info, -I, -?

  Display a help message and exit.

- --ndb

  Print the error message for a MySQL Cluster error code.

- --silent, -s

  Silent mode. Print only the error message.

- --verbose, -v

  Verbose mode. Print error code and message. This is the default behavior.

- --version, -V

  Display version information and exit.

4.8.2 replace — A String-Replacement Utility

The replace utility program changes strings in place in files or on the standard input.

Invoke replace in one of the following ways:

```
shell> replace from to [from to] ... -- file_name [file_name] ...
shell> replace from to [from to] ... < file_name
```

from represents a string to look for and *to* represents its replacement. There can be one or more pairs of strings.

Use the -- option to indicate where the string-replacement list ends and the file names begin. In this case, any file named on the command line is modified in place, so you may want to make a copy of the original before converting it. *replace* prints a message indicating which of the input files it actually modifies.

If the `--` option is not given, `replace` reads the standard input and writes to the standard output.

`replace` uses a finite state machine to match longer strings first. It can be used to swap strings. For example, the following command swaps `a` and `b` in the given files, `file1` and `file2`:

```
shell> replace a b b a -- file1 file2 ...
```

`replace` supports the following options.

- `-?`, `-I`

  Display a help message and exit.

- `-#debug_options`

  Enable debugging.

- `-s`

  Silent mode. Print less information what the program does.

- `-v`

  Verbose mode. Print more information about what the program does.

- `-V`

  Display version information and exit.

4.8.3 `resolveip` — Resolve Host name to IP Address or Vice Versa

The `resolveip` utility resolves host names to IP addresses and vice versa.

Invoke `resolveip` like this:

```
shell> resolveip [options] {host_name|ip-addr} ...
```

`resolveip` supports the following options.

- `--help`, `--info`, `-?`, `-I`

  Display a help message and exit.

- `--silent`, `-s`

  Silent mode. Produce less output.

- `--version`, `-V`

  Display version information and exit.

Chapter 5 MySQL Server Administration

Table of Contents

MySQL Server (`mysqld`) is the main program that does most of the work in a MySQL installation. This chapter provides an overview of MySQL Server and covers general server administration:

- Server configuration.
- The server log files.
- Management of multiple servers on a single machine.

For additional information on administrative topics, see also:

- Chapter 6, *Security*
- Chapter 7, *Backup and Recovery*
- Chapter 16, *Replication*

5.1 The MySQL Server

`mysqld` is the MySQL server. The following discussion covers these MySQL server configuration topics:

- Startup options that the server supports. You can specify these options on the command line, through configuration files, or both.

- Server system variables. These variables reflect the current state and values of the startup options, some of which can be modified while the server is running.
- Server status variables. These variables contain counters and statistics about runtime operation.
- How to set the server SQL mode. This setting modifies certain aspects of SQL syntax and semantics, for example for compatibility with code from other database systems, or to control the error handling for particular situations.
- The server shutdown process. There are performance and reliability considerations depending on the type of table (transactional or nontransactional) and whether you use replication.

Note

Not all storage engines are supported by all MySQL server binaries and configurations. To find out how to determine which storage engines your MySQL server installation supports, see Section 13.7.5.15, "`SHOW ENGINES` Syntax".

5.1.1 Server Option and Variable Reference

The following table provides a list of all the command line options, server and status variables applicable within `mysqld`.

The table lists command-line options (Cmd-line), options valid in configuration files (Option file), server system variables (System Var), and status variables (Status var) in one unified list, with notification of where each option/variable is valid. If a server option set on the command line or in an option file differs from the name of the corresponding server system or status variable, the variable name is noted immediately below the corresponding option. For status variables, the scope of the variable is shown (Scope) as either global, session, or both. Please see the corresponding sections for details on setting and using the options and variables. Where appropriate, a direct link to further information on the item as available.

Table 5.1 Option/Variable Summary

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
abort-slave-event-count	Yes	Yes				
Aborted_clients				Yes	Global	No
Aborted_connects				Yes	Global	No
allow-suspicious-udfs	Yes	Yes				
ansi	Yes	Yes				
audit_log_format			Yes		Global	No
auto_increment_increment			Yes		Both	Yes
auto_increment_offset			Yes		Both	Yes
autocommit	Yes	Yes	Yes		Both	Yes
automatic_sp_privileges			Yes		Global	Yes
back_log			Yes		Global	No
basedir	Yes	Yes	Yes		Global	No
big-tables	Yes	Yes			Both	Yes

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
- *Variable*: big_tables			Yes		Both	Yes
bind-address	Yes	Yes			Global	No
- *Variable*: bind_address			Yes		Global	No
Binlog_cache_disk_use				Yes	Global	No
binlog_cache_size	Yes	Yes	Yes		Global	Yes
Binlog_cache_use				Yes	Global	No
binlog-checksum	Yes	Yes				
binlog_checksum			Yes		Global	Yes
binlog_direct_non_transactional_updates	Yes	Yes	Yes		Both	Yes
binlog-do-db	Yes	Yes				
binlog-format	Yes	Yes			Both	Yes
- *Variable*: binlog_format			Yes		Both	Yes
binlog-ignore-db	Yes	Yes				
binlog_max_flush_queue_time			Yes		Global	Yes
binlog_order_commits			Yes		Global	Yes
binlog-row-event-max-size	Yes	Yes				
binlog_row_image	Yes	Yes	Yes		Both	Yes
binlog_rows_query_log_events			Yes		Both	Yes
binlog-rows-query-log-events	Yes	Yes				
- *Variable*: binlog_rows_query_log_events						
Binlog_stmt_cache_disk_use				Yes	Global	No
binlog_stmt_cache_size	Yes	Yes	Yes		Global	Yes
Binlog_stmt_cache_use				Yes	Global	No
block_encryption_mode	Yes	Yes	Yes		Both	Yes
bootstrap	Yes	Yes				
bulk_insert_buffer_size	Yes	Yes	Yes		Both	Yes
Bytes_received				Yes	Both	No
Bytes_sent				Yes	Both	No
character_set_client			Yes		Both	Yes
character-set-client-handshake	Yes	Yes				
character_set_connection			Yes		Both	Yes
character_set_database[a]			Yes		Both	Yes

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
character-set-filesystem	Yes	Yes			Both	Yes
- *Variable*: character_set_filesystem			Yes		Both	Yes
character_set_results			Yes		Both	Yes
character-set-server	Yes	Yes			Both	Yes
- *Variable*: character_set_server			Yes		Both	Yes
character_set_system			Yes		Global	No
character-sets-dir	Yes	Yes			Global	No
- *Variable*: character_sets_dir			Yes		Global	No
chroot	Yes	Yes				
collation_connection			Yes		Both	Yes
collation_database[b]			Yes		Both	Yes
collation-server	Yes	Yes			Both	Yes
- *Variable*: collation_server			Yes		Both	Yes
Com_admin_commands				Yes	Both	No
Com_alter_db				Yes	Both	No
Com_alter_db_upgrade				Yes	Both	No
Com_alter_event				Yes	Both	No
Com_alter_function				Yes	Both	No
Com_alter_procedure				Yes	Both	No
Com_alter_server				Yes	Both	No
Com_alter_table				Yes	Both	No
Com_alter_tablespace				Yes	Both	No
Com_alter_user				Yes	Both	No
Com_analyze				Yes	Both	No
Com_assign_to_keycache				Yes	Both	No
Com_begin				Yes	Both	No
Com_binlog				Yes	Both	No
Com_call_procedure				Yes	Both	No
Com_change_db				Yes	Both	No
Com_change_master				Yes	Both	No
Com_change_repl_filter				Yes	Both	No
Com_check				Yes	Both	No
Com_checksum				Yes	Both	No
Com_commit				Yes	Both	No

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
Com_create_db				Yes	Both	No
Com_create_event				Yes	Both	No
Com_create_function				Yes	Both	No
Com_create_index				Yes	Both	No
Com_create_procedure				Yes	Both	No
Com_create_server				Yes	Both	No
Com_create_table				Yes	Both	No
Com_create_trigger				Yes	Both	No
Com_create_udf				Yes	Both	No
Com_create_user				Yes	Both	No
Com_create_view				Yes	Both	No
Com_dealloc_sql				Yes	Both	No
Com_delete				Yes	Both	No
Com_delete_multi				Yes	Both	No
Com_do				Yes	Both	No
Com_drop_db				Yes	Both	No
Com_drop_event				Yes	Both	No
Com_drop_function				Yes	Both	No
Com_drop_index				Yes	Both	No
Com_drop_procedure				Yes	Both	No
Com_drop_server				Yes	Both	No
Com_drop_table				Yes	Both	No
Com_drop_trigger				Yes	Both	No
Com_drop_user				Yes	Both	No
Com_drop_view				Yes	Both	No
Com_empty_query				Yes	Both	No
Com_execute_sql				Yes	Both	No
Com_flush				Yes	Both	No
Com_get_diagnostics				Yes	Both	No
Com_grant				Yes	Both	No
Com_ha_close				Yes	Both	No
Com_ha_open				Yes	Both	No
Com_ha_read				Yes	Both	No
Com_help				Yes	Both	No
Com_insert				Yes	Both	No
Com_insert_select				Yes	Both	No
Com_install_plugin				Yes	Both	No
Com_kill				Yes	Both	No

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
Com_load				Yes	Both	No
Com_lock_tables				Yes	Both	No
Com_optimize				Yes	Both	No
Com_preload_keys				Yes	Both	No
Com_prepare_sql				Yes	Both	No
Com_purge				Yes	Both	No
Com_purge_before_date				Yes	Both	No
Com_release_savepoint				Yes	Both	No
Com_rename_table				Yes	Both	No
Com_rename_user				Yes	Both	No
Com_repair				Yes	Both	No
Com_replace				Yes	Both	No
Com_replace_select				Yes	Both	No
Com_reset				Yes	Both	No
Com_resignal				Yes	Both	No
Com_revoke				Yes	Both	No
Com_revoke_all				Yes	Both	No
Com_rollback				Yes	Both	No
Com_rollback_to_savepoint				Yes	Both	No
Com_savepoint				Yes	Both	No
Com_select				Yes	Both	No
Com_set_option				Yes	Both	No
Com_show_authors				Yes	Both	No
Com_show_binlog_events				Yes	Both	No
Com_show_binlogs				Yes	Both	No
Com_show_charsets				Yes	Both	No
Com_show_collations				Yes	Both	No
Com_show_contributors				Yes	Both	No
Com_show_create_db				Yes	Both	No
Com_show_create_event				Yes	Both	No
Com_show_create_func				Yes	Both	No
Com_show_create_proc				Yes	Both	No
Com_show_create_table				Yes	Both	No
Com_show_create_trigger				Yes	Both	No
Com_show_databases				Yes	Both	No
Com_show_engine_logs				Yes	Both	No
Com_show_engine_mutex				Yes	Both	No
Com_show_engine_status				Yes	Both	No

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
Com_show_errors				Yes	Both	No
Com_show_events				Yes	Both	No
Com_show_fields				Yes	Both	No
Com_show_function_code				Yes	Both	No
Com_show_function_status				Yes	Both	No
Com_show_grants				Yes	Both	No
Com_show_keys				Yes	Both	No
Com_show_master_status				Yes	Both	No
Com_show_new_master				Yes	Both	No
Com_show_open_tables				Yes	Both	No
Com_show_plugins				Yes	Both	No
Com_show_privileges				Yes	Both	No
Com_show_procedure_code				Yes	Both	No
Com_show_procedure_status				Yes	Both	No
Com_show_processlist				Yes	Both	No
Com_show_profile				Yes	Both	No
Com_show_profiles				Yes	Both	No
Com_show_relaylog_events				Yes	Both	No
Com_show_slave_hosts				Yes	Both	No
Com_show_slave_status				Yes	Both	No
Com_show_status				Yes	Both	No
Com_show_storage_engines				Yes	Both	No
Com_show_table_status				Yes	Both	No
Com_show_tables				Yes	Both	No
Com_show_triggers				Yes	Both	No
Com_show_variables				Yes	Both	No
Com_show_warnings				Yes	Both	No
Com_signal				Yes	Both	No
Com_slave_start				Yes	Both	No
Com_slave_stop				Yes	Both	No
Com_stmt_close				Yes	Both	No
Com_stmt_execute				Yes	Both	No
Com_stmt_fetch				Yes	Both	No
Com_stmt_prepare				Yes	Both	No
Com_stmt_reprepare				Yes	Both	No
Com_stmt_reset				Yes	Both	No
Com_stmt_send_long_data				Yes	Both	No
Com_truncate				Yes	Both	No

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
Com_uninstall_plugin				Yes	Both	No
Com_unlock_tables				Yes	Both	No
Com_update				Yes	Both	No
Com_update_multi				Yes	Both	No
Com_xa_commit				Yes	Both	No
Com_xa_end				Yes	Both	No
Com_xa_prepare				Yes	Both	No
Com_xa_recover				Yes	Both	No
Com_xa_rollback				Yes	Both	No
Com_xa_start				Yes	Both	No
completion_type	Yes	Yes	Yes		Both	Yes
Compression				Yes	Session	No
concurrent_insert	Yes	Yes	Yes		Global	Yes
connect_timeout	Yes	Yes	Yes		Global	Yes
Connection_errors_accept				Yes	Global	No
Connection_errors_internal				Yes	Global	No
Connection_errors_max_connections				Yes	Global	No
Connection_errors_peer_addr				Yes	Global	No
Connection_errors_select				Yes	Global	No
Connection_errors_tcpwrap				Yes	Global	No
Connections				Yes	Global	No
console	Yes	Yes				
core_file			Yes		Global	No
core-file	Yes	Yes				
Created_tmp_disk_tables				Yes	Both	No
Created_tmp_files				Yes	Global	No
Created_tmp_tables				Yes	Both	No
daemon_memcached_enable_binlog	Yes	Yes	Yes		Global	No
daemon_memcached_engine_lib_name	Yes	Yes	Yes		Global	No
daemon_memcached_engine_lib_path	Yes	Yes	Yes		Global	No
daemon_memcached_option	Yes	Yes	Yes		Global	No
daemon_memcached_r_batch_size	Yes	Yes	Yes		Global	No
daemon_memcached_w_batch_size	Yes	Yes	Yes		Global	No
datadir	Yes	Yes	Yes		Global	No
date_format			Yes		Global	No
datetime_format			Yes		Global	No
debug	Yes	Yes	Yes		Both	Yes
debug_sync			Yes		Session	Yes

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
debug-sync-timeout	Yes	Yes				
default-authentication-plugin	Yes	Yes				
default_authentication_plugin	Yes	Yes	Yes		Global	No
default_password_lifetime	Yes	Yes	Yes		Global	Yes
default-storage-engine	Yes	Yes			Both	Yes
- *Variable*: default_storage_engine			Yes		Both	Yes
default-time-zone	Yes	Yes				
default_tmp_storage_engine	Yes	Yes	Yes		Both	Yes
default_week_format	Yes	Yes	Yes		Both	Yes
defaults-extra-file	Yes					
defaults-file	Yes					
defaults-group-suffix	Yes					
delay-key-write	Yes	Yes			Global	Yes
- *Variable*: delay_key_write			Yes		Global	Yes
Delayed_errors				Yes	Global	No
delayed_insert_limit	Yes	Yes	Yes		Global	Yes
Delayed_insert_threads				Yes	Global	No
delayed_insert_timeout	Yes	Yes	Yes		Global	Yes
delayed_queue_size	Yes	Yes	Yes		Global	Yes
Delayed_writes				Yes	Global	No
des-key-file	Yes	Yes				
disconnect_on_expired_password	Yes	Yes	Yes		Session	No
disconnect-slave-event-count	Yes	Yes				
div_precision_increment	Yes	Yes	Yes		Both	Yes
enable-named-pipe	Yes	Yes				
- *Variable*: named_pipe						
end_markers_in_json			Yes		Both	Yes
enforce_gtid_consistency	Yes	Yes	Yes		Global	No
enforce-gtid-consistency	Yes	Yes	Yes		Global	No
eq_range_index_dive_limit			Yes		Both	Yes

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
error_count			Yes		Session	No
event-scheduler	Yes	Yes			Global	Yes
- *Variable*: event_scheduler			Yes		Global	Yes
exit-info	Yes	Yes				
expire_logs_days	Yes	Yes	Yes		Global	Yes
explicit_defaults_for_timestamp	Yes	Yes	Yes		Session	No
external-locking	Yes	Yes				
- *Variable*: skip_external_locking						
external_user			Yes		Session	No
federated	Yes	Yes				
flush	Yes	Yes	Yes		Global	Yes
Flush_commands				Yes	Global	No
flush_time	Yes	Yes	Yes		Global	Yes
foreign_key_checks			Yes		Both	Yes
ft_boolean_syntax	Yes	Yes	Yes		Global	Yes
ft_max_word_len	Yes	Yes	Yes		Global	No
ft_min_word_len	Yes	Yes	Yes		Global	No
ft_query_expansion_limit	Yes	Yes	Yes		Global	No
ft_stopword_file	Yes	Yes	Yes		Global	No
gdb	Yes	Yes				
general-log	Yes	Yes			Global	Yes
- *Variable*: general_log			Yes		Global	Yes
general_log_file	Yes	Yes	Yes		Global	Yes
group_concat_max_len	Yes	Yes	Yes		Both	Yes
gtid_executed			Yes		Both	No
gtid_mode			Yes		Global	No
gtid-mode	Yes	Yes			Global	No
- *Variable*: gtid_mode			Yes		Global	No
gtid_next			Yes		Session	Yes
gtid_owned			Yes		Both	No
gtid_purged			Yes		Global	Yes
Handler_commit				Yes	Both	No
Handler_delete				Yes	Both	No
Handler_discover				Yes	Both	No
Handler_external_lock				Yes	Both	No

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
Handler_mrr_init				Yes	Both	No
Handler_prepare				Yes	Both	No
Handler_read_first				Yes	Both	No
Handler_read_key				Yes	Both	No
Handler_read_last				Yes	Both	No
Handler_read_next				Yes	Both	No
Handler_read_prev				Yes	Both	No
Handler_read_rnd				Yes	Both	No
Handler_read_rnd_next				Yes	Both	No
Handler_rollback				Yes	Both	No
Handler_savepoint				Yes	Both	No
Handler_savepoint_rollback				Yes	Both	No
Handler_update				Yes	Both	No
Handler_write				Yes	Both	No
have_compress			Yes		Global	No
have_crypt			Yes		Global	No
have_dynamic_loading			Yes		Global	No
have_geometry			Yes		Global	No
have_openssl			Yes		Global	No
have_profiling			Yes		Global	No
have_query_cache			Yes		Global	No
have_rtree_keys			Yes		Global	No
have_ssl			Yes		Global	No
have_symlink			Yes		Global	No
help	Yes	Yes				
host_cache_size			Yes		Global	Yes
hostname			Yes		Global	No
identity			Yes		Session	Yes
ignore-builtin-innodb	Yes	Yes			Global	No
- *Variable*: ignore_builtin_innodb			Yes		Global	No
ignore-db-dir	Yes	Yes				
ignore_db_dirs			Yes		Global	No
init_connect	Yes	Yes	Yes		Global	Yes
init-file	Yes	Yes			Global	No
- *Variable*: init_file			Yes		Global	No
init_slave	Yes	Yes	Yes		Global	Yes

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
innodb	Yes	Yes				
innodb_adaptive_flushing	Yes	Yes	Yes		Global	Yes
innodb_adaptive_flushing_lwm	Yes	Yes	Yes		Global	Yes
innodb_adaptive_hash_index	Yes	Yes	Yes		Global	Yes
innodb_adaptive_max_sleep_delay	Yes	Yes	Yes		Global	Yes
innodb_additional_mem_pool_size	Yes	Yes	Yes		Global	No
innodb_api_bk_commit_interval	Yes	Yes	Yes		Global	Yes
innodb_api_disable_rowlock	Yes	Yes	Yes		Global	No
innodb_api_enable_binlog	Yes	Yes	Yes		Global	No
innodb_api_enable_mdl	Yes	Yes	Yes		Global	No
innodb_api_trx_level	Yes	Yes	Yes		Global	Yes
innodb_autoextend_increment	Yes	Yes	Yes		Global	Yes
innodb_autoinc_lock_mode	Yes	Yes	Yes		Global	No
Innodb_available_undo_logs				Yes	Global	No
Innodb_buffer_pool_bytes_data				Yes	Global	No
Innodb_buffer_pool_bytes_dirty				Yes	Global	No
innodb_buffer_pool_dump_at_shutdown	Yes	Yes	Yes		Global	Yes
innodb_buffer_pool_dump_now	Yes	Yes	Yes		Global	Yes
innodb_buffer_pool_dump_pct	Yes	Yes	Yes		Global	Yes
Innodb_buffer_pool_dump_status				Yes	Global	No
innodb_buffer_pool_filename	Yes	Yes	Yes		Global	Yes
innodb_buffer_pool_instances	Yes	Yes	Yes		Global	No
innodb_buffer_pool_load_abort	Yes	Yes	Yes		Global	Yes
innodb_buffer_pool_load_at_startup	Yes	Yes	Yes		Global	No
innodb_buffer_pool_load_now	Yes	Yes	Yes		Global	Yes
Innodb_buffer_pool_load_status				Yes	Global	No
Innodb_buffer_pool_pages_data				Yes	Global	No
Innodb_buffer_pool_pages_dirty				Yes	Global	No
Innodb_buffer_pool_pages_flushed				Yes	Global	No
Innodb_buffer_pool_pages_free				Yes	Global	No
Innodb_buffer_pool_pages_latched				Yes	Global	No
Innodb_buffer_pool_pages_misc				Yes	Global	No
Innodb_buffer_pool_pages_total				Yes	Global	No
Innodb_buffer_pool_read_ahead				Yes	Global	No
Innodb_buffer_pool_read_ahead_evicted				Yes	Global	No
Innodb_buffer_pool_read_requests				Yes	Global	No
Innodb_buffer_pool_reads				Yes	Global	No
innodb_buffer_pool_size	Yes	Yes	Yes		Global	No

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
Innodb_buffer_pool_wait_free				Yes	Global	No
Innodb_buffer_pool_write_requests				Yes	Global	No
innodb_change_buffer_max_size	Yes	Yes	Yes		Global	Yes
innodb_change_buffering	Yes	Yes	Yes		Global	Yes
innodb_checksum_algorithm	Yes	Yes	Yes		Global	Yes
innodb_checksums	Yes	Yes	Yes		Global	No
innodb_cmp_per_index_enabled	Yes	Yes	Yes		Global	Yes
innodb_commit_concurrency	Yes	Yes	Yes		Global	Yes
innodb_compression_failure_threshold_pct	Yes	Yes	Yes		Global	Yes
innodb_compression_level	Yes	Yes	Yes		Global	Yes
innodb_compression_pad_pct_max	Yes	Yes	Yes		Global	Yes
innodb_concurrency_tickets	Yes	Yes	Yes		Global	Yes
innodb_data_file_path	Yes	Yes	Yes		Global	No
Innodb_data_fsyncs				Yes	Global	No
innodb_data_home_dir	Yes	Yes	Yes		Global	No
Innodb_data_pending_fsyncs				Yes	Global	No
Innodb_data_pending_reads				Yes	Global	No
Innodb_data_pending_writes				Yes	Global	No
Innodb_data_read				Yes	Global	No
Innodb_data_reads				Yes	Global	No
Innodb_data_writes				Yes	Global	No
Innodb_data_written				Yes	Global	No
Innodb_dblwr_pages_written				Yes	Global	No
Innodb_dblwr_writes				Yes	Global	No
innodb_disable_sort_file_cache	Yes	Yes	Yes		Global	Yes
innodb_doublewrite	Yes	Yes	Yes		Global	No
innodb_fast_shutdown	Yes	Yes	Yes		Global	Yes
innodb_file_format	Yes	Yes	Yes		Global	Yes
innodb_file_format_check	Yes	Yes	Yes		Global	No
innodb_file_format_max	Yes	Yes	Yes		Global	Yes
innodb_file_per_table	Yes	Yes	Yes		Global	Yes
innodb_flush_log_at_timeout			Yes		Global	Yes
innodb_flush_log_at_trx_commit	Yes	Yes	Yes		Global	Yes
innodb_flush_method	Yes	Yes	Yes		Global	No
innodb_flush_neighbors	Yes	Yes	Yes		Global	Yes
innodb_flushing_avg_loops	Yes	Yes	Yes		Global	Yes
innodb_force_load_corrupted	Yes	Yes	Yes		Global	No
innodb_force_recovery	Yes	Yes	Yes		Global	No

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
innodb_ft_aux_table	Yes	Yes	Yes		Global	Yes
innodb_ft_cache_size	Yes	Yes	Yes		Global	No
innodb_ft_enable_diag_print	Yes	Yes	Yes		Global	Yes
innodb_ft_enable_stopword	Yes	Yes	Yes		Global	Yes
innodb_ft_max_token_size	Yes	Yes	Yes		Global	No
innodb_ft_min_token_size	Yes	Yes	Yes		Global	No
innodb_ft_num_word_optimize	Yes	Yes	Yes		Global	Yes
innodb_ft_result_cache_limit	Yes	Yes	Yes		Global	Yes
innodb_ft_server_stopword_table	Yes	Yes	Yes		Global	Yes
innodb_ft_sort_pll_degree	Yes	Yes	Yes		Global	No
innodb_ft_total_cache_size	Yes	Yes	Yes		Global	No
innodb_ft_user_stopword_table	Yes	Yes	Yes		Both	Yes
Innodb_have_atomic_builtins				Yes	Global	No
innodb_io_capacity	Yes	Yes	Yes		Global	Yes
innodb_io_capacity_max	Yes	Yes	Yes		Global	Yes
innodb_large_prefix	Yes	Yes	Yes		Global	Yes
innodb_lock_wait_timeout	Yes	Yes	Yes		Both	Yes
innodb_locks_unsafe_for_binlog	Yes	Yes	Yes		Global	No
innodb_log_buffer_size	Yes	Yes	Yes		Global	No
innodb_log_compressed_pages	Yes	Yes	Yes		Global	Yes
innodb_log_file_size	Yes	Yes	Yes		Global	No
innodb_log_files_in_group	Yes	Yes	Yes		Global	No
innodb_log_group_home_dir	Yes	Yes	Yes		Global	No
Innodb_log_waits				Yes	Global	No
innodb_log_write_ahead_size	Yes	Yes	Yes		Global	Yes
Innodb_log_write_requests				Yes	Global	No
Innodb_log_writes				Yes	Global	No
innodb_lru_scan_depth	Yes	Yes	Yes		Global	Yes
innodb_max_dirty_pages_pct	Yes	Yes	Yes		Global	Yes
innodb_max_dirty_pages_pct_lwm	Yes	Yes	Yes		Global	Yes
innodb_max_purge_lag	Yes	Yes	Yes		Global	Yes
innodb_max_purge_lag_delay	Yes	Yes	Yes		Global	Yes
innodb_monitor_disable	Yes	Yes	Yes		Global	Yes
innodb_monitor_enable	Yes	Yes	Yes		Global	Yes
innodb_monitor_reset	Yes	Yes	Yes		Global	Yes
innodb_monitor_reset_all	Yes	Yes	Yes		Global	Yes
Innodb_num_open_files				Yes	Global	No
innodb_old_blocks_pct	Yes	Yes	Yes		Global	Yes

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
innodb_old_blocks_time	Yes	Yes	Yes		Global	Yes
innodb_online_alter_log_max_size	Yes	Yes	Yes		Global	Yes
innodb_open_files	Yes	Yes	Yes		Global	No
innodb_optimize_fulltext_only	Yes	Yes	Yes		Global	Yes
Innodb_os_log_fsyncs				Yes	Global	No
Innodb_os_log_pending_fsyncs				Yes	Global	No
Innodb_os_log_pending_writes				Yes	Global	No
Innodb_os_log_written				Yes	Global	No
innodb_page_cleaners	Yes	Yes	Yes		Global	No
innodb_page_size	Yes	Yes	Yes		Global	No
Innodb_page_size				Yes	Global	No
Innodb_pages_created				Yes	Global	No
Innodb_pages_read				Yes	Global	No
Innodb_pages_written				Yes	Global	No
innodb_print_all_deadlocks	Yes	Yes	Yes		Global	Yes
innodb_purge_batch_size	Yes	Yes	Yes		Global	Yes
innodb_purge_threads	Yes	Yes	Yes		Global	No
innodb_random_read_ahead	Yes	Yes	Yes		Global	Yes
innodb_read_ahead_threshold	Yes	Yes	Yes		Global	Yes
innodb_read_io_threads	Yes	Yes	Yes		Global	No
innodb_read_only	Yes	Yes	Yes		Global	No
innodb_replication_delay	Yes	Yes	Yes		Global	Yes
innodb_rollback_on_timeout	Yes	Yes	Yes		Global	No
innodb_rollback_segments	Yes	Yes	Yes		Global	Yes
Innodb_row_lock_current_waits				Yes	Global	No
Innodb_row_lock_time				Yes	Global	No
Innodb_row_lock_time_avg				Yes	Global	No
Innodb_row_lock_time_max				Yes	Global	No
Innodb_row_lock_waits				Yes	Global	No
Innodb_rows_deleted				Yes	Global	No
Innodb_rows_inserted				Yes	Global	No
Innodb_rows_read				Yes	Global	No
Innodb_rows_updated				Yes	Global	No
innodb_sort_buffer_size	Yes	Yes	Yes		Global	No
innodb_spin_wait_delay	Yes	Yes	Yes		Global	Yes
innodb_stats_auto_recalc	Yes	Yes	Yes		Global	Yes
innodb_stats_method	Yes	Yes	Yes		Global	Yes
innodb_stats_on_metadata	Yes	Yes	Yes		Global	Yes

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
innodb_stats_persistent	Yes	Yes	Yes		Global	Yes
innodb_stats_persistent_sample_pages	Yes	Yes	Yes		Global	Yes
innodb_stats_sample_pages	Yes	Yes	Yes		Global	Yes
innodb_stats_transient_sample_pages	Yes	Yes	Yes		Global	Yes
innodb-status-file	Yes	Yes				
innodb_status_output	Yes	Yes	Yes		Global	Yes
innodb_status_output_locks	Yes	Yes	Yes		Global	Yes
innodb_strict_mode	Yes	Yes	Yes		Both	Yes
innodb_support_xa	Yes	Yes	Yes		Both	Yes
innodb_sync_array_size	Yes	Yes	Yes		Global	No
innodb_sync_spin_loops	Yes	Yes	Yes		Global	Yes
innodb_table_locks	Yes	Yes	Yes		Both	Yes
innodb_temp_data_file_path	Yes	Yes	Yes		Global	No
innodb_thread_concurrency	Yes	Yes	Yes		Global	Yes
innodb_thread_sleep_delay	Yes	Yes	Yes		Global	Yes
Innodb_truncated_status_writes				Yes	Global	No
innodb_undo_directory	Yes	Yes	Yes		Global	No
innodb_undo_logs	Yes	Yes	Yes		Global	Yes
innodb_undo_tablespaces	Yes	Yes	Yes		Global	No
innodb_use_native_aio	Yes	Yes	Yes		Global	No
innodb_use_sys_malloc	Yes	Yes	Yes		Global	No
innodb_version			Yes		Global	No
innodb_write_io_threads	Yes	Yes	Yes		Global	No
insert_id			Yes		Session	Yes
install	Yes					
install-manual	Yes					
interactive_timeout	Yes	Yes	Yes		Both	Yes
join_buffer_size	Yes	Yes	Yes		Both	Yes
keep_files_on_create	Yes	Yes	Yes		Both	Yes
Key_blocks_not_flushed				Yes	Global	No
Key_blocks_unused				Yes	Global	No
Key_blocks_used				Yes	Global	No
key_buffer_size	Yes	Yes	Yes		Global	Yes
key_cache_age_threshold	Yes	Yes	Yes		Global	Yes
key_cache_block_size	Yes	Yes	Yes		Global	Yes
key_cache_division_limit	Yes	Yes	Yes		Global	Yes
Key_read_requests				Yes	Global	No
Key_reads				Yes	Global	No

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
Key_write_requests				Yes	Global	No
Key_writes				Yes	Global	No
language	Yes	Yes	Yes		Global	No
large_files_support			Yes		Global	No
large_page_size			Yes		Global	No
large-pages	Yes	Yes			Global	No
- *Variable*: large_pages			Yes		Global	No
last_insert_id			Yes		Session	Yes
Last_query_cost				Yes	Session	No
Last_query_partial_plans				Yes	Session	No
lc-messages	Yes	Yes			Both	Yes
- *Variable*: lc_messages			Yes		Both	Yes
lc-messages-dir	Yes	Yes			Global	No
- *Variable*: lc_messages_dir			Yes		Global	No
lc_time_names			Yes		Both	Yes
license			Yes		Global	No
local_infile			Yes		Global	Yes
local-service	Yes					
lock_wait_timeout	Yes	Yes	Yes		Both	Yes
locked_in_memory			Yes		Global	No
log_bin			Yes		Global	No
log-bin	Yes	Yes	Yes		Global	No
log_bin_basename			Yes		Global	No
log_bin_index			Yes		Global	No
log-bin-index	Yes	Yes				
log-bin-trust-function-creators	Yes	Yes			Global	Yes
- *Variable*: log_bin_trust_function_creators			Yes		Global	Yes
log_bin_use_v1_row_events	Yes	Yes	Yes		Global	No
log-bin-use-v1-row-events	Yes	Yes			Global	No
- *Variable*: log_bin_use_v1_row_events			Yes		Global	No
log-error	Yes	Yes			Global	No
- *Variable*: log_error			Yes		Global	No

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
log_error_verbosity	Yes	Yes	Yes		Global	Yes
log-isam	Yes	Yes				
log-output	Yes	Yes			Global	Yes
- *Variable*: log_output			Yes		Global	Yes
log-queries-not-using-indexes	Yes	Yes			Global	Yes
- *Variable*: log_queries_not_using_indexes			Yes		Global	Yes
log-raw	Yes	Yes				
log-short-format	Yes	Yes				
log-slave-updates	Yes	Yes			Global	No
- *Variable*: log_slave_updates			Yes		Global	No
log_slave_updates	Yes	Yes	Yes		Global	No
log_slow_admin_statements			Yes		Global	Yes
log-slow-admin-statements	Yes	Yes				
log_slow_slave_statements			Yes		Global	Yes
log-slow-slave-statements	Yes	Yes				
log-tc	Yes	Yes				
log-tc-size	Yes	Yes				
log_throttle_queries_not_using_indexes			Yes		Global	Yes
log_timestamps	Yes	Yes	Yes		Global	Yes
log-warnings	Yes	Yes			Global	Yes
- *Variable*: log_warnings			Yes		Global	Yes
long_query_time	Yes	Yes	Yes		Both	Yes
low-priority-updates	Yes	Yes			Both	Yes
- *Variable*: low_priority_updates			Yes		Both	Yes
lower_case_file_system			Yes		Global	No
lower_case_table_names	Yes	Yes	Yes		Global	No
master-info-file	Yes	Yes				
master_info_repository	Yes	Yes	Yes		Global	Yes
master-info-repository	Yes	Yes				
- *Variable*: master_info_repository						

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
master-retry-count	Yes	Yes				
master_verify_checksum			Yes		Global	Yes
master-verify-checksum	Yes	Yes				
- *Variable*: master_verify_checksum						
max_allowed_packet	Yes	Yes	Yes		Global	Yes
max_binlog_cache_size	Yes	Yes	Yes		Global	Yes
max-binlog-dump-events	Yes	Yes				
max_binlog_size	Yes	Yes	Yes		Global	Yes
max_binlog_stmt_cache_size	Yes	Yes	Yes		Global	Yes
max_connect_errors	Yes	Yes	Yes		Global	Yes
max_connections	Yes	Yes	Yes		Global	Yes
max_delayed_threads	Yes	Yes	Yes		Both	Yes
max_error_count	Yes	Yes	Yes		Both	Yes
max_heap_table_size	Yes	Yes	Yes		Both	Yes
max_insert_delayed_threads			Yes		Both	Yes
max_join_size	Yes	Yes	Yes		Both	Yes
max_length_for_sort_data	Yes	Yes	Yes		Both	Yes
max_prepared_stmt_count	Yes	Yes	Yes		Global	Yes
max_relay_log_size	Yes	Yes	Yes		Global	Yes
max_seeks_for_key	Yes	Yes	Yes		Both	Yes
max_sort_length	Yes	Yes	Yes		Both	Yes
max_sp_recursion_depth	Yes	Yes	Yes		Both	Yes
max_statement_time	Yes	Yes	Yes		Both	Yes
Max_statement_time_exceeded				Yes	Both	No
Max_statement_time_set				Yes	Both	No
Max_statement_time_set_failed				Yes	Both	No
Max_used_connections				Yes	Global	No
Max_used_connections_time				Yes	Global	No
max_user_connections	Yes	Yes	Yes		Both	Yes
max_write_lock_count	Yes	Yes	Yes		Global	Yes
memlock	Yes	Yes	Yes		Global	No
metadata_locks_cache_size			Yes		Global	No
metadata_locks_hash_instances			Yes		Global	No
min-examined-row-limit	Yes	Yes	Yes		Both	Yes

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
myisam-block-size	Yes	Yes				
myisam_data_pointer_size	Yes	Yes	Yes		Global	Yes
myisam_max_sort_file_size	Yes	Yes	Yes		Global	Yes
myisam_mmap_size	Yes	Yes	Yes		Global	No
myisam-recover-options	Yes	Yes				
- *Variable*: myisam_recover_options						
myisam_recover_options			Yes		Global	No
myisam_repair_threads	Yes	Yes	Yes		Both	Yes
myisam_sort_buffer_size	Yes	Yes	Yes		Both	Yes
myisam_stats_method	Yes	Yes	Yes		Both	Yes
myisam_use_mmap	Yes	Yes	Yes		Global	Yes
named_pipe			Yes		Global	No
Ndb_conflict_fn_max				Yes	Global	No
Ndb_conflict_fn_old				Yes	Global	No
Ndb_number_of_data_nodes				Yes	Global	No
net_buffer_length	Yes	Yes	Yes		Both	Yes
net_read_timeout	Yes	Yes	Yes		Both	Yes
net_retry_count	Yes	Yes	Yes		Both	Yes
net_write_timeout	Yes	Yes	Yes		Both	Yes
new	Yes	Yes	Yes		Both	Yes
no-defaults	Yes					
Not_flushed_delayed_rows				Yes	Global	No
old	Yes	Yes	Yes		Global	No
old-alter-table	Yes	Yes			Both	Yes
- *Variable*: old_alter_table			Yes		Both	Yes
old_passwords			Yes		Both	Yes
old-style-user-limits	Yes	Yes				
Open_files				Yes	Global	No
open-files-limit	Yes	Yes			Global	No
- *Variable*: open_files_limit			Yes		Global	No
Open_streams				Yes	Global	No
Open_table_definitions				Yes	Global	No
Open_tables				Yes	Both	No

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
Opened_files				Yes	Global	No
Opened_table_definitions				Yes	Both	No
Opened_tables				Yes	Both	No
optimizer_prune_level	Yes	Yes	Yes		Both	Yes
optimizer_search_depth	Yes	Yes	Yes		Both	Yes
optimizer_switch	Yes	Yes	Yes		Both	Yes
optimizer_trace			Yes		Both	Yes
optimizer_trace_features			Yes		Both	Yes
optimizer_trace_limit			Yes		Both	Yes
optimizer_trace_max_mem_size			Yes		Both	Yes
optimizer_trace_offset			Yes		Both	Yes
partition	Yes	Yes				
- *Variable*: have_partitioning						
performance_schema	Yes	Yes	Yes		Global	No
Performance_schema_accounts_lost				Yes	Global	No
performance_schema_accounts_size	Yes	Yes	Yes		Global	No
Performance_schema_cond_classes_lost				Yes	Global	No
Performance_schema_cond_instances_lost				Yes	Global	No
performance-schema-consumer-events-stages-current	Yes	Yes				
performance-schema-consumer-events-stages-history	Yes	Yes				
performance-schema-consumer-events-stages-history-long	Yes	Yes				
performance-schema-consumer-events-statements-current	Yes	Yes				
performance-schema-consumer-events-statements-history	Yes	Yes				

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
performance-schema-consumer-events-statements-history-long	Yes	Yes				
performance-schema-consumer-events-transactions-current	Yes	Yes				
performance-schema-consumer-events-transactions-history	Yes	Yes				
performance-schema-consumer-events-transactions-history-long	Yes	Yes				
performance-schema-consumer-events-waits-current	Yes	Yes				
performance-schema-consumer-events-waits-history	Yes	Yes				
performance-schema-consumer-events-waits-history-long	Yes	Yes				
performance-schema-consumer-global-instrumentation	Yes	Yes				
performance-schema-consumer-statements-digest	Yes	Yes				
performance-schema-consumer-thread-instrumentation	Yes	Yes				
Performance_schema_digest_lost				Yes	Global	No
performance_schema_digests_size	Yes	Yes	Yes		Global	No
performance_schema_events_stages_history_long_size	Yes	Yes	Yes		Global	No

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
performance_schema_events_stages_history_size	Yes	Yes	Yes		Global	No
performance_schema_events_statements_history_long_size	Yes	Yes	Yes		Global	No
performance_schema_events_statements_history_size	Yes	Yes	Yes		Global	No
performance_schema_events_transactions_history_long_size	Yes	Yes	Yes		Global	No
performance_schema_events_transactions_history_size	Yes	Yes	Yes		Global	No
performance_schema_events_waits_history_long_size	Yes	Yes	Yes		Global	No
performance_schema_events_waits_history_size	Yes	Yes	Yes		Global	No
Performance_schema_file_classes_lost				Yes	Global	No
Performance_schema_file_handles_lost				Yes	Global	No
Performance_schema_file_instances_lost				Yes	Global	No
Performance_schema_hosts_lost				Yes	Global	No
performance_schema_hosts_size	Yes	Yes	Yes		Global	No
performance-schema-instrument	Yes	Yes				
Performance_schema_locker_lost				Yes	Global	No
performance_schema_max_cond_classes	Yes	Yes	Yes		Global	No
performance_schema_max_cond_instances	Yes	Yes	Yes		Global	No
performance_schema_max_file_classes	Yes	Yes	Yes		Global	No
performance_schema_max_file_handles	Yes	Yes	Yes		Global	No
performance_schema_max_file_instances	Yes	Yes	Yes		Global	No
performance_schema_max_memory_classes	Yes	Yes	Yes		Global	No
performance_schema_max_metadata_locks	Yes	Yes	Yes		Global	No
performance_schema_max_mutex_classes	Yes	Yes	Yes		Global	No
performance_schema_max_mutex_instances	Yes	Yes	Yes		Global	No
performance_schema_max_prepared_statements_instances	Yes	Yes	Yes		Global	No
performance_schema_max_program_instances	Yes	Yes	Yes		Global	No
performance_schema_max_rwlock_classes	Yes	Yes	Yes		Global	No
performance_schema_max_rwlock_instances	Yes	Yes	Yes		Global	No
performance_schema_max_socket_classes	Yes	Yes	Yes		Global	No
performance_schema_max_socket_instances	Yes	Yes	Yes		Global	No
performance_schema_max_stage_classes	Yes	Yes	Yes		Global	No
performance_schema_max_statement_classes	Yes	Yes	Yes		Global	No
performance_schema_max_statement_stack	Yes	Yes	Yes		Global	No
performance_schema_max_table_handles	Yes	Yes	Yes		Global	No
performance_schema_max_table_instances	Yes	Yes	Yes		Global	No
performance_schema_max_thread_classes	Yes	Yes	Yes		Global	No
performance_schema_max_thread_instances	Yes	Yes	Yes		Global	No

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
Performance_schema_memory_classes_lost				Yes	Global	No
Performance_schema_metadata_lock_lost				Yes	Global	No
Performance_schema_mutex_classes_lost				Yes	Global	No
Performance_schema_mutex_instances_lost				Yes	Global	No
Performance_schema_nested_statement_lost				Yes	Global	No
Performance_schema_prepared_statements_lost				Yes	Global	No
Performance_schema_program_lost				Yes	Global	No
Performance_schema_rwlock_classes_lost				Yes	Global	No
Performance_schema_rwlock_instances_lost				Yes	Global	No
Performance_schema_session_connect_attrs_lost				Yes	Global	No
performance_schema_session_connect_attrs_size	Yes	Yes	Yes		Global	No
performance_schema_setup_actors_size	Yes	Yes	Yes		Global	No
performance_schema_setup_objects_size	Yes	Yes	Yes		Global	No
Performance_schema_socket_classes_lost				Yes	Global	No
Performance_schema_socket_instances_lost				Yes	Global	No
Performance_schema_stage_classes_lost				Yes	Global	No
Performance_schema_statement_classes_lost				Yes	Global	No
Performance_schema_table_handles_lost				Yes	Global	No
Performance_schema_table_instances_lost				Yes	Global	No
Performance_schema_thread_classes_lost				Yes	Global	No
Performance_schema_thread_instances_lost				Yes	Global	No
Performance_schema_users_lost				Yes	Global	No
performance_schema_users_size	Yes	Yes	Yes		Global	No
pid-file	Yes	Yes			Global	No
- *Variable*: pid_file			Yes		Global	No
plugin	Yes	Yes				
plugin_dir	Yes	Yes	Yes		Global	No
plugin-load	Yes	Yes				
plugin-load-add	Yes	Yes				
port	Yes	Yes	Yes		Global	No
port-open-timeout	Yes	Yes				
preload_buffer_size	Yes	Yes	Yes		Both	Yes
Prepared_stmt_count				Yes	Global	No
print-defaults	Yes					
profiling			Yes		Both	Yes
profiling_history_size	Yes	Yes	Yes		Both	Yes
protocol_version			Yes		Global	No
proxy_user			Yes		Session	No

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
pseudo_slave_mode			Yes		Session	Yes
pseudo_thread_id			Yes		Session	Yes
Qcache_free_blocks				Yes	Global	No
Qcache_free_memory				Yes	Global	No
Qcache_hits				Yes	Global	No
Qcache_inserts				Yes	Global	No
Qcache_lowmem_prunes				Yes	Global	No
Qcache_not_cached				Yes	Global	No
Qcache_queries_in_cache				Yes	Global	No
Qcache_total_blocks				Yes	Global	No
Queries				Yes	Both	No
query_alloc_block_size	Yes	Yes	Yes		Both	Yes
query_cache_limit	Yes	Yes	Yes		Global	Yes
query_cache_min_res_unit	Yes	Yes	Yes		Global	Yes
query_cache_size	Yes	Yes	Yes		Global	Yes
query_cache_type	Yes	Yes	Yes		Both	Yes
query_cache_wlock_invalidate	Yes	Yes	Yes		Both	Yes
query_prealloc_size	Yes	Yes	Yes		Both	Yes
Questions				Yes	Both	No
rand_seed1			Yes		Session	Yes
rand_seed2			Yes		Session	Yes
range_alloc_block_size	Yes	Yes	Yes		Both	Yes
read_buffer_size	Yes	Yes	Yes		Both	Yes
read_only	Yes	Yes	Yes		Global	Yes
read_rnd_buffer_size	Yes	Yes	Yes		Both	Yes
relay-log	Yes	Yes			Global	No
- *Variable*: relay_log			Yes		Global	No
relay_log_basename			Yes		Global	No
relay-log-index	Yes	Yes			Global	No
- *Variable*: relay_log_index			Yes		Global	No
relay_log_index	Yes	Yes	Yes		Global	No
relay-log-info-file	Yes	Yes				
- *Variable*: relay_log_info_file						
relay_log_info_file	Yes	Yes	Yes		Global	No
relay-log-info-repository	Yes	Yes				

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
- *Variable*: relay_log_info_repository						
relay_log_info_repository			Yes		Global	Yes
relay_log_purge	Yes	Yes	Yes		Global	Yes
relay_log_recovery	Yes	Yes	Yes		Global	No
relay-log-recovery	Yes	Yes				
- *Variable*: relay_log_recovery						
relay_log_space_limit	Yes	Yes	Yes		Global	No
remove	Yes					
replicate-do-db	Yes	Yes				
replicate-do-table	Yes	Yes				
replicate-ignore-db	Yes	Yes				
replicate-ignore-table	Yes	Yes				
replicate-rewrite-db	Yes	Yes				
replicate-same-server-id	Yes	Yes				
replicate-wild-do-table	Yes	Yes				
replicate-wild-ignore-table	Yes	Yes				
report-host	Yes	Yes			Global	No
- *Variable*: report_host			Yes		Global	No
report-password	Yes	Yes			Global	No
- *Variable*: report_password			Yes		Global	No
report-port	Yes	Yes			Global	No
- *Variable*: report_port			Yes		Global	No
report-user	Yes	Yes			Global	No
- *Variable*: report_user			Yes		Global	No
Rpl_semi_sync_master_clients				Yes	Global	No
rpl_semi_sync_master_enabled			Yes		Global	Yes
Rpl_semi_sync_master_net_avg_wait_time				Yes	Global	No
Rpl_semi_sync_master_net_wait_time				Yes	Global	No
Rpl_semi_sync_master_net_waits				Yes	Global	No

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
Rpl_semi_sync_master_no_times				Yes	Global	No
Rpl_semi_sync_master_no_tx				Yes	Global	No
Rpl_semi_sync_master_status				Yes	Global	No
Rpl_semi_sync_master_timefunc_failures				Yes	Global	No
rpl_semi_sync_master_timeout			Yes		Global	Yes
rpl_semi_sync_master_trace_level			Yes		Global	Yes
Rpl_semi_sync_master_tx_avg_wait_time				Yes	Global	No
Rpl_semi_sync_master_tx_wait_time				Yes	Global	No
Rpl_semi_sync_master_tx_waits				Yes	Global	No
rpl_semi_sync_master_wait_for_slave_count			Yes		Global	Yes
rpl_semi_sync_master_wait_no_slave			Yes		Global	Yes
rpl_semi_sync_master_wait_point			Yes		Global	Yes
Rpl_semi_sync_master_wait_pos_backtraverse				Yes	Global	No
Rpl_semi_sync_master_wait_sessions				Yes	Global	No
Rpl_semi_sync_master_yes_tx				Yes	Global	No
rpl_semi_sync_slave_enabled			Yes		Global	Yes
Rpl_semi_sync_slave_status				Yes	Global	No
rpl_semi_sync_slave_trace_level			Yes		Global	Yes
rpl_stop_slave_timeout	Yes	Yes	Yes		Global	Yes
Rsa_public_key				Yes	Global	No
safe-user-create	Yes	Yes				
secure-auth	Yes	Yes			Global	Yes
- *Variable*: secure_auth			Yes		Global	Yes
secure-file-priv	Yes	Yes			Global	No
- *Variable*: secure_file_priv			Yes		Global	No
Select_full_join				Yes	Both	No
Select_full_range_join				Yes	Both	No
Select_range				Yes	Both	No
Select_range_check				Yes	Both	No
Select_scan				Yes	Both	No
server-id [2162]	Yes	Yes			Global	Yes
- *Variable*: server_id			Yes		Global	Yes
server_uuid [2162]			Yes		Global	No
session_track_schema	Yes	Yes	Yes		Both	Yes
session_track_state_change	Yes	Yes	Yes		Both	Yes
session_track_system_variables	Yes	Yes	Yes		Both	Yes

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
sha256_password_private_key_path			Yes		Global	No
sha256_password_public_key_path			Yes		Global	No
shared_memory			Yes		Global	No
shared_memory_base_name			Yes		Global	No
show-slave-auth-info	Yes	Yes				
skip-character-set-client-handshake	Yes	Yes				
skip-concurrent-insert	Yes	Yes				
- *Variable*: concurrent_insert						
skip-event-scheduler	Yes	Yes				
skip_external_locking	Yes	Yes	Yes		Global	No
skip-grant-tables	Yes	Yes				
skip-host-cache	Yes	Yes				
skip-name-resolve	Yes	Yes			Global	No
- *Variable*: skip_name_resolve			Yes		Global	No
skip-networking	Yes	Yes			Global	No
- *Variable*: skip_networking			Yes		Global	No
skip-new	Yes	Yes				
skip-partition	Yes	Yes				
skip-show-database	Yes	Yes			Global	No
- *Variable*: skip_show_database			Yes		Global	No
skip-slave-start	Yes	Yes				
skip-ssl	Yes	Yes				
skip-stack-trace	Yes	Yes				
skip-symbolic-links	Yes					
slave_allow_batching	Yes	Yes	Yes		Global	Yes
slave_checkpoint_group	Yes	Yes	Yes		Global	Yes
slave-checkpoint-group	Yes	Yes				
- *Variable*: slave_checkpoint_group						

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
slave_checkpoint_period	Yes	Yes	Yes		Global	Yes
slave-checkpoint-period	Yes	Yes				
- *Variable*: slave_checkpoint_period						
slave_compressed_protocol	Yes	Yes	Yes		Global	Yes
slave_exec_mode	Yes	Yes	Yes		Global	Yes
Slave_heartbeat_period				Yes	Global	No
Slave_last_heartbeat				Yes	Global	No
slave-load-tmpdir	Yes	Yes			Global	No
- *Variable*: slave_load_tmpdir			Yes		Global	No
slave_max_allowed_packet			Yes		Global	Yes
slave-max-allowed-packet	Yes	Yes				
- *Variable*: slave_max_allowed_packet						
slave-net-timeout	Yes	Yes			Global	Yes
- *Variable*: slave_net_timeout			Yes		Global	Yes
Slave_open_temp_tables				Yes	Global	No
slave_parallel_type			Yes		Global	Yes
slave-parallel-type	Yes	Yes				
- *Variable*: slave_parallel_type						
slave_parallel_workers			Yes		Global	Yes
slave-parallel-workers	Yes	Yes				
- *Variable*: slave_parallel_workers						
slave_pending_jobs_size_max			Yes		Global	Yes
slave-pending-jobs-size-max	Yes					
- *Variable*: slave_pending_jobs_size_max						
Slave_received_heartbeats				Yes	Global	No
Slave_retried_transactions				Yes	Global	No
slave-rows-search-algorithms	Yes	Yes				
- *Variable*: slave_rows_search_algorithms						
slave_rows_search_algorithms			Yes		Global	Yes

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
Slave_running				Yes	Global	No
slave-skip-errors	Yes	Yes			Global	No
- *Variable*: slave_skip_errors			Yes		Global	No
slave_sql_verify_checksum			Yes		Global	Yes
slave-sql-verify-checksum	Yes	Yes				
slave_transaction_retries	Yes	Yes	Yes		Global	Yes
slave_type_conversions	Yes	Yes	Yes		Global	No
Slow_launch_threads				Yes	Both	No
slow_launch_time	Yes	Yes	Yes		Global	Yes
Slow_queries				Yes	Both	No
slow-query-log	Yes	Yes			Global	Yes
- *Variable*: slow_query_log			Yes		Global	Yes
slow_query_log_file	Yes	Yes	Yes		Global	Yes
slow-start-timeout	Yes	Yes				
socket	Yes	Yes	Yes		Global	No
sort_buffer_size	Yes	Yes	Yes		Both	Yes
Sort_merge_passes				Yes	Both	No
Sort_range				Yes	Both	No
Sort_rows				Yes	Both	No
Sort_scan				Yes	Both	No
sporadic-binlog-dump-fail	Yes	Yes				
sql_auto_is_null			Yes		Both	Yes
sql_big_selects			Yes		Both	Yes
sql_buffer_result			Yes		Both	Yes
sql_log_bin			Yes		Both	Yes
sql_log_off			Yes		Both	Yes
sql-mode	Yes	Yes			Both	Yes
- *Variable*: sql_mode			Yes		Both	Yes
sql_notes			Yes		Both	Yes
sql_quote_show_create			Yes		Both	Yes
sql_safe_updates			Yes		Both	Yes
sql_select_limit			Yes		Both	Yes
sql_slave_skip_counter			Yes		Global	Yes
sql_warnings			Yes		Both	Yes

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
ssl	Yes	Yes				
Ssl_accept_renegotiates				Yes	Global	No
Ssl_accepts				Yes	Global	No
ssl-ca	Yes	Yes			Global	No
- *Variable*: ssl_ca			Yes		Global	No
Ssl_callback_cache_hits				Yes	Global	No
ssl-capath	Yes	Yes			Global	No
- *Variable*: ssl_capath			Yes		Global	No
ssl-cert	Yes	Yes			Global	No
- *Variable*: ssl_cert			Yes		Global	No
ssl-cipher	Yes	Yes			Global	No
- *Variable*: ssl_cipher			Yes		Global	No
Ssl_cipher				Yes	Both	No
Ssl_cipher_list				Yes	Both	No
Ssl_client_connects				Yes	Global	No
Ssl_connect_renegotiates				Yes	Global	No
ssl-crl	Yes	Yes			Global	No
- *Variable*: ssl_crl			Yes		Global	No
ssl-crlpath	Yes	Yes			Global	No
- *Variable*: ssl_crlpath			Yes		Global	No
Ssl_ctx_verify_depth				Yes	Global	No
Ssl_ctx_verify_mode				Yes	Global	No
Ssl_default_timeout				Yes	Both	No
Ssl_finished_accepts				Yes	Global	No
Ssl_finished_connects				Yes	Global	No
ssl-key	Yes	Yes			Global	No
- *Variable*: ssl_key			Yes		Global	No
Ssl_server_not_after				Yes	Both	No
Ssl_server_not_before				Yes	Both	No
Ssl_session_cache_hits				Yes	Global	No
Ssl_session_cache_misses				Yes	Global	No
Ssl_session_cache_mode				Yes	Global	No
Ssl_session_cache_overflows				Yes	Global	No
Ssl_session_cache_size				Yes	Global	No

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
Ssl_session_cache_timeouts				Yes	Global	No
Ssl_sessions_reused				Yes	Both	No
Ssl_used_session_cache_entries				Yes	Global	No
Ssl_verify_depth				Yes	Both	No
Ssl_verify_mode				Yes	Both	No
ssl-verify-server-cert	Yes	Yes				
Ssl_version				Yes	Both	No
standalone	Yes	Yes				
storage_engine			Yes		Both	Yes
stored_program_cache	Yes	Yes	Yes		Global	Yes
super-large-pages	Yes	Yes				
symbolic-links	Yes	Yes				
sync_binlog	Yes	Yes	Yes		Global	Yes
sync_frm	Yes	Yes	Yes		Global	Yes
sync_master_info	Yes	Yes	Yes		Global	Yes
sync_relay_log	Yes	Yes	Yes		Global	Yes
sync_relay_log_info	Yes	Yes	Yes		Global	Yes
sysdate-is-now	Yes	Yes				
system_time_zone			Yes		Global	No
table_definition_cache			Yes		Global	Yes
Table_locks_immediate				Yes	Global	No
Table_locks_waited				Yes	Global	No
table_open_cache			Yes		Global	Yes
Table_open_cache_hits				Yes	Both	No
table_open_cache_instances			Yes		Global	No
Table_open_cache_misses				Yes	Both	No
Table_open_cache_overflows				Yes	Both	No
tc-heuristic-recover	Yes	Yes				
Tc_log_max_pages_used				Yes	Global	No
Tc_log_page_size				Yes	Global	No
Tc_log_page_waits				Yes	Global	No
temp-pool	Yes	Yes				
thread_cache_size	Yes	Yes	Yes		Global	Yes
thread_concurrency	Yes	Yes	Yes		Global	No
thread_handling	Yes	Yes	Yes		Global	No
thread_stack	Yes	Yes	Yes		Global	No

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
Threads_cached				Yes	Global	No
Threads_connected				Yes	Global	No
Threads_created				Yes	Global	No
Threads_running				Yes	Global	No
time_format			Yes		Global	No
time_zone			Yes		Both	Yes
timed_mutexes	Yes	Yes	Yes		Global	Yes
timestamp			Yes		Session	Yes
tmp_table_size	Yes	Yes	Yes		Both	Yes
tmpdir	Yes	Yes	Yes		Global	No
transaction_alloc_block_size	Yes	Yes	Yes		Both	Yes
transaction-isolation	Yes	Yes				
- *Variable*: tx_isolation						
transaction_prealloc_size	Yes	Yes	Yes		Both	Yes
transaction-read-only	Yes	Yes				
- *Variable*: tx_read_only						
tx_isolation			Yes		Both	Yes
tx_read_only			Yes		Both	Yes
unique_checks			Yes		Both	Yes
updatable_views_with_limit	Yes	Yes	Yes		Both	Yes
Uptime				Yes	Global	No
Uptime_since_flush_status				Yes	Global	No
user	Yes	Yes				
validate-password	Yes	Yes				
validate_password_dictionary_file			Yes		Global	No
validate_password_length			Yes		Global	Yes
validate_password_mixed_case_count			Yes		Global	Yes
validate_password_number_count			Yes		Global	Yes
validate_password_policy			Yes		Global	Yes
validate_password_special_char_count			Yes		Global	Yes
validate_user_plugins			Yes		Global	No
verbose	Yes	Yes				
version			Yes		Global	No
version_comment			Yes		Global	No
version_compile_machine			Yes		Global	No

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
version_compile_os			Yes		Global	No
wait_timeout	Yes	Yes	Yes		Both	Yes
warning_count			Yes		Session	No

[a] This option is dynamic, but only the server should set this information. You should not set the value of this variable manually.

[b] This option is dynamic, but only the server should set this information. You should not set the value of this variable manually.

5.1.2 Server Configuration Defaults

The MySQL server has many operating parameters, which you can change at server startup using command-line options or configuration files (option files). It is also possible to change many parameters at runtime. For general instructions on setting parameters at startup or runtime, see Section 5.1.3, “Server Command Options”, and Section 5.1.4, “Server System Variables”.

On Unix platforms, mysql_install_db creates a default option file named my.cnf in the base installation directory. This file is created from a template included in the distribution package named my-default.cnf. You can find the template in or under the base installation directory. When started using mysqld_safe, the server uses my.cnf file by default. If my.cnf already exists, mysql_install_db assumes it to be in use and writes a new file named my-new.cnf instead.

With one exception, the settings in the default option file are commented and have no effect. The exception is that the file changes the sql_mode system variable from its default of NO_ENGINE_SUBSTITUTION to also include STRICT_TRANS_TABLES:

```
sql_mode=NO_ENGINE_SUBSTITUTION,STRICT_TRANS_TABLES
```

This setting produces a server configuration that results in errors rather than warnings for bad data in operations that modify transactional tables. See Section 5.1.7, “Server SQL Modes”.

On Windows, MySQL Installer interacts with the user and creates a file named my.ini in the base installation directory as the default option file. If you install on Windows from a Zip archive, you can copy the my-default.ini template file in the base installation directory to my.ini and use the latter as the default option file.

Note

On Windows, the .ini or .cnf option file extension might not be displayed.

On any platform, after completing the installation process, you can edit the default option file at any time to modify the parameters used by the server. For example, to use a parameter setting in the file that is commented with a # character at the beginning of the line, remove the #, and modify the parameter value if necessary. To disable a setting, either add a # to the beginning of the line or remove it.

For additional information about option file format and syntax, see Section 4.2.3.3, “Using Option Files”.

5.1.3 Server Command Options

When you start the mysqld server, you can specify program options using any of the methods described in Section 4.2.3, “Specifying Program Options”. The most common methods are to provide options in an option file or on the command line. However, in most cases it is desirable to make sure that the server uses the same options each time it runs. The best way to ensure this is to list them in an option file. See Section 4.2.3.3, “Using Option Files”.

mysqld reads options from the [mysqld] and [server] groups. mysqld_safe reads options from the [mysqld], [server], [mysqld_safe], and [safe_mysqld] groups. mysql.server reads options from the [mysqld] and [mysql.server] groups.

An embedded MySQL server usually reads options from the [server], [embedded], and [*xxxxx*_SERVER] groups, where *xxxxx* is the name of the application into which the server is embedded.

mysqld accepts many command options. For a brief summary, execute mysqld --help. To see the full list, use mysqld --verbose --help.

The following list shows some of the most common server options. Additional options are described in other sections:

- Options that affect security: See Section 6.1.4, "Security-Related mysqld Options and Variables".
- SSL-related options: See Section 6.3.11.4, "SSL Command Options".
- Binary log control options: See Section 5.2.4, "The Binary Log".
- Replication-related options: See Section 16.1.4, "Replication and Binary Logging Options and Variables".
- Options for loading plugins such as pluggable storage engines: See Section 5.1.8.1, "Installing and Uninstalling Plugins".
- Options specific to particular storage engines: See Section 14.2.13, "InnoDB Startup Options and System Variables" and Section 14.3.1, "MyISAM Startup Options".

You can also set the values of server system variables by using variable names as options, as described at the end of this section.

Some options control the size of buffers or caches. For a given buffer, the server might need to allocate internal data structures. These structures typically are allocated from the total memory allocated to the buffer, and the amount of space required might be platform dependent. This means that when you assign a value to an option that controls a buffer size, the amount of space actually available might differ from the value assigned. In some cases, the amount might be less than the value assigned. It is also possible that the server will adjust a value upward. For example, if you assign a value of 0 to an option for which the minimal value is 1024, the server will set the value to 1024.

Values for buffer sizes, lengths, and stack sizes are given in bytes unless otherwise specified.

Some options take file name values. Unless otherwise specified, the default file location is the data directory if the value is a relative path name. To specify the location explicitly, use an absolute path name. Suppose that the data directory is /var/mysql/data. If a file-valued option is given as a relative path name, it will be located under /var/mysql/data. If the value is an absolute path name, its location is as given by the path name.

- --help, -?

Command-Line Format	-?
	--help
Option-File Format	help

  Display a short help message and exit. Use both the --verbose and --help options to see the full message.

- --allow-suspicious-udfs

Command-Line Format	--allow-suspicious-udfs

Option-File Format	allow-suspicious-udfs	
	Permitted Values	
	Type	boolean
	Default	FALSE

This option controls whether user-defined functions that have only an xxx symbol for the main function can be loaded. By default, the option is off and only UDFs that have at least one auxiliary symbol can be loaded; this prevents attempts at loading functions from shared object files other than those containing legitimate UDFs. See Section 22.3.2.6, “User-Defined Function Security Precautions”.

- --ansi

Command-Line Format	--ansi
	-a
Option-File Format	ansi

Use standard (ANSI) SQL syntax instead of MySQL syntax. For more precise control over the server SQL mode, use the --sql-mode option instead. See Section 1.8, “MySQL Standards Compliance”, and Section 5.1.7, “Server SQL Modes”.

- --basedir=*path*, -b *path*

Command-Line Format	--basedir=path	
	-b	
Option-File Format	basedir	
System Variable Name	basedir	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	directory name

The path to the MySQL installation directory. All paths are usually resolved relative to this directory.

- --big-tables

Command-Line Format	--big-tables	
Option-File Format	big-tables	
System Variable Name	big_tables	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	boolean
	Default	OFF

Enable large result sets by saving all temporary sets in files. This option prevents most “table full” errors, but also slows down queries for which in-memory tables would suffice. Since MySQL 3.23.2, the server is able to handle large result sets automatically by using memory for small temporary tables and switching to disk tables where necessary.

- `--bind-address=addr`

Command-Line Format	`--bind-address=addr`	
Option-File Format	`bind-address`	
System Variable Name	`bind_address`	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	`string`
	Default	`*`

The MySQL server listens on a single network socket for TCP/IP connections. This socket is bound to a single address, but it is possible for an address to map onto multiple network interfaces. To specify an address, use the `--bind-address=addr` option at server startup, where `addr` is an IPv4 or IPv6 address or a host name. If `addr` is a host name, the server resolves the name to an IP address and binds to that address.

The server treats different types of addresses as follows:

- If the address is `*`, the server accepts TCP/IP connections on all server host IPv6 and IPv4 interfaces if the server host supports IPv6, or accepts TCP/IP connections on all IPv4 addresses otherwise. Use this address to permit both IPv4 and IPv6 connections on all server interfaces. This value is the default) in MySQL 5.7.

- If the address is `0.0.0.0`, the server accepts TCP/IP connections on all server host IPv4 interfaces.

- If the address is `::`, the server accepts TCP/IP connections on all server host IPv4 and IPv6 interfaces.

- If the address is an IPv4-mapped address, the server accepts TCP/IP connections for that address, in either IPv4 or IPv6 format. For example, if the server is bound to `::ffff:127.0.0.1`, clients can connect using `--host=127.0.0.1` or `--host=::ffff:127.0.0.1`.

- If the address is a “regular” IPv4 or IPv6 address (such as `127.0.0.1` or `::1`), the server accepts TCP/IP connections only for that IPv4 or IPv6 address.

If you intend to bind the server to a specific address, be sure that the `mysql.user` grant table contains an account with administrative privileges that you can use to connect to that address. Otherwise, you will not be able to shut down the server. For example, if you bind the server to `*`, you can connect to it using all existing accounts. But if you bind the server to `::1`, it accepts connections only on that address. In that case, first make sure that the `'root'@'::1'` account is present in the `mysql.user` table so you can still connect to the server to shut it down.

- `--binlog-format={ROW|STATEMENT|MIXED}`

Command-Line Format	`--binlog-format=format`
Option-File Format	`binlog-format`
System Variable Name	`binlog_format`

Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	`enumeration`
	Default	`STATEMENT`
	Valid Values	`ROW`
		`STATEMENT`
		`MIXED`

Specify whether to use row-based, statement-based, or mixed replication. Statement-based is the default in MySQL 5.7. See Section 16.1.2, “Replication Formats”.

Under some conditions, changing this variable at runtime is not possible, or causes replication to fail. See Section 5.2.4.2, “Setting The Binary Log Format”, for more information.

Setting the binary logging format without enabling binary logging sets the `binlog_format` global system variable and logs a warning.

- `--bootstrap`

Command-Line Format	`--bootstrap`	
Option-File Format	`bootstrap`	

This option is used by the `mysql_install_db` script to create the MySQL privilege tables without having to start a full MySQL server.

Replication and global transaction identifiers are automatically disabled whenever this option is used (Bug #1332602). See Section 16.1.3, “Replication with Global Transaction Identifiers”.

- `--character-sets-dir=`*`path`*

Command-Line Format	`--character-sets-dir=path`	
Option-File Format	`character-sets-dir`	
System Variable Name	`character_sets_dir`	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	`directory name`

The directory where character sets are installed. See Section 10.5, “Character Set Configuration”.

- `--character-set-client-handshake`

Command-Line Format	`--character-set-client-handshake`	
Option-File Format	`character-set-client-handshake`	
	Permitted Values	
	Type	`boolean`
	Default	`TRUE`

Do not ignore character set information sent by the client. To ignore client information and use the default server character set, use `--skip-character-set-client-handshake`; this makes MySQL behave like MySQL 4.0.

- `--character-set-filesystem=`*`charset_name`*

Command-Line Format	`--character-set-filesystem=name`	
Option-File Format	`character-set-filesystem`	
System Variable Name	`character_set_filesystem`	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	`string`
	Default	`binary`

The file system character set. This option sets the `character_set_filesystem` system variable.

- `--character-set-server=`*`charset_name`*, `-C` *`charset_name`*

Command-Line Format	`--character-set-server`	
Option-File Format	`character-set-server`	
System Variable Name	`character_set_server`	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	`string`
	Default	`latin1`

Use *`charset_name`* as the default server character set. See Section 10.5, “Character Set Configuration”. If you use this option to specify a nondefault character set, you should also use `--collation-server` to specify the collation.

- `--chroot=`*`path`*, `-r` *`path`*

Command-Line Format	`--chroot=name`	
	`-r name`	
Option-File Format	`chroot`	
	Permitted Values	
	Type	`directory name`

Put the `mysqld` server in a closed environment during startup by using the `chroot()` system call. This is a recommended security measure. Note that use of this option somewhat limits `LOAD DATA INFILE` and `SELECT ... INTO OUTFILE`.

- `--collation-server=`*`collation_name`*

Command-Line Format	`--collation-server`

Option-File Format	`collation-server`	
System Variable Name	`collation_server`	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	`string`
	Default	`latin1_swedish_ci`

Use *`collation_name`* as the default server collation. See Section 10.5, “Character Set Configuration”.

- `--console`

Command-Line Format	`--console`	
Option-File Format	`console`	
Platform Specific	Windows	

(Windows only.) Write error log messages to `stderr` and `stdout` even if `--log-error` is specified. `mysqld` does not close the console window if this option is used.

If both `--log-error` and `--console` are specified, `--console` takes precedence. The server writes to the console, but not to the log file. (In MySQL 5.5 and 5.6, the precedence is reversed: `--log-error` causes `--console` to be ignored.)

- `--core-file`

Command-Line Format	`--core-file`	
Option-File Format	`core-file`	
	Permitted Values	
	Type	`boolean`
	Default	`OFF`

Write a core file if `mysqld` dies. The name and location of the core file is system dependent. On Linux, a core file named `core.`*`pid`* is written to the current working directory of the process, which for `mysqld` is the data directory. *`pid`* represents the process ID of the server process. On Mac OS X, a core file named `core.`*`pid`* is written to the `/cores` directory. On Solaris, use the `coreadm` command to specify where to write the core file and how to name it.

For some systems, to get a core file you must also specify the `--core-file-size` option to `mysqld_safe`. See Section 4.3.2, “`mysqld_safe` — MySQL Server Startup Script”. On some systems, such as Solaris, you do not get a core file if you are also using the `--user` option. There might be additional restrictions or limitations. For example, it might be necessary to execute `ulimit -c unlimited` before starting the server. Consult your system documentation.

- `--datadir=`*`path`*, `-h` *`path`*

Command-Line Format	`--datadir=path`
	`-h`
Option-File Format	`datadir`
System Variable Name	`datadir`

Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	`directory name`

The path to the data directory.

- `--debug[=`*`debug_options`*`]`, `-# [`*`debug_options`*`]`

Command-Line Format	`--debug[=debug_options]`	
Option-File Format	`debug`	
System Variable Name	`debug`	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type (Unix)	`string`
	Default	`d:t:i:o,/tmp/mysqld.trace`
	Permitted Values	
	Type (Windows)	`string`
	Default	`d:t:i:O,\mysqld.trace`

If MySQL is configured with `-DWITH_DEBUG=1`, you can use this option to get a trace file of what `mysqld` is doing. A typical *`debug_options`* string is `d:t:o,`*`file_name`*. The default is `d:t:i:o,/tmp/mysqld.trace` on Unix and `d:t:i:O,\mysqld.trace` on Windows.

Using `-DWITH_DEBUG=1` to configure MySQL with debugging support enables you to use the `--debug="d,parser_debug"` option when you start the server. This causes the Bison parser that is used to process SQL statements to dump a parser trace to the server's standard error output. Typically, this output is written to the error log.

This option may be given multiple times. Values that begin with `+` or `-` are added to or subtracted from the previous value. For example, `--debug=T --debug=+P` sets the value to `P:T`.

For more information, see Section 22.4.3, “The DBUG Package”.

- `--debug-sync-timeout[=`*`N`*`]`

Command-Line Format	`--debug-sync-timeout[=#]`	
Option-File Format	`debug-sync-timeout`	
	Permitted Values	
	Type	`numeric`

Controls whether the Debug Sync facility for testing and debugging is enabled. Use of Debug Sync requires that MySQL be configured with the `-DENABLE_DEBUG_SYNC=1` option (see Section 2.8.4, “MySQL Source-Configuration Options”). If Debug Sync is not compiled in, this option is not available. The option value is a timeout in seconds. The default value is 0, which disables Debug Sync. To enable it, specify a value greater than 0; this value also becomes the default timeout for individual synchronization points. If the option is given without a value, the timeout is set to 300 seconds.

For a description of the Debug Sync facility and how to use synchronization points, see MySQL Internals: Test Synchronization.

- `--default-authentication-plugin=plugin_name`

Removed	5.7.2	
Command-Line Format	`--default-authentication-plugin=plugin_name`	
Option-File Format	`default-authentication-plugin`	
	Permitted Values	
	Type	`enumeration`
	Default	`mysql_native_password`
	Valid Values	`mysql_native_password` `sha256_password`

This option sets the default authentication plugin. It was removed in MySQL 5.7.2 and replaced by the `default_authentication_plugin` system variable. The variable is used the same way as the option at server startup, but also enables the default plugin value to be inspected as runtime. For usage details, see the description of `default_authentication_plugin`.

- `--default-storage-engine=type`

Command-Line Format	`--default-storage-engine=name`	
Option-File Format	`default-storage-engine`	
System Variable Name	`default_storage_engine`	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	`enumeration`
	Default	`InnoDB`

Set the default storage engine for tables. See Chapter 14, *Storage Engines*. This option sets the storage engine for permanent tables only. To set the storage engine for `TEMPORARY` tables, set the `default_tmp_storage_engine` system variable.

If you disable the default storage engine at server startup, you must set the default engine for both permanent and `TEMPORARY` tables to a different engine or the server will not start.

- `--default-time-zone=timezone`

Command-Line Format	`--default-time-zone=name`	
Option-File Format	`default-time-zone`	
	Permitted Values	
	Type	`string`

Set the default server time zone. This option sets the global `time_zone` system variable. If this option is not given, the default time zone is the same as the system time zone (given by the value of the `system_time_zone` system variable.

- `--defaults-extra-file=`*`file_name`*

  Read this option file after the global option file but (on Unix) before the user option file. If the file does not exist or is otherwise inaccessible, an error occurs. *`file_name`* is interpreted relative to the current directory if given as a relative path name rather than a full path name.

- `--defaults-file=`*`file_name`*

  Use only the given option file. If the file does not exist or is otherwise inaccessible, an error occurs. *`file_name`* is interpreted relative to the current directory if given as a relative path name rather than a full path name.

- `--defaults-group-suffix=`*`str`*

  Read not only the usual option groups, but also groups with the usual names and a suffix of *`str`*. For example, `mysqld` normally reads the `[mysqld]` group. If the `--defaults-group-suffix=_other` option is given, `mysqld` also reads the `[mysqld_other]` group.

- `--delay-key-write[={OFF|ON|ALL}]`

Command-Line Format	`--delay-key-write[=name]`	
Option-File Format	`delay-key-write`	
System Variable Name	`delay_key_write`	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	`enumeration`
	Default	`ON`
	Valid Values	`ON`
		`OFF`
		`ALL`

Specify how to use delayed key writes. Delayed key writing causes key buffers not to be flushed between writes for `MyISAM` tables. `OFF` disables delayed key writes. `ON` enables delayed key writes for those tables that were created with the `DELAY_KEY_WRITE` option. `ALL` delays key writes for all `MyISAM` tables. See Section 8.11.2, “Tuning Server Parameters”, and Section 14.3.1, “`MyISAM` Startup Options”.

Note

If you set this variable to `ALL`, you should not use `MyISAM` tables from within another program (such as another MySQL server or `myisamchk`) when the tables are in use. Doing so leads to index corruption.

- `--des-key-file=`*`file_name`*

Command-Line Format	`--des-key-file=file_name`
Option-File Format	`des-key-file`

Read the default DES keys from this file. These keys are used by the `DES_ENCRYPT()` and `DES_DECRYPT()` functions.

- `--enable-named-pipe`

Command-Line Format	`--enable-named-pipe`
Option-File Format	`enable-named-pipe`
Platform Specific	Windows

Enable support for named pipes. This option applies only on Windows.

- `--event-scheduler[=`*`value`*`]`

Command-Line Format	`--event-scheduler[=value]`	
Option-File Format	`event-scheduler`	
System Variable Name	`event_scheduler`	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	`enumeration`
	Default	`OFF`
	Valid Values	`ON`
		`OFF`
		`DISABLED`

Enable or disable, and start or stop, the event scheduler.

For detailed information, see The `--event-scheduler` Option [2386].

- `--exit-info[=`*`flags`*`]`, `-T [`*`flags`*`]`

Command-Line Format	`--exit-info[=flags]`	
	`-T [flags]`	
Option-File Format	`exit-info`	
	Permitted Values	
	Type	`numeric`

This is a bit mask of different flags that you can use for debugging the `mysqld` server. Do not use this option unless you know *exactly* what it does!

- `--external-locking`

Command-Line Format	`--external-locking`	
Option-File Format	`external-locking`	
	Permitted Values	
	Type	`boolean`

	Default	FALSE

Enable external locking (system locking), which is disabled by default as of MySQL 4.0. Note that if you use this option on a system on which `lockd` does not fully work (such as Linux), it is easy for `mysqld` to deadlock.

To disable external locking explicitly, use `--skip-external-locking`.

External locking affects only `MyISAM` table access. For more information, including conditions under which it can and cannot be used, see Section 8.10.5, “External Locking”.

- `--flush`

Command-Line Format	`--flush`	
Option-File Format	`flush`	
System Variable Name	`flush`	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	`boolean`
	Default	`OFF`

Flush (synchronize) all changes to disk after each SQL statement. Normally, MySQL does a write of all changes to disk only after each SQL statement and lets the operating system handle the synchronizing to disk. See Section C.5.4.2, “What to Do If MySQL Keeps Crashing”.

- `--gdb`

Command-Line Format	`--gdb`	
Option-File Format	`gdb`	
	Permitted Values	
	Type	`boolean`
	Default	`FALSE`

Install an interrupt handler for `SIGINT` (needed to stop `mysqld` with `^C` to set breakpoints) and disable stack tracing and core file handling. See Section 22.4, “Debugging and Porting MySQL”.

- `--general-log[={0|1}]`

Command-Line Format	`--general-log`	
Option-File Format	`general-log`	
System Variable Name	`general_log`	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	`boolean`
	Default	`OFF`

Specify the initial general query log state. With no argument or an argument of 1, the `--general-log` option enables the log. If omitted or given with an argument of 0, the option disables the log.

- `--ignore-db-dir=dir_name`

Command-Line Format	`--ignore-db-dir`	
Option-File Format	`ignore-db-dir`	
	Permitted Values	
	Type	`directory name`

This option tells the server to ignore the given directory name for purposes of the `SHOW DATABASES` statement or `INFORMATION_SCHEMA` tables. For example, if a MySQL configuration locates the data directory at the root of a file system on Unix, the system might create a `lost+found` directory there that the server should ignore. Starting the server with `--ignore-db-dir=lost+found` causes that name not to be listed as a database.

To specify more than one name, use this option multiple times, once for each name. Specifying the option with an empty value (that is, as `--ignore-db-dir=`) resets the directory list to the empty list.

Instances of this option given at server startup are used to set the `ignore_db_dirs` system variable.

- `--init-file=file_name`

Command-Line Format	`--init-file=file_name`	
Option-File Format	`init-file`	
System Variable Name	`init_file`	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	`file name`

Read SQL statements from this file at startup. Each statement must be on a single line and should not include comments.

- `--innodb-xxx`

Set an option for the `InnoDB` storage engine. The `InnoDB` options are listed in Section 14.2.13, “`InnoDB` Startup Options and System Variables”.

- `--install [service_name]`

Command-Line Format	`--install [service_name]`

(Windows only) Install the server as a Windows service that starts automatically during Windows startup. The default service name is `MySQL` if no *`service_name`* value is given. For more information, see Section 2.3.5.7, “Starting MySQL as a Windows Service”.

Note

If the server is started with the `--defaults-file` and `--install` options, `--install` must be first.

- `--install-manual [`*`service_name`*`]`

Command-Line Format	`--install-manual [service_name]`

(Windows only) Install the server as a Windows service that must be started manually. It does not start automatically during Windows startup. The default service name is `MySQL` if no *`service_name`* value is given. For more information, see Section 2.3.5.7, “Starting MySQL as a Windows Service”.

> **Note**
>
> If the server is started with the `--defaults-file` and `--install-manual` options, `--install-manual` must be first.

- `--language=`*`lang_name`*`, -L `*`lang_name`*

Deprecated	5.6.1, by lc-messages-dir	
Command-Line Format	`--language=name`	
	`-L`	
Option-File Format	`language`	
System Variable Name	`language`	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	`directory name`
	Default	`/usr/local/mysql/share/mysql/english/`

The language to use for error messages. *`lang_name`* can be given as the language name or as the full path name to the directory where the language files are installed. See Section 10.2, “Setting the Error Message Language”.

In MySQL 5.7, `--lc-messages-dir` and `--lc-messages` should be used rather than `--language`, which is deprecated (and handled as an alias for `--lc-messages-dir`). The `--language` option will be removed in a future MySQL release.

- `--large-pages`

Command-Line Format	`--large-pages`	
Option-File Format	`large-pages`	
System Variable Name	`large_pages`	
Variable Scope	Global	
Dynamic Variable	No	
Platform Specific	Linux	
	Permitted Values	
	Type (Linux)	`boolean`
	Default	`FALSE`

Some hardware/operating system architectures support memory pages greater than the default (usually 4KB). The actual implementation of this support depends on the underlying hardware and operating system. Applications that perform a lot of memory accesses may obtain performance improvements by using large pages due to reduced Translation Lookaside Buffer (TLB) misses.

MySQL 5.7 supports the Linux implementation of large page support (which is called HugeTLB in Linux). See Section 8.11.4.2, “Enabling Large Page Support”. For Solaris support of large pages, see the description of the `--super-large-pages` option.

`--large-pages` is disabled by default.

- `--lc-messages=`*`locale_name`*

Command-Line Format	`--lc-messages=name`	
Option-File Format	`lc-messages`	
System Variable Name	`lc_messages`	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	`string`

The locale to use for error messages. The server converts the argument to a language name and combines it with the value of the `--lc-messages-dir` to produce the location for the error message file. See Section 10.2, “Setting the Error Message Language”.

- `--lc-messages-dir=`*`path`*

Command-Line Format	`--lc-messages-dir=path`	
Option-File Format	`lc-messages-dir`	
System Variable Name	`lc_messages_dir`	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	`directory name`

The directory where error messages are located. The value is used together with the value of `--lc-messages` to produce the location for the error message file. See Section 10.2, “Setting the Error Message Language”.

- `--local-service`

Command-Line Format	`--local-service`

(Windows only) A `--local-service` option following the service name causes the server to run using the `LocalService` Windows account that has limited system privileges. This account is available only for Windows XP or newer. If both `--defaults-file` and `--local-service` are given following the service name, they can be in any order. See Section 2.3.5.7, “Starting MySQL as a Windows Service”.

- `--log-error[=`*`file_name`*`]`

Command-Line Format	`--log-error[=name]`	
Option-File Format	`log-error`	
System Variable Name	`log_error`	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	`file name`

Log errors and startup messages to this file. See Section 5.2.2, "The Error Log". If you omit the file name, MySQL uses `host_name.err`. If the file name has no extension, the server adds an extension of `.err`.

- `--log-isam[=file_name]`

Command-Line Format	`--log-isam[=name]`	
Option-File Format	`log-isam`	
	Permitted Values	
	Type	`file name`

Log all `MyISAM` changes to this file (used only when debugging `MyISAM`).

- `--log-output=value,...`

Command-Line Format	`--log-output=name`	
Option-File Format	`log-output`	
System Variable Name	`log_output`	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	`set`
	Default	`FILE`
	Valid Values	`TABLE`
		`FILE`
		`NONE`

This option determines the destination for general query log and slow query log output. The option value can be given as one or more of the words `TABLE`, `FILE`, or `NONE`. `TABLE` select logging to the `general_log` and `slow_log` tables in the `mysql` database as a destination. `FILE` selects logging to log files as a destination. `NONE` disables logging. If `NONE` is present in the option value, it takes precedence over any other words that are present. `TABLE` and `FILE` can both be given to select to both log output destinations.

This option selects log output destinations, but does not enable log output. To do that, use the `--general_log` and `--slow_query_log` options. For `FILE` logging, the `--general_log_file` and `-slow_query_log_file` options determine the log file location. For more information, see Section 5.2.1, "Selecting General Query and Slow Query Log Output Destinations".

- `--log-queries-not-using-indexes`

Command-Line Format	`--log-queries-not-using-indexes`	
Option-File Format	`log-queries-not-using-indexes`	
System Variable Name	`log_queries_not_using_indexes`	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	`boolean`
	Default	`OFF`

If you are using this option with the slow query log enabled, queries that are expected to retrieve all rows are logged. See Section 5.2.5, “The Slow Query Log”. This option does not necessarily mean that no index is used. For example, a query that uses a full index scan uses an index but would be logged because the index would not limit the number of rows.

- `--log-raw`

Command-Line Format	`--log-raw[=value]`	
Option-File Format	`log-raw`	
	Permitted Values	
	Type	`boolean`
	Default	`OFF`

In MySQL 5.7, passwords in certain statements written to the general query log, slow query log, and binary log are rewritten by the server not to occur literally in plain text. Password rewriting can be suppressed for the general query log by starting the server with the `--log-raw` option. This option may be useful for diagnostic purposes, to see the exact text of statements as received by the server, but for security reasons is not recommended for production use.

For more information, see Section 6.1.2.3, “Passwords and Logging”.

- `--log-short-format`

Command-Line Format	`--log-short-format`	
Option-File Format	`log-short-format`	
	Permitted Values	
	Type	`boolean`
	Default	`FALSE`

Log less information to the binary log and slow query log, if they have been activated.

- `--log-slow-admin-statements`

Removed	5.7.1	
Command-Line Format	`--log-slow-`	through 5.7.0

	admin-statements	
Option-File Format	log-slow-admin-statements	
	Permitted Values	
	Type	boolean
	Default	OFF

Include slow administrative statements in the statements written to the slow query log. Administrative statements include `ALTER TABLE`, `ANALYZE TABLE`, `CHECK TABLE`, `CREATE INDEX`, `DROP INDEX`, `OPTIMIZE TABLE`, and `REPAIR TABLE`.

This command-line option was removed in MySQL 5.7.1 and replaced by the `log_slow_admin_statements` system variable. The system variable can be set on the command line or in option files the same way as the option, so there is no need for any changes at server startup, but the system variable also makes it possible to examine or set the value at runtime.

- `--log-tc=`*`file_name`*

Command-Line Format	--log-tc=name	
Option-File Format	log-tc	
	Permitted Values	
	Type	file name
	Default	tc.log

The name of the memory-mapped transaction coordinator log file (for XA transactions that affect multiple storage engines when the binary log is disabled). The default name is `tc.log`. The file is created under the data directory if not given as a full path name. Currently, this option is unused.

- `--log-tc-size=`*`size`*

Command-Line Format	--log-tc-size=#	
Option-File Format	log-tc-size	
	Permitted Values	
	Platform Bit Size	32
	Type	numeric
	Default	24576
	Max Value	4294967295
	Permitted Values	
	Platform Bit Size	64
	Type	numeric
	Default	24576
	Max Value	18446744073709547520

The size in bytes of the memory-mapped transaction coordinator log. The default size is 24KB.

- `--log-warnings[=level]`, `-W [level]`

Deprecated	5.7.2	
Command-Line Format	`--log-warnings[=#]`	
	`-W [#]`	
Option-File Format	`log-warnings[=#]`	
System Variable Name	`log_warnings`	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Platform Bit Size	`32`
	Type	`numeric`
	Default	`1`
	Range	`0 .. 4294967295`
	Permitted Values	
	Platform Bit Size	`64`
	Type	`numeric`
	Default	`1`
	Range	`0 .. 18446744073709547520`

Note

As of MySQL 5.7.2, the `log_error_verbosity` system variable is preferred over, and should be used instead of, the `--log-warnings` option or `log_warnings` system variable. For more information, see the descriptions of `log_error_verbosity` and `log_warnings`. The `--log-warnings` command-line option and `log_warnings` system variable are deprecated and will be removed in a future MySQL release.

Print out warnings such as `Aborted connection...` to the error log. This option is enabled (1) by default. To disable it, use `--log-warnings=0`. Specifying the option without a *`level`* value increments the current value by 1. Enabling this option by setting it greater than 0 is recommended, for example, if you use replication (you get more information about what is happening, such as messages about network failures and reconnections). If the value is greater than 1, aborted connections are written to the error log, and access-denied errors for new connection attempts are written. See Section C.5.2.11, “Communication Errors and Aborted Connections”.

If a slave server was started with `--log-warnings` enabled, the slave prints messages to the error log to provide information about its status, such as the binary log and relay log coordinates where it starts its job, when it is switching to another relay log, when it reconnects after a disconnect, and so forth. The server logs messages about statements that are unsafe for statement-based logging if `--log-warnings` is greater than 0.

- `--low-priority-updates`

<table>
<tr><th>Command-Line Format</th><td colspan="2">--low-priority-updates</td></tr>
<tr><th>Option-File Format</th><td colspan="2">low-priority-updates</td></tr>
<tr><th>System Variable Name</th><td colspan="2">low_priority_updates</td></tr>
<tr><th>Variable Scope</th><td colspan="2">Global, Session</td></tr>
<tr><th>Dynamic Variable</th><td colspan="2">Yes</td></tr>
<tr><td rowspan="3"></td><th colspan="2">Permitted Values</th></tr>
<tr><th>Type</th><td>boolean</td></tr>
<tr><th>Default</th><td>FALSE</td></tr>
</table>

Give table-modifying operations (`INSERT`, `REPLACE`, `DELETE`, `UPDATE`) lower priority than selects. This can also be done using `{INSERT | REPLACE | DELETE | UPDATE} LOW_PRIORITY ...` to lower the priority of only one query, or by `SET LOW_PRIORITY_UPDATES=1` to change the priority in one thread. This affects only storage engines that use only table-level locking (`MyISAM`, `MEMORY`, `MERGE`). See Section 8.10.2, “Table Locking Issues”.

- `--min-examined-row-limit=number`

<table>
<tr><th>Command-Line Format</th><td colspan="2">--min-examined-row-limit=#</td></tr>
<tr><th>Option-File Format</th><td colspan="2">min-examined-row-limit</td></tr>
<tr><th>System Variable Name</th><td colspan="2">min_examined_row_limit</td></tr>
<tr><th>Variable Scope</th><td colspan="2">Global, Session</td></tr>
<tr><th>Dynamic Variable</th><td colspan="2">Yes</td></tr>
<tr><td rowspan="5"></td><th colspan="2">Permitted Values</th></tr>
<tr><th>Platform Bit Size</th><td>32</td></tr>
<tr><th>Type</th><td>numeric</td></tr>
<tr><th>Default</th><td>0</td></tr>
<tr><th>Range</th><td>0 .. 4294967295</td></tr>
<tr><td rowspan="5"></td><th colspan="2">Permitted Values</th></tr>
<tr><th>Platform Bit Size</th><td>64</td></tr>
<tr><th>Type</th><td>numeric</td></tr>
<tr><th>Default</th><td>0</td></tr>
<tr><th>Range</th><td>0 .. 18446744073709547520</td></tr>
</table>

When this option is set, queries which examine fewer than *`number`* rows are not written to the slow query log. The default is 0.

- `--memlock`

Command-Line Format	`--memlock`
Option-File Format	`memlock`
System Variable Name	`locked_in_memory`
Variable Scope	Global

Dynamic Variable	No	
	Permitted Values	
	Type	`boolean`
	Default	`FALSE`

Lock the `mysqld` process in memory. This option might help if you have a problem where the operating system is causing `mysqld` to swap to disk.

`--memlock` works on systems that support the `mlockall()` system call; this includes Solaris, most Linux distributions that use a 2.4 or newer kernel, and perhaps other Unix systems. On Linux systems, you can tell whether or not `mlockall()` (and thus this option) is supported by checking to see whether or not it is defined in the system `mman.h` file, like this:

```
shell> grep mlockall /usr/include/sys/mman.h
```

If `mlockall()` is supported, you should see in the output of the previous command something like the following:

```
extern int mlockall (int __flags) __THROW;
```

Important

Use of this option may require you to run the server as `root`, which, for reasons of security, is normally not a good idea. See Section 6.1.5, “How to Run MySQL as a Normal User”.

On Linux and perhaps other systems, you can avoid the need to run the server as `root` by changing the `limits.conf` file. See the notes regarding the memlock limit in Section 8.11.4.2, “Enabling Large Page Support”.

You must not try to use this option on a system that does not support the `mlockall()` system call; if you do so, `mysqld` will very likely crash as soon as you try to start it.

- `--myisam-block-size=N`

Command-Line Format	`--myisam-block-size=#`	
Option-File Format	`myisam-block-size`	
	Permitted Values	
	Type	`numeric`
	Default	`1024`
	Range	`1024 .. 16384`

The block size to be used for `MyISAM` index pages.

- `--myisam-recover-options[=option[,option]...]]`

Command-Line Format	`--myisam-recover-options[=name]`	
Option-File Format	`myisam-recover-options`	
	Permitted Values	

	Type	enumeration
	Default	OFF
	Valid Values	OFF
		DEFAULT
		BACKUP
		FORCE
		QUICK

Set the MyISAM storage engine recovery mode. The option value is any combination of the values of OFF, DEFAULT, BACKUP, FORCE, or QUICK. If you specify multiple values, separate them by commas. Specifying the option with no argument is the same as specifying DEFAULT, and specifying with an explicit value of "" disables recovery (same as a value of OFF). If recovery is enabled, each time mysqld opens a MyISAM table, it checks whether the table is marked as crashed or was not closed properly. (The last option works only if you are running with external locking disabled.) If this is the case, mysqld runs a check on the table. If the table was corrupted, mysqld attempts to repair it.

The following options affect how the repair works.

Option	Description
OFF	No recovery.
DEFAULT	Recovery without backup, forcing, or quick checking.
BACKUP	If the data file was changed during recovery, save a backup of the *tbl_name*.MYD file as *tbl_name-datetime*.BAK.
FORCE	Run recovery even if we would lose more than one row from the .MYD file.
QUICK	Do not check the rows in the table if there are not any delete blocks.

Before the server automatically repairs a table, it writes a note about the repair to the error log. If you want to be able to recover from most problems without user intervention, you should use the options BACKUP,FORCE. This forces a repair of a table even if some rows would be deleted, but it keeps the old data file as a backup so that you can later examine what happened.

See Section 14.3.1, “MyISAM Startup Options”.

- --no-defaults

  Do not read any option files. If program startup fails due to reading unknown options from an option file, --no-defaults can be used to prevent them from being read.

  The exception is that the .mylogin.cnf file, if it exists, is read in all cases. This permits passwords to be specified in a safer way than on the command line even when --no-defaults is used. (.mylogin.cnf is created by the mysql_config_editor utility. See Section 4.6.6, “mysql_config_editor — MySQL Configuration Utility”.)

- --old-alter-table

Command-Line Format	--old-alter-table
Option-File Format	old-alter-table
System Variable Name	old_alter_table
Variable Scope	Global, Session

Dynamic Variable	Yes	
	Permitted Values	
	Type	boolean
	Default	OFF

When this option is given, the server does not use the optimized method of processing an ALTER TABLE operation. It reverts to using a temporary table, copying over the data, and then renaming the temporary table to the original, as used by MySQL 5.0 and earlier. For more information on the operation of ALTER TABLE, see Section 13.1.6, “ALTER TABLE Syntax”.

- --old-style-user-limits

Command-Line Format	--old-style-user-limits	
Option-File Format	old-style-user-limits	
	Permitted Values	
	Type	boolean
	Default	FALSE

Enable old-style user limits. (Before MySQL 5.0.3, account resource limits were counted separately for each host from which a user connected rather than per account row in the user table.) See Section 6.3.4, “Setting Account Resource Limits”.

- --open-files-limit=*count*

Command-Line Format	--open-files-limit=#	
Option-File Format	open-files-limit	
System Variable Name	open_files_limit	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	numeric
	Default	(autosized)
	Range	0 .. platform dependent

Changes the number of file descriptors available to mysqld. You should try increasing the value of this option if mysqld gives you the error Too many open files. mysqld uses the option value to reserve descriptors with setrlimit(). Internally, the maximum value for this option is the maximum unsigned integer value, but the actual maximum is platform dependent. If the requested number of file descriptors cannot be allocated, mysqld writes a warning to the error log.

mysqld may attempt to allocate more than the requested number of descriptors (if they are available), using the values of max_connections and table_open_cache to estimate whether more descriptors will be needed.

On Unix, the value cannot be set less than ulimit -n.

- --partition[=*value*]

Command-Line Format	--partition

Option-File Format	`partition`	
Disabled by	`skip-partition`	
	Permitted Values	
	Type	`boolean`
	Default	`ON`

Enables or disables user-defined partitioning support in the MySQL Server.

- `--performance-schema-xxx`

  Configure a Performance Schema option. For details, see Section 20.11, “Performance Schema Command Options”.

- `--pid-file=`*`path`*

Command-Line Format	`--pid-file=file_name`	
Option-File Format	`pid-file`	
System Variable Name	`pid_file`	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	`file name`

  The path name of the process ID file. The server creates the file in the data directory unless an absolute path name is given to specify a different directory. This file is used by other programs such as `mysqld_safe` to determine the server's process ID.

- `--plugin-`*`xxx`*

  Specifies an option that pertains to a server plugin. For example, many storage engines can be built as plugins, and for such engines, options for them can be specified with a `--plugin` prefix. Thus, the `--innodb_file_per_table` option for `InnoDB` can be specified as `--plugin-innodb_file_per_table`.

  For boolean options that can be enabled or disabled, the `--skip` prefix and other alternative formats are supported as well (see Section 4.2.3.2, “Program Option Modifiers”). For example, `--skip-plugin-innodb_file_per_table` disables `innodb_file_per_table`.

  The rationale for the `--plugin` prefix is that it enables plugin options to be specified unambiguously if there is a name conflict with a built-in server option. For example, were a plugin writer to name a plugin “sql” and implement a “mode” option, the option name might be `--sql-mode`, which would conflict with the built-in option of the same name. In such cases, references to the conflicting name are resolved in favor of the built-in option. To avoid the ambiguity, users can specify the plugin option as `--plugin-sql-mode`. Use of the `--plugin` prefix for plugin options is recommended to avoid any question of ambiguity.

- `--plugin-load=`*`plugin_list`*

Command-Line Format	`--plugin-load=plugin_list`	
Option-File Format	`plugin-load`	
	Permitted Values	

	Type	`string`

This option tells the server to load the named plugins at startup. The option value is a semicolon-separated list of *`name=plugin_library`* pairs. Each *`name`* is the name of the plugin, and *`plugin_library`* is the name of the shared library that contains the plugin code. Each library file must be located in the directory named by the `plugin_dir` system variable. For example, if plugins named `myplug1` and `myplug2` have library files `myplug1.so` and `myplug2.so`, use this option to load them at startup:

```
shell> mysqld --plugin-load="myplug1=myplug1.so;myplug2=myplug2.so"
```

Quotes are used around the argument value here because semicolon (`;`) is interpreted as a special character by some command interpreters. (Unix shells treat it as a command terminator, for example.)

If multiple `--plugin-load` options are given, only the last one is used. Additional plugins to load may be specified using `--plugin-load-add` options.

If a plugin library is named without any preceding plugin name, the server loads all plugins in the library.

Each plugin is loaded for a single invocation of `mysqld` only. After a restart, the plugin is not loaded unless `--plugin-load` is used again. This is in contrast to `INSTALL PLUGIN`, which adds an entry to the `mysql.plugins` table to cause the plugin to be loaded for every normal server startup.

Under normal startup, the server determines which plugins to load by reading the `mysql.plugins` system table. If the server is started with the `--skip-grant-tables` option, it does not consult the `mysql.plugins` table and does not load plugins listed there. `--plugin-load` enables plugins to be loaded even when `--skip-grant-tables` is given. `--plugin-load` also enables plugins to be loaded at startup under configurations when plugins cannot be loaded at runtime.

For additional information about plugin loading, see Section 5.1.8.1, “Installing and Uninstalling Plugins”.

- `--plugin-load-add=`*`plugin_list`*

Command-Line Format	`--plugin-load-add=plugin_list`	
Option-File Format	`plugin-load-add`	
	Permitted Values	
	Type	`string`

This option complements the `--plugin-load` option. `--plugin-load-add` adds a plugin or plugins to the set of plugins to be loaded at startup. The argument format is the same as for `--plugin-load`. `--plugin-load-add` can be used to avoid specifying a large set of plugins as a single long unwieldy `--plugin-load` argument.

`--plugin-load-add` can be given in the absence of `--plugin-load`, but any instance of `--plugin-load-add` that appears before `--plugin-load`. has no effect because `--plugin-load` resets the set of plugins to load. In other words, these options:

```
--plugin-load=x --plugin-load-add=y
```

are equivalent to this option:

```
--plugin-load="x;y"
```

But these options:

```
--plugin-load-add=y --plugin-load=x
```

are equivalent to this option:

```
--plugin-load=x
```

For additional information about plugin loading, see Section 5.1.8.1, “Installing and Uninstalling Plugins”.

- `--port=port_num`, `-P port_num`

Command-Line Format	`--port=#`	
	`-P`	
Option-File Format	`port`	
System Variable Name	`port`	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	`numeric`
	Default	`3306`
	Range	`0 .. 65535`

The port number to use when listening for TCP/IP connections. The port number must be 1024 or higher unless the server is started by the `root` system user.

- `--port-open-timeout=num`

Command-Line Format	`--port-open-timeout=#`	
Option-File Format	`port-open-timeout`	
	Permitted Values	
	Type	`numeric`
	Default	`0`

On some systems, when the server is stopped, the TCP/IP port might not become available immediately. If the server is restarted quickly afterward, its attempt to reopen the port can fail. This option indicates how many seconds the server should wait for the TCP/IP port to become free if it cannot be opened. The default is not to wait.

- `--print-defaults`

Print the program name and all options that it gets from option files.

- `--remove [service_name]`

Command-Line Format	`--remove [service_name]`

(Windows only) Remove a MySQL Windows service. The default service name is `MySQL` if no *`service_name`* value is given. For more information, see Section 2.3.5.7, “Starting MySQL as a Windows Service”.

- `--safe-user-create`

Command-Line Format	`--safe-user-create`	
Option-File Format	`safe-user-create`	
	Permitted Values	
	Type	`boolean`
	Default	`FALSE`

If this option is enabled, a user cannot create new MySQL users by using the `GRANT` statement unless the user has the `INSERT` privilege for the `mysql.user` table or any column in the table. If you want a user to have the ability to create new users that have those privileges that the user has the right to grant, you should grant the user the following privilege:

```
GRANT INSERT(user) ON mysql.user TO 'user_name'@'host_name';
```

This ensures that the user cannot change any privilege columns directly, but has to use the `GRANT` statement to give privileges to other users.

- `--secure-auth`

Command-Line Format	`--secure-auth`	
Option-File Format	`secure-auth`	
System Variable Name	`secure_auth`	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	`boolean`
	Default	`ON`

This option causes the server to block connections by clients that attempt to use accounts that have passwords stored in the old (pre-4.1) format. Use it to prevent all use of passwords employing the old format (and hence insecure communication over the network). This option is enabled by default; to disable it, use `--skip-secure-auth`.

Server startup fails with an error if this option is enabled and the privilege tables are in pre-4.1 format. See Section C.5.2.4, “`Client does not support authentication protocol`”.

The `mysql` client also has a `--secure-auth` option, which prevents connections to a server if the server requires a password in old format for the client account.

Note

Passwords that use the pre-4.1 hashing method are less secure than passwords that use the native password hashing method and should be avoided. Pre-4.1 passwords are deprecated and support for them will be removed in a future MySQL release.

- `--secure-file-priv=`*`path`*

Command-Line Format	`--secure-file-priv=path`	
Option-File Format	`secure-file-priv`	
System Variable Name	`secure_file_priv`	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	`string`

  This option limits the effect of the `LOAD_FILE()` function and the `LOAD DATA` and `SELECT ... INTO OUTFILE` statements to work only with files in the specified directory.

- `--shared-memory`

System Variable Name	`shared_memory`
Variable Scope	Global
Dynamic Variable	No
Platform Specific	Windows

  Enable shared-memory connections by local clients. This option is available only on Windows.

- `--shared-memory-base-name=`*`name`*

System Variable Name	`shared_memory_base_name`
Variable Scope	Global
Dynamic Variable	No
Platform Specific	Windows

  The name of shared memory to use for shared-memory connections. This option is available only on Windows. The default name is `MYSQL`. The name is case sensitive.

- `--skip-concurrent-insert`

  Turn off the ability to select and insert at the same time on `MyISAM` tables. (This is to be used only if you think you have found a bug in this feature.) See Section 8.10.3, “Concurrent Inserts”.

- `--skip-event-scheduler`

Command-Line Format	`--skip-event-scheduler`
	`--disable-event-scheduler`
Option-File Format	`skip-event-scheduler`

  Turns the Event Scheduler `OFF`. This is not the same as disabling the Event Scheduler, which requires setting `--event-scheduler=DISABLED`; see The `--event-scheduler` Option [2386], for more information.

- `--skip-grant-tables`

This option causes the server to start without using the privilege system at all, which gives anyone with access to the server *unrestricted access to all databases*. You can cause a running server to start using the grant tables again by executing `mysqladmin flush-privileges` or `mysqladmin reload` command from a system shell, or by issuing a MySQL `FLUSH PRIVILEGES` statement after connecting to the server. This option also suppresses loading of plugins that were installed with the `INSTALL PLUGIN` statement, user-defined functions (UDFs), and scheduled events. To cause plugins to be loaded anyway, use the `--plugin-load` option.

Note that `FLUSH PRIVILEGES` might be executed implicitly by other actions performed after startup. For example, `mysql_upgrade` flushes the privileges during the upgrade procedure.

- `--skip-host-cache`

  Disable use of the internal host cache for faster name-to-IP resolution. In this case, the server performs a DNS lookup every time a client connects. See Section 8.11.5.2, “DNS Lookup Optimization and the Host Cache”.

  Use of `--skip-host-cache` is similar to setting the `host_cache_size` system variable to 0, but `host_cache_size` is more flexible because it can also be used to resize, enable, or disable the host cache at runtime, not just at server startup.

  If you start the server with `--skip-host-cache`, that does not prevent changes to the value of `host_cache_size`, but such changes have no effect and the cache is not re-enabled even if `host_cache_size` is set larger than 0.

- `--skip-innodb`

  Disable the `InnoDB` storage engine. In this case, because the default storage engine is `InnoDB`, the server will not start unless you also use `--default-storage-engine` and `--default-tmp-storage-engine` to set the default to some other engine for both permanent and `TEMPORARY` tables.

- `--skip-name-resolve`

  Do not resolve host names when checking client connections. Use only IP addresses. If you use this option, all `Host` column values in the grant tables must be IP addresses or `localhost`. See Section 8.11.5.2, “DNS Lookup Optimization and the Host Cache”.

  Depending on the network configuration of your system and the `Host` values for your accounts, clients may need to connect using an explicit `--host` option, such as `--host=localhost`, `--host=127.0.0.1`, or `--host=::1`.

- `--skip-networking`

  Do not listen for TCP/IP connections at all. All interaction with `mysqld` must be made using named pipes or shared memory (on Windows) or Unix socket files (on Unix). This option is highly recommended for systems where only local clients are permitted. See Section 8.11.5.2, “DNS Lookup Optimization and the Host Cache”.

- `--skip-partition`

Command-Line Format	`--skip-partition`
	`--disable-partition`
Option-File Format	`skip-partition`

Disables user-defined partitioning. Partitioned tables can be seen using `SHOW TABLES` or by querying the `INFORMATION_SCHEMA.TABLES` table, but cannot be created or modified, nor can data in such tables be accessed. All partition-specific columns in the `INFORMATION_SCHEMA.PARTITIONS` table display `NULL`.

Since `DROP TABLE` removes table definition (`.frm`) files, this statement works on partitioned tables even when partitioning is disabled using the option. The statement, however, does not remove `.par` files associated with partitioned tables in such cases. For this reason, you should avoid dropping partitioned tables with partitioning disabled, or take action to remove the orphaned `.par` files manually.

- `--ssl*`

  Options that begin with `--ssl` specify whether to permit clients to connect using SSL and indicate where to find SSL keys and certificates. See Section 6.3.11.4, “SSL Command Options”.

- `--standalone`

Command-Line Format	`--standalone`
Option-File Format	`standalone`
Platform Specific	Windows

  Available on Windows only; instructs the MySQL server not to run as a service.

- `--super-large-pages`

Command-Line Format	`--super-large-pages`	
Option-File Format	`super-large-pages`	
Platform Specific	Solaris	
	Permitted Values	
	Type (Solaris)	`boolean`
	Default	`FALSE`

  Standard use of large pages in MySQL attempts to use the largest size supported, up to 4MB. Under Solaris, a “super large pages” feature enables uses of pages up to 256MB. This feature is available for recent SPARC platforms. It can be enabled or disabled by using the `--super-large-pages` or `--skip-super-large-pages` option.

- `--symbolic-links`, `--skip-symbolic-links`

Command-Line Format	`--symbolic-links`
Option-File Format	`symbolic-links`

  Enable or disable symbolic link support. On Unix, enabling symbolic links means that you can link a `MyISAM` index file or data file to another directory with the `INDEX DIRECTORY` or `DATA DIRECTORY` options of the `CREATE TABLE` statement. If you delete or rename the table, the files that its symbolic links point to also are deleted or renamed. See Using Symbolic Links for `MyISAM` Tables on Unix.

  This option has no meaning on Windows.

- `--skip-show-database`

Command-Line Format	`--skip-show-database`
Option-File Format	`skip-show-database`
System Variable Name	`skip_show_database`
Variable Scope	Global
Dynamic Variable	No

This option sets the `skip_show_database` system variable that controls who is permitted to use the `SHOW DATABASES` statement. See Section 5.1.4, “Server System Variables”.

- `--skip-stack-trace`

Command-Line Format	`--skip-stack-trace`
Option-File Format	`skip-stack-trace`

Do not write stack traces. This option is useful when you are running `mysqld` under a debugger. On some systems, you also must use this option to get a core file. See Section 22.4, “Debugging and Porting MySQL”.

- `--slow-query-log[={0|1}]`

Command-Line Format	`--slow-query-log`	
Option-File Format	`slow-query-log`	
System Variable Name	`slow_query_log`	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	`boolean`
	Default	`OFF`

Specify the initial slow query log state. With no argument or an argument of 1, the `--slow-query-log` option enables the log. If omitted or given with an argument of 0, the option disables the log.

- `--slow-start-timeout=`*`timeout`*

Command-Line Format	`--slow-start-timeout=#`	
Option-File Format	`slow-start-timeout`	
	Permitted Values	
	Type (Windows)	`numeric`
	Default	`15000`

This option controls the Windows service control manager's service start timeout. The value is the maximum number of milliseconds that the service control manager waits before trying to kill the windows service during startup. The default value is 15000 (15 seconds). If the MySQL service takes too long to start, you may need to increase this value. A value of 0 means there is no timeout.

- `--socket=`*`path`*

Command-Line Format	`--socket=name`	
Option-File Format	`socket`	
System Variable Name	`socket`	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	`file name`
	Default	`/tmp/mysql.sock`

On Unix, this option specifies the Unix socket file to use when listening for local connections. The default value is `/tmp/mysql.sock`. If this option is given, the server creates the file in the data directory unless an absolute path name is given to specify a different directory. On Windows, the option specifies the pipe name to use when listening for local connections that use a named pipe. The default value is `MySQL` (not case sensitive).

- `--sql-mode=`*`value`*`[,`*`value`*`[,`*`value`*`...]]`

Command-Line Format	`--sql-mode=name`	
Option-File Format	`sql-mode`	
System Variable Name	`sql_mode`	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	`set`
	Default	`NO_ENGINE_SUBSTITUTION`
	Valid Values	`ALLOW_INVALID_DATES`
		`ANSI_QUOTES`
		`ERROR_FOR_DIVISION_BY_ZERO`
		`HIGH_NOT_PRECEDENCE`
		`IGNORE_SPACE`
		`NO_AUTO_CREATE_USER`
		`NO_AUTO_VALUE_ON_ZERO`
		`NO_BACKSLASH_ESCAPES`
		`NO_DIR_IN_CREATE`
		`NO_ENGINE_SUBSTITUTION`
		`NO_FIELD_OPTIONS`
		`NO_KEY_OPTIONS`
		`NO_TABLE_OPTIONS`
		`NO_UNSIGNED_SUBTRACTION`
		`NO_ZERO_DATE`
		`NO_ZERO_IN_DATE`

		ONLY_FULL_GROUP_BY
		PAD_CHAR_TO_FULL_LENGTH
		PIPES_AS_CONCAT
		REAL_AS_FLOAT
		STRICT_ALL_TABLES
		STRICT_TRANS_TABLES

Set the SQL mode. See Section 5.1.7, “Server SQL Modes”.

> **Note**
>
> MySQL installation programs may configure the SQL mode during the installation process. For example, `mysql_install_db` creates a default option file named `my.cnf` in the base installation directory. This file contains a line that sets the SQL mode; see Section 4.4.3, “`mysql_install_db` — Initialize MySQL Data Directory”.
>
> If the SQL mode differs from the default or from what you expect, check for a setting in an option file that the server reads at startup.

- `--sysdate-is-now`

Command-Line Format	`--sysdate-is-now`	
Option-File Format	`sysdate-is-now`	
	Permitted Values	
	Type	`boolean`
	Default	`FALSE`

`SYSDATE()` by default returns the time at which it executes, not the time at which the statement in which it occurs begins executing. This differs from the behavior of `NOW()`. This option causes `SYSDATE()` to be an alias for `NOW()`. For information about the implications for binary logging and replication, see the description for `SYSDATE()` in Section 12.7, “Date and Time Functions” and for `SET TIMESTAMP` in Section 5.1.4, “Server System Variables”.

- `--tc-heuristic-recover={COMMIT|ROLLBACK}`

Command-Line Format	`--tc-heuristic-recover=name`	
Option-File Format	`tc-heuristic-recover`	
	Permitted Values	
	Type	`enumeration`
	Default	`COMMIT`
	Valid Values	`COMMIT`
		`ROLLBACK`

The type of decision to use in the heuristic recovery process. Currently, this option is unused.

- `--temp-pool`

Command-Line Format	`--temp-pool`

Option-File Format	`temp-pool`	
	Permitted Values	
	Type	`boolean`
	Default	`TRUE`

This option causes most temporary files created by the server to use a small set of names, rather than a unique name for each new file. This works around a problem in the Linux kernel dealing with creating many new files with different names. With the old behavior, Linux seems to “leak” memory, because it is being allocated to the directory entry cache rather than to the disk cache. This option is ignored except on Linux.

- `--transaction-isolation=`*`level`*

Command-Line Format	`--transaction-isolation=name`	
Option-File Format	`transaction-isolation`	
	Permitted Values	
	Type	`enumeration`
	Default	`REPEATABLE-READ`
	Valid Values	`READ-UNCOMMITTED`
		`READ-COMMITTED`
		`REPEATABLE-READ`
		`SERIALIZABLE`

Sets the default transaction isolation level. The `level` value can be `READ-UNCOMMITTED`, `READ-COMMITTED`, `REPEATABLE-READ`, or `SERIALIZABLE`. See Section 13.3.6, “`SET TRANSACTION` Syntax”.

The default transaction isolation level can also be set at runtime using the `SET TRANSACTION` statement or by setting the `tx_isolation` system variable.

- `--transaction-read-only`

Command-Line Format	`--transaction-read-only`	
Option-File Format	`transaction-read-only`	
	Permitted Values	
	Type	`boolean`
	Default	`OFF`

Sets the default transaction access mode. By default, read-only mode is disabled, so the mode is read/write.

To set the default transaction access mode at runtime, use the `SET TRANSACTION` statement or set the `tx_read_only` system variable. See Section 13.3.6, “`SET TRANSACTION` Syntax”.

- `--tmpdir=`*`path`*, `-t` *`path`*

Command-Line Format	`--tmpdir=path`
	`-t`

Option-File Format	tmpdir	
System Variable Name	tmpdir	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	directory name

The path of the directory to use for creating temporary files. It might be useful if your default /tmp directory resides on a partition that is too small to hold temporary tables. This option accepts several paths that are used in round-robin fashion. Paths should be separated by colon characters (":") on Unix and semicolon characters (";") on Windows. If the MySQL server is acting as a replication slave, you should not set --tmpdir to point to a directory on a memory-based file system or to a directory that is cleared when the server host restarts. For more information about the storage location of temporary files, see Section C.5.4.4, "Where MySQL Stores Temporary Files". A replication slave needs some of its temporary files to survive a machine restart so that it can replicate temporary tables or LOAD DATA INFILE operations. If files in the temporary file directory are lost when the server restarts, replication fails.

- --user={*user_name*|*user_id*}, -u {*user_name*|*user_id*}

Command-Line Format	--user=name	
	-u name	
Option-File Format	user	
	Permitted Values	
	Type	string

Run the mysqld server as the user having the name *user_name* or the numeric user ID *user_id*. ("User" in this context refers to a system login account, not a MySQL user listed in the grant tables.)

This option is *mandatory* when starting mysqld as root. The server changes its user ID during its startup sequence, causing it to run as that particular user rather than as root. See Section 6.1.1, "Security Guidelines".

To avoid a possible security hole where a user adds a --user=root option to a my.cnf file (thus causing the server to run as root), mysqld uses only the first --user option specified and produces a warning if there are multiple --user options. Options in /etc/my.cnf and $MYSQL_HOME/my.cnf are processed before command-line options, so it is recommended that you put a --user option in /etc/my.cnf and specify a value other than root. The option in /etc/my.cnf is found before any other --user options, which ensures that the server runs as a user other than root, and that a warning results if any other --user option is found.

- --verbose, -v

  Use this option with the --help option for detailed help.

- --version, -V

  Display version information and exit.

You can assign a value to a server system variable by using an option of the form --*var_name*=*value*. For example, --key_buffer_size=32M sets the key_buffer_size variable to a value of 32MB.

Note that when you assign a value to a variable, MySQL might automatically correct the value to stay within a given range, or adjust the value to the closest permissible value if only certain values are permitted.

If you want to restrict the maximum value to which a variable can be set at runtime with `SET`, you can define this by using the `--maximum-var_name=value` command-line option.

You can change the values of most system variables for a running server with the `SET` statement. See Section 13.7.4, "SET Syntax".

Section 5.1.4, "Server System Variables", provides a full description for all variables, and additional information for setting them at server startup and runtime. Section 8.11.2, "Tuning Server Parameters", includes information on optimizing the server by tuning system variables.

5.1.4 Server System Variables

The MySQL server maintains many system variables that indicate how it is configured. Each system variable has a default value. System variables can be set at server startup using options on the command line or in an option file. Most of them can be changed dynamically while the server is running by means of the `SET` statement, which enables you to modify operation of the server without having to stop and restart it. You can refer to system variable values in expressions.

There are several ways to see the names and values of system variables:

- To see the values that a server will use based on its compiled-in defaults and any option files that it reads, use this command:

```
mysqld --verbose --help
```

- To see the values that a server will use based on its compiled-in defaults, ignoring the settings in any option files, use this command:

```
mysqld --no-defaults --verbose --help
```

- To see the current values used by a running server, use the `SHOW VARIABLES` statement.

This section provides a description of each system variable. Variables with no version indicated are present in all MySQL 5.7 releases. For historical information concerning their implementation, please see http://dev.mysql.com/doc/refman/5.0/en/, and http://dev.mysql.com/doc/refman/4.1/en/.

The following table lists all available system variables.

Table 5.2 System Variable Summary

Name	Cmd-Line	Option file	System Var	Var Scope	Dynamic
audit_log_format			Yes	Global	No
auto_increment_increment			Yes	Both	Yes
auto_increment_offset			Yes	Both	Yes
autocommit	Yes	Yes	Yes	Both	Yes
automatic_sp_privileges			Yes	Global	Yes
back_log			Yes	Global	No
basedir	Yes	Yes	Yes	Global	No
big-tables	Yes	Yes			Yes

Name	Cmd-Line	Option file	System Var	Var Scope	Dynamic
- *Variable*: big_tables			Yes	Both	Yes
bind-address	Yes	Yes			No
- *Variable*: bind_address			Yes	Global	No
binlog_cache_size	Yes	Yes	Yes	Global	Yes
binlog_checksum			Yes	Global	Yes
binlog_direct_non_transactional_updates	Yes	Yes	Yes	Both	Yes
binlog-format	Yes	Yes			Yes
- *Variable*: binlog_format			Yes	Both	Yes
binlog_max_flush_queue_time			Yes	Global	Yes
binlog_order_commits			Yes	Global	Yes
binlog_row_image	Yes	Yes	Yes	Both	Yes
binlog_rows_query_log_events			Yes	Both	Yes
binlog_stmt_cache_size	Yes	Yes	Yes	Global	Yes
block_encryption_mode	Yes	Yes	Yes	Both	Yes
bulk_insert_buffer_size	Yes	Yes	Yes	Both	Yes
character_set_client			Yes	Both	Yes
character_set_connection			Yes	Both	Yes
character_set_database[a]			Yes	Both	Yes
character-set-filesystem	Yes	Yes			Yes
- *Variable*: character_set_filesystem			Yes	Both	Yes
character_set_results			Yes	Both	Yes
character-set-server	Yes	Yes			Yes
- *Variable*: character_set_server			Yes	Both	Yes
character_set_system			Yes	Global	No
character-sets-dir	Yes	Yes			No
- *Variable*: character_sets_dir			Yes	Global	No
collation_connection			Yes	Both	Yes
collation_database[b]			Yes	Both	Yes
collation-server	Yes	Yes			Yes
- *Variable*: collation_server			Yes	Both	Yes
completion_type	Yes	Yes	Yes	Both	Yes
concurrent_insert	Yes	Yes	Yes	Global	Yes
connect_timeout	Yes	Yes	Yes	Global	Yes

Name	Cmd-Line	Option file	System Var	Var Scope	Dynamic
core_file			Yes	Global	No
daemon_memcached_enable_binlog	Yes	Yes	Yes	Global	No
daemon_memcached_engine_lib_name	Yes	Yes	Yes	Global	No
daemon_memcached_engine_lib_path	Yes	Yes	Yes	Global	No
daemon_memcached_option	Yes	Yes	Yes	Global	No
daemon_memcached_r_batch_size	Yes	Yes	Yes	Global	No
daemon_memcached_w_batch_size	Yes	Yes	Yes	Global	No
datadir	Yes	Yes	Yes	Global	No
date_format			Yes	Global	No
datetime_format			Yes	Global	No
debug	Yes	Yes	Yes	Both	Yes
debug_sync			Yes	Session	Yes
default_authentication_plugin	Yes	Yes	Yes	Global	No
default_password_lifetime	Yes	Yes	Yes	Global	Yes
default-storage-engine	Yes	Yes			Yes
- *Variable*: default_storage_engine			Yes	Both	Yes
default_tmp_storage_engine	Yes	Yes	Yes	Both	Yes
default_week_format	Yes	Yes	Yes	Both	Yes
delay-key-write	Yes	Yes			Yes
- *Variable*: delay_key_write			Yes	Global	Yes
delayed_insert_limit	Yes	Yes	Yes	Global	Yes
delayed_insert_timeout	Yes	Yes	Yes	Global	Yes
delayed_queue_size	Yes	Yes	Yes	Global	Yes
disconnect_on_expired_password	Yes	Yes	Yes	Session	No
div_precision_increment	Yes	Yes	Yes	Both	Yes
end_markers_in_json			Yes	Both	Yes
enforce_gtid_consistency	Yes	Yes	Yes	Global	No
enforce-gtid-consistency	Yes	Yes	Yes	Global	No
eq_range_index_dive_limit			Yes	Both	Yes
error_count			Yes	Session	No
event-scheduler	Yes	Yes			Yes
- *Variable*: event_scheduler			Yes	Global	Yes
expire_logs_days	Yes	Yes	Yes	Global	Yes
explicit_defaults_for_timestamp	Yes	Yes	Yes	Session	No

Name	Cmd-Line	Option file	System Var	Var Scope	Dynamic
external_user			Yes	Session	No
flush	Yes	Yes	Yes	Global	Yes
flush_time	Yes	Yes	Yes	Global	Yes
foreign_key_checks			Yes	Both	Yes
ft_boolean_syntax	Yes	Yes	Yes	Global	Yes
ft_max_word_len	Yes	Yes	Yes	Global	No
ft_min_word_len	Yes	Yes	Yes	Global	No
ft_query_expansion_limit	Yes	Yes	Yes	Global	No
ft_stopword_file	Yes	Yes	Yes	Global	No
general-log	Yes	Yes			Yes
- *Variable*: general_log			Yes	Global	Yes
general_log_file	Yes	Yes	Yes	Global	Yes
group_concat_max_len	Yes	Yes	Yes	Both	Yes
gtid_executed			Yes	Both	No
gtid_mode			Yes	Global	No
gtid-mode	Yes	Yes			No
- *Variable*: gtid_mode			Yes	Global	No
gtid_next			Yes	Session	Yes
gtid_owned			Yes	Both	No
gtid_purged			Yes	Global	Yes
have_compress			Yes	Global	No
have_crypt			Yes	Global	No
have_dynamic_loading			Yes	Global	No
have_geometry			Yes	Global	No
have_openssl			Yes	Global	No
have_profiling			Yes	Global	No
have_query_cache			Yes	Global	No
have_rtree_keys			Yes	Global	No
have_ssl			Yes	Global	No
have_symlink			Yes	Global	No
host_cache_size			Yes	Global	Yes
hostname			Yes	Global	No
identity			Yes	Session	Yes
ignore-builtin-innodb	Yes	Yes			No
- *Variable*: ignore_builtin_innodb			Yes	Global	No
ignore_db_dirs			Yes	Global	No

Name	Cmd-Line	Option file	System Var	Var Scope	Dynamic
init_connect	Yes	Yes	Yes	Global	Yes
init-file	Yes	Yes			No
- *Variable*: init_file			Yes	Global	No
init_slave	Yes	Yes	Yes	Global	Yes
innodb_adaptive_flushing	Yes	Yes	Yes	Global	Yes
innodb_adaptive_flushing_lwm	Yes	Yes	Yes	Global	Yes
innodb_adaptive_hash_index	Yes	Yes	Yes	Global	Yes
innodb_adaptive_max_sleep_delay	Yes	Yes	Yes	Global	Yes
innodb_additional_mem_pool_size	Yes	Yes	Yes	Global	No
innodb_api_bk_commit_interval	Yes	Yes	Yes	Global	Yes
innodb_api_disable_rowlock	Yes	Yes	Yes	Global	No
innodb_api_enable_binlog	Yes	Yes	Yes	Global	No
innodb_api_enable_mdl	Yes	Yes	Yes	Global	No
innodb_api_trx_level	Yes	Yes	Yes	Global	Yes
innodb_autoextend_increment	Yes	Yes	Yes	Global	Yes
innodb_autoinc_lock_mode	Yes	Yes	Yes	Global	No
innodb_buffer_pool_dump_at_shutdown	Yes	Yes	Yes	Global	Yes
innodb_buffer_pool_dump_now	Yes	Yes	Yes	Global	Yes
innodb_buffer_pool_dump_pct	Yes	Yes	Yes	Global	Yes
innodb_buffer_pool_filename	Yes	Yes	Yes	Global	Yes
innodb_buffer_pool_instances	Yes	Yes	Yes	Global	No
innodb_buffer_pool_load_abort	Yes	Yes	Yes	Global	Yes
innodb_buffer_pool_load_at_startup	Yes	Yes	Yes	Global	No
innodb_buffer_pool_load_now	Yes	Yes	Yes	Global	Yes
innodb_buffer_pool_size	Yes	Yes	Yes	Global	No
innodb_change_buffer_max_size	Yes	Yes	Yes	Global	Yes
innodb_change_buffering	Yes	Yes	Yes	Global	Yes
innodb_checksum_algorithm	Yes	Yes	Yes	Global	Yes
innodb_checksums	Yes	Yes	Yes	Global	No
innodb_cmp_per_index_enabled	Yes	Yes	Yes	Global	Yes
innodb_commit_concurrency	Yes	Yes	Yes	Global	Yes
innodb_compression_failure_threshold_pct	Yes	Yes	Yes	Global	Yes
innodb_compression_level	Yes	Yes	Yes	Global	Yes
innodb_compression_pad_pct_max	Yes	Yes	Yes	Global	Yes
innodb_concurrency_tickets	Yes	Yes	Yes	Global	Yes
innodb_data_file_path	Yes	Yes	Yes	Global	No
innodb_data_home_dir	Yes	Yes	Yes	Global	No
innodb_disable_sort_file_cache	Yes	Yes	Yes	Global	Yes

Name	Cmd-Line	Option file	System Var	Var Scope	Dynamic
innodb_doublewrite	Yes	Yes	Yes	Global	No
innodb_fast_shutdown	Yes	Yes	Yes	Global	Yes
innodb_file_format	Yes	Yes	Yes	Global	Yes
innodb_file_format_check	Yes	Yes	Yes	Global	No
innodb_file_format_max	Yes	Yes	Yes	Global	Yes
innodb_file_per_table	Yes	Yes	Yes	Global	Yes
innodb_flush_log_at_timeout			Yes	Global	Yes
innodb_flush_log_at_trx_commit	Yes	Yes	Yes	Global	Yes
innodb_flush_method	Yes	Yes	Yes	Global	No
innodb_flush_neighbors	Yes	Yes	Yes	Global	Yes
innodb_flushing_avg_loops	Yes	Yes	Yes	Global	Yes
innodb_force_load_corrupted	Yes	Yes	Yes	Global	No
innodb_force_recovery	Yes	Yes	Yes	Global	No
innodb_ft_aux_table	Yes	Yes	Yes	Global	Yes
innodb_ft_cache_size	Yes	Yes	Yes	Global	No
innodb_ft_enable_diag_print	Yes	Yes	Yes	Global	Yes
innodb_ft_enable_stopword	Yes	Yes	Yes	Global	Yes
innodb_ft_max_token_size	Yes	Yes	Yes	Global	No
innodb_ft_min_token_size	Yes	Yes	Yes	Global	No
innodb_ft_num_word_optimize	Yes	Yes	Yes	Global	Yes
innodb_ft_result_cache_limit	Yes	Yes	Yes	Global	Yes
innodb_ft_server_stopword_table	Yes	Yes	Yes	Global	Yes
innodb_ft_sort_pll_degree	Yes	Yes	Yes	Global	No
innodb_ft_total_cache_size	Yes	Yes	Yes	Global	No
innodb_ft_user_stopword_table	Yes	Yes	Yes	Both	Yes
innodb_io_capacity	Yes	Yes	Yes	Global	Yes
innodb_io_capacity_max	Yes	Yes	Yes	Global	Yes
innodb_large_prefix	Yes	Yes	Yes	Global	Yes
innodb_lock_wait_timeout	Yes	Yes	Yes	Both	Yes
innodb_locks_unsafe_for_binlog	Yes	Yes	Yes	Global	No
innodb_log_buffer_size	Yes	Yes	Yes	Global	No
innodb_log_compressed_pages	Yes	Yes	Yes	Global	Yes
innodb_log_file_size	Yes	Yes	Yes	Global	No
innodb_log_files_in_group	Yes	Yes	Yes	Global	No
innodb_log_group_home_dir	Yes	Yes	Yes	Global	No
innodb_log_write_ahead_size	Yes	Yes	Yes	Global	Yes
innodb_lru_scan_depth	Yes	Yes	Yes	Global	Yes
innodb_max_dirty_pages_pct	Yes	Yes	Yes	Global	Yes

Name	Cmd-Line	Option file	System Var	Var Scope	Dynamic
innodb_max_dirty_pages_pct_lwm	Yes	Yes	Yes	Global	Yes
innodb_max_purge_lag	Yes	Yes	Yes	Global	Yes
innodb_max_purge_lag_delay	Yes	Yes	Yes	Global	Yes
innodb_monitor_disable	Yes	Yes	Yes	Global	Yes
innodb_monitor_enable	Yes	Yes	Yes	Global	Yes
innodb_monitor_reset	Yes	Yes	Yes	Global	Yes
innodb_monitor_reset_all	Yes	Yes	Yes	Global	Yes
innodb_old_blocks_pct	Yes	Yes	Yes	Global	Yes
innodb_old_blocks_time	Yes	Yes	Yes	Global	Yes
innodb_online_alter_log_max_size	Yes	Yes	Yes	Global	Yes
innodb_open_files	Yes	Yes	Yes	Global	No
innodb_optimize_fulltext_only	Yes	Yes	Yes	Global	Yes
innodb_page_cleaners	Yes	Yes	Yes	Global	No
innodb_page_size	Yes	Yes	Yes	Global	No
innodb_print_all_deadlocks	Yes	Yes	Yes	Global	Yes
innodb_purge_batch_size	Yes	Yes	Yes	Global	Yes
innodb_purge_threads	Yes	Yes	Yes	Global	No
innodb_random_read_ahead	Yes	Yes	Yes	Global	Yes
innodb_read_ahead_threshold	Yes	Yes	Yes	Global	Yes
innodb_read_io_threads	Yes	Yes	Yes	Global	No
innodb_read_only	Yes	Yes	Yes	Global	No
innodb_replication_delay	Yes	Yes	Yes	Global	Yes
innodb_rollback_on_timeout	Yes	Yes	Yes	Global	No
innodb_rollback_segments	Yes	Yes	Yes	Global	Yes
innodb_sort_buffer_size	Yes	Yes	Yes	Global	No
innodb_spin_wait_delay	Yes	Yes	Yes	Global	Yes
innodb_stats_auto_recalc	Yes	Yes	Yes	Global	Yes
innodb_stats_method	Yes	Yes	Yes	Global	Yes
innodb_stats_on_metadata	Yes	Yes	Yes	Global	Yes
innodb_stats_persistent	Yes	Yes	Yes	Global	Yes
innodb_stats_persistent_sample_pages	Yes	Yes	Yes	Global	Yes
innodb_stats_sample_pages	Yes	Yes	Yes	Global	Yes
innodb_stats_transient_sample_pages	Yes	Yes	Yes	Global	Yes
innodb_status_output	Yes	Yes	Yes	Global	Yes
innodb_status_output_locks	Yes	Yes	Yes	Global	Yes
innodb_strict_mode	Yes	Yes	Yes	Both	Yes
innodb_support_xa	Yes	Yes	Yes	Both	Yes
innodb_sync_array_size	Yes	Yes	Yes	Global	No

Name	Cmd-Line	Option file	System Var	Var Scope	Dynamic
innodb_sync_spin_loops	Yes	Yes	Yes	Global	Yes
innodb_table_locks	Yes	Yes	Yes	Both	Yes
innodb_temp_data_file_path	Yes	Yes	Yes	Global	No
innodb_thread_concurrency	Yes	Yes	Yes	Global	Yes
innodb_thread_sleep_delay	Yes	Yes	Yes	Global	Yes
innodb_undo_directory	Yes	Yes	Yes	Global	No
innodb_undo_logs	Yes	Yes	Yes	Global	Yes
innodb_undo_tablespaces	Yes	Yes	Yes	Global	No
innodb_use_native_aio	Yes	Yes	Yes	Global	No
innodb_use_sys_malloc	Yes	Yes	Yes	Global	No
innodb_version			Yes	Global	No
innodb_write_io_threads	Yes	Yes	Yes	Global	No
insert_id			Yes	Session	Yes
interactive_timeout	Yes	Yes	Yes	Both	Yes
join_buffer_size	Yes	Yes	Yes	Both	Yes
keep_files_on_create	Yes	Yes	Yes	Both	Yes
key_buffer_size	Yes	Yes	Yes	Global	Yes
key_cache_age_threshold	Yes	Yes	Yes	Global	Yes
key_cache_block_size	Yes	Yes	Yes	Global	Yes
key_cache_division_limit	Yes	Yes	Yes	Global	Yes
language	Yes	Yes	Yes	Global	No
large_files_support			Yes	Global	No
large_page_size			Yes	Global	No
large-pages	Yes	Yes			No
- *Variable*: large_pages			Yes	Global	No
last_insert_id			Yes	Session	Yes
lc-messages	Yes	Yes			Yes
- *Variable*: lc_messages			Yes	Both	Yes
lc-messages-dir	Yes	Yes			No
- *Variable*: lc_messages_dir			Yes	Global	No
lc_time_names			Yes	Both	Yes
license			Yes	Global	No
local_infile			Yes	Global	Yes
lock_wait_timeout	Yes	Yes	Yes	Both	Yes
locked_in_memory			Yes	Global	No
log_bin			Yes	Global	No

Name	Cmd-Line	Option file	System Var	Var Scope	Dynamic
log-bin	Yes	Yes	Yes	Global	No
log_bin_basename			Yes	Global	No
log_bin_index			Yes	Global	No
log-bin-trust-function-creators	Yes	Yes			Yes
- *Variable*: log_bin_trust_function_creators			Yes	Global	Yes
log_bin_use_v1_row_events	Yes	Yes	Yes	Global	No
log-bin-use-v1-row-events	Yes	Yes			No
- *Variable*: log_bin_use_v1_row_events			Yes	Global	No
log-error	Yes	Yes			No
- *Variable*: log_error			Yes	Global	No
log_error_verbosity	Yes	Yes	Yes	Global	Yes
log-output	Yes	Yes			Yes
- *Variable*: log_output			Yes	Global	Yes
log-queries-not-using-indexes	Yes	Yes			Yes
- *Variable*: log_queries_not_using_indexes			Yes	Global	Yes
log-slave-updates	Yes	Yes			No
- *Variable*: log_slave_updates			Yes	Global	No
log_slave_updates	Yes	Yes	Yes	Global	No
log_slow_admin_statements			Yes	Global	Yes
log_slow_slave_statements			Yes	Global	Yes
log_throttle_queries_not_using_indexes			Yes	Global	Yes
log_timestamps	Yes	Yes	Yes	Global	Yes
log-warnings	Yes	Yes			Yes
- *Variable*: log_warnings			Yes	Global	Yes
long_query_time	Yes	Yes	Yes	Both	Yes
low-priority-updates	Yes	Yes			Yes
- *Variable*: low_priority_updates			Yes	Both	Yes
lower_case_file_system			Yes	Global	No
lower_case_table_names	Yes	Yes	Yes	Global	No
master_info_repository	Yes	Yes	Yes	Global	Yes
master_verify_checksum			Yes	Global	Yes

Name	Cmd-Line	Option file	System Var	Var Scope	Dynamic
max_allowed_packet	Yes	Yes	Yes	Global	Yes
max_binlog_cache_size	Yes	Yes	Yes	Global	Yes
max_binlog_size	Yes	Yes	Yes	Global	Yes
max_binlog_stmt_cache_size	Yes	Yes	Yes	Global	Yes
max_connect_errors	Yes	Yes	Yes	Global	Yes
max_connections	Yes	Yes	Yes	Global	Yes
max_delayed_threads	Yes	Yes	Yes	Both	Yes
max_error_count	Yes	Yes	Yes	Both	Yes
max_heap_table_size	Yes	Yes	Yes	Both	Yes
max_insert_delayed_threads			Yes	Both	Yes
max_join_size	Yes	Yes	Yes	Both	Yes
max_length_for_sort_data	Yes	Yes	Yes	Both	Yes
max_prepared_stmt_count	Yes	Yes	Yes	Global	Yes
max_relay_log_size	Yes	Yes	Yes	Global	Yes
max_seeks_for_key	Yes	Yes	Yes	Both	Yes
max_sort_length	Yes	Yes	Yes	Both	Yes
max_sp_recursion_depth	Yes	Yes	Yes	Both	Yes
max_statement_time	Yes	Yes	Yes	Both	Yes
max_user_connections	Yes	Yes	Yes	Both	Yes
max_write_lock_count	Yes	Yes	Yes	Global	Yes
memlock	Yes	Yes	Yes	Global	No
metadata_locks_cache_size			Yes	Global	No
metadata_locks_hash_instances			Yes	Global	No
min-examined-row-limit	Yes	Yes	Yes	Both	Yes
myisam_data_pointer_size	Yes	Yes	Yes	Global	Yes
myisam_max_sort_file_size	Yes	Yes	Yes	Global	Yes
myisam_mmap_size	Yes	Yes	Yes	Global	No
myisam_recover_options			Yes	Global	No
myisam_repair_threads	Yes	Yes	Yes	Both	Yes
myisam_sort_buffer_size	Yes	Yes	Yes	Both	Yes
myisam_stats_method	Yes	Yes	Yes	Both	Yes
myisam_use_mmap	Yes	Yes	Yes	Global	Yes
named_pipe			Yes	Global	No
net_buffer_length	Yes	Yes	Yes	Both	Yes
net_read_timeout	Yes	Yes	Yes	Both	Yes
net_retry_count	Yes	Yes	Yes	Both	Yes
net_write_timeout	Yes	Yes	Yes	Both	Yes

Name	Cmd-Line	Option file	System Var	Var Scope	Dynamic
new	Yes	Yes	Yes	Both	Yes
old	Yes	Yes	Yes	Global	No
old-alter-table	Yes	Yes			Yes
- *Variable*: old_alter_table			Yes	Both	Yes
old_passwords			Yes	Both	Yes
open-files-limit	Yes	Yes			No
- *Variable*: open_files_limit			Yes	Global	No
optimizer_prune_level	Yes	Yes	Yes	Both	Yes
optimizer_search_depth	Yes	Yes	Yes	Both	Yes
optimizer_switch	Yes	Yes	Yes	Both	Yes
optimizer_trace			Yes	Both	Yes
optimizer_trace_features			Yes	Both	Yes
optimizer_trace_limit			Yes	Both	Yes
optimizer_trace_max_mem_size			Yes	Both	Yes
optimizer_trace_offset			Yes	Both	Yes
performance_schema	Yes	Yes	Yes	Global	No
performance_schema_accounts_size	Yes	Yes	Yes	Global	No
performance_schema_digests_size	Yes	Yes	Yes	Global	No
performance_schema_events_stages_history_long_size	Yes	Yes	Yes	Global	No
performance_schema_events_stages_history_size	Yes	Yes	Yes	Global	No
performance_schema_events_statements_history_long_size	Yes	Yes	Yes	Global	No
performance_schema_events_statements_history_size	Yes	Yes	Yes	Global	No
performance_schema_events_transactions_history_long_size	Yes	Yes	Yes	Global	No
performance_schema_events_transactions_history_size	Yes	Yes	Yes	Global	No
performance_schema_events_waits_history_long_size	Yes	Yes	Yes	Global	No
performance_schema_events_waits_history_size	Yes	Yes	Yes	Global	No
performance_schema_hosts_size	Yes	Yes	Yes	Global	No
performance_schema_max_cond_classes	Yes	Yes	Yes	Global	No
performance_schema_max_cond_instances	Yes	Yes	Yes	Global	No
performance_schema_max_file_classes	Yes	Yes	Yes	Global	No
performance_schema_max_file_handles	Yes	Yes	Yes	Global	No
performance_schema_max_file_instances	Yes	Yes	Yes	Global	No
performance_schema_max_memory_classes	Yes	Yes	Yes	Global	No
performance_schema_max_metadata_locks	Yes	Yes	Yes	Global	No
performance_schema_max_mutex_classes	Yes	Yes	Yes	Global	No
performance_schema_max_mutex_instances	Yes	Yes	Yes	Global	No

Name	Cmd-Line	Option file	System Var	Var Scope	Dynamic
performance_schema_max_prepared_statements_instances	Yes	Yes	Yes	Global	No
performance_schema_max_program_instances	Yes	Yes	Yes	Global	No
performance_schema_max_rwlock_classes	Yes	Yes	Yes	Global	No
performance_schema_max_rwlock_instances	Yes	Yes	Yes	Global	No
performance_schema_max_socket_classes	Yes	Yes	Yes	Global	No
performance_schema_max_socket_instances	Yes	Yes	Yes	Global	No
performance_schema_max_stage_classes	Yes	Yes	Yes	Global	No
performance_schema_max_statement_classes	Yes	Yes	Yes	Global	No
performance_schema_max_statement_stack	Yes	Yes	Yes	Global	No
performance_schema_max_table_handles	Yes	Yes	Yes	Global	No
performance_schema_max_table_instances	Yes	Yes	Yes	Global	No
performance_schema_max_thread_classes	Yes	Yes	Yes	Global	No
performance_schema_max_thread_instances	Yes	Yes	Yes	Global	No
performance_schema_session_connect_attrs_size	Yes	Yes	Yes	Global	No
performance_schema_setup_actors_size	Yes	Yes	Yes	Global	No
performance_schema_setup_objects_size	Yes	Yes	Yes	Global	No
performance_schema_users_size	Yes	Yes	Yes	Global	No
pid-file	Yes	Yes			No
- *Variable*: pid_file			Yes	Global	No
plugin_dir	Yes	Yes	Yes	Global	No
port	Yes	Yes	Yes	Global	No
preload_buffer_size	Yes	Yes	Yes	Both	Yes
profiling			Yes	Both	Yes
profiling_history_size	Yes	Yes	Yes	Both	Yes
protocol_version			Yes	Global	No
proxy_user			Yes	Session	No
pseudo_slave_mode			Yes	Session	Yes
pseudo_thread_id			Yes	Session	Yes
query_alloc_block_size	Yes	Yes	Yes	Both	Yes
query_cache_limit	Yes	Yes	Yes	Global	Yes
query_cache_min_res_unit	Yes	Yes	Yes	Global	Yes
query_cache_size	Yes	Yes	Yes	Global	Yes
query_cache_type	Yes	Yes	Yes	Both	Yes
query_cache_wlock_invalidate	Yes	Yes	Yes	Both	Yes
query_prealloc_size	Yes	Yes	Yes	Both	Yes
rand_seed1			Yes	Session	Yes
rand_seed2			Yes	Session	Yes
range_alloc_block_size	Yes	Yes	Yes	Both	Yes

Name	Cmd-Line	Option file	System Var	Var Scope	Dynamic
read_buffer_size	Yes	Yes	Yes	Both	Yes
read_only	Yes	Yes	Yes	Global	Yes
read_rnd_buffer_size	Yes	Yes	Yes	Both	Yes
relay-log	Yes	Yes			No
- *Variable*: relay_log			Yes	Global	No
relay_log_basename			Yes	Global	No
relay-log-index	Yes	Yes			No
- *Variable*: relay_log_index			Yes	Global	No
relay_log_index	Yes	Yes	Yes	Global	No
relay_log_info_file	Yes	Yes	Yes	Global	No
relay_log_info_repository			Yes	Global	Yes
relay_log_purge	Yes	Yes	Yes	Global	Yes
relay_log_recovery	Yes	Yes	Yes	Global	No
relay_log_space_limit	Yes	Yes	Yes	Global	No
report-host	Yes	Yes			No
- *Variable*: report_host			Yes	Global	No
report-password	Yes	Yes			No
- *Variable*: report_password			Yes	Global	No
report-port	Yes	Yes			No
- *Variable*: report_port			Yes	Global	No
report-user	Yes	Yes			No
- *Variable*: report_user			Yes	Global	No
rpl_semi_sync_master_enabled			Yes	Global	Yes
rpl_semi_sync_master_timeout			Yes	Global	Yes
rpl_semi_sync_master_trace_level			Yes	Global	Yes
rpl_semi_sync_master_wait_for_slave_count			Yes	Global	Yes
rpl_semi_sync_master_wait_no_slave			Yes	Global	Yes
rpl_semi_sync_master_wait_point			Yes	Global	Yes
rpl_semi_sync_slave_enabled			Yes	Global	Yes
rpl_semi_sync_slave_trace_level			Yes	Global	Yes
rpl_stop_slave_timeout	Yes	Yes	Yes	Global	Yes
secure-auth	Yes	Yes			Yes
- *Variable*: secure_auth			Yes	Global	Yes
secure-file-priv	Yes	Yes			No

Name	Cmd-Line	Option file	System Var	Var Scope	Dynamic
- *Variable*: secure_file_priv			Yes	Global	No
server-id [2162]	Yes	Yes			Yes
- *Variable*: server_id			Yes	Global	Yes
server_uuid [2162]			Yes	Global	No
session_track_schema	Yes	Yes	Yes	Both	Yes
session_track_state_change	Yes	Yes	Yes	Both	Yes
session_track_system_variables	Yes	Yes	Yes	Both	Yes
sha256_password_private_key_path			Yes	Global	No
sha256_password_public_key_path			Yes	Global	No
shared_memory			Yes	Global	No
shared_memory_base_name			Yes	Global	No
skip_external_locking	Yes	Yes	Yes	Global	No
skip-name-resolve	Yes	Yes			No
- *Variable*: skip_name_resolve			Yes	Global	No
skip-networking	Yes	Yes			No
- *Variable*: skip_networking			Yes	Global	No
skip-show-database	Yes	Yes			No
- *Variable*: skip_show_database			Yes	Global	No
slave_allow_batching	Yes	Yes	Yes	Global	Yes
slave_checkpoint_group	Yes	Yes	Yes	Global	Yes
slave_checkpoint_period	Yes	Yes	Yes	Global	Yes
slave_compressed_protocol	Yes	Yes	Yes	Global	Yes
slave_exec_mode	Yes	Yes	Yes	Global	Yes
slave-load-tmpdir	Yes	Yes			No
- *Variable*: slave_load_tmpdir			Yes	Global	No
slave_max_allowed_packet			Yes	Global	Yes
slave-net-timeout	Yes	Yes			Yes
- *Variable*: slave_net_timeout			Yes	Global	Yes
slave_parallel_type			Yes	Global	Yes
slave_parallel_workers			Yes	Global	Yes
slave_pending_jobs_size_max			Yes	Global	Yes
slave_rows_search_algorithms			Yes	Global	Yes
slave-skip-errors	Yes	Yes			No

Name	Cmd-Line	Option file	System Var	Var Scope	Dynamic
- *Variable*: slave_skip_errors			Yes	Global	No
slave_sql_verify_checksum			Yes	Global	Yes
slave_transaction_retries	Yes	Yes	Yes	Global	Yes
slave_type_conversions	Yes	Yes	Yes	Global	No
slow_launch_time	Yes	Yes	Yes	Global	Yes
slow-query-log	Yes	Yes			Yes
- *Variable*: slow_query_log			Yes	Global	Yes
slow_query_log_file	Yes	Yes	Yes	Global	Yes
socket	Yes	Yes	Yes	Global	No
sort_buffer_size	Yes	Yes	Yes	Both	Yes
sql_auto_is_null			Yes	Both	Yes
sql_big_selects			Yes	Both	Yes
sql_buffer_result			Yes	Both	Yes
sql_log_bin			Yes	Both	Yes
sql_log_off			Yes	Both	Yes
sql-mode	Yes	Yes			Yes
- *Variable*: sql_mode			Yes	Both	Yes
sql_notes			Yes	Both	Yes
sql_quote_show_create			Yes	Both	Yes
sql_safe_updates			Yes	Both	Yes
sql_select_limit			Yes	Both	Yes
sql_slave_skip_counter			Yes	Global	Yes
sql_warnings			Yes	Both	Yes
ssl-ca	Yes	Yes			No
- *Variable*: ssl_ca			Yes	Global	No
ssl-capath	Yes	Yes			No
- *Variable*: ssl_capath			Yes	Global	No
ssl-cert	Yes	Yes			No
- *Variable*: ssl_cert			Yes	Global	No
ssl-cipher	Yes	Yes			No
- *Variable*: ssl_cipher			Yes	Global	No
ssl-crl	Yes	Yes			No
- *Variable*: ssl_crl			Yes	Global	No
ssl-crlpath	Yes	Yes			No
- *Variable*: ssl_crlpath			Yes	Global	No

Name	Cmd-Line	Option file	System Var	Var Scope	Dynamic
ssl-key	Yes	Yes			No
- *Variable*: ssl_key			Yes	Global	No
storage_engine			Yes	Both	Yes
stored_program_cache	Yes	Yes	Yes	Global	Yes
sync_binlog	Yes	Yes	Yes	Global	Yes
sync_frm	Yes	Yes	Yes	Global	Yes
sync_master_info	Yes	Yes	Yes	Global	Yes
sync_relay_log	Yes	Yes	Yes	Global	Yes
sync_relay_log_info	Yes	Yes	Yes	Global	Yes
system_time_zone			Yes	Global	No
table_definition_cache			Yes	Global	Yes
table_open_cache			Yes	Global	Yes
table_open_cache_instances			Yes	Global	No
thread_cache_size	Yes	Yes	Yes	Global	Yes
thread_concurrency	Yes	Yes	Yes	Global	No
thread_handling	Yes	Yes	Yes	Global	No
thread_stack	Yes	Yes	Yes	Global	No
time_format			Yes	Global	No
time_zone			Yes	Both	Yes
timed_mutexes	Yes	Yes	Yes	Global	Yes
timestamp			Yes	Session	Yes
tmp_table_size	Yes	Yes	Yes	Both	Yes
tmpdir	Yes	Yes	Yes	Global	No
transaction_alloc_block_size	Yes	Yes	Yes	Both	Yes
transaction_prealloc_size	Yes	Yes	Yes	Both	Yes
tx_isolation			Yes	Both	Yes
tx_read_only			Yes	Both	Yes
unique_checks			Yes	Both	Yes
updatable_views_with_limit	Yes	Yes	Yes	Both	Yes
validate_password_dictionary_file			Yes	Global	No
validate_password_length			Yes	Global	Yes
validate_password_mixed_case_count			Yes	Global	Yes
validate_password_number_count			Yes	Global	Yes
validate_password_policy			Yes	Global	Yes
validate_password_special_char_count			Yes	Global	Yes
validate_user_plugins			Yes	Global	No
version			Yes	Global	No
version_comment			Yes	Global	No

Name	Cmd-Line	Option file	System Var	Var Scope	Dynamic
version_compile_machine			Yes	Global	No
version_compile_os			Yes	Global	No
wait_timeout	Yes	Yes	Yes	Both	Yes
warning_count			Yes	Session	No

[a] This option is dynamic, but only the server should set this information. You should not set the value of this variable manually.

[b] This option is dynamic, but only the server should set this information. You should not set the value of this variable manually.

For additional system variable information, see these sections:

- Section 5.1.5, "Using System Variables", discusses the syntax for setting and displaying system variable values.
- Section 5.1.5.2, "Dynamic System Variables", lists the variables that can be set at runtime.
- Information on tuning system variables can be found in Section 8.11.2, "Tuning Server Parameters".
- Section 14.2.13, "`InnoDB` Startup Options and System Variables", lists `InnoDB` system variables.
- For information on server system variables specific to replication, see Section 16.1.4, "Replication and Binary Logging Options and Variables".

> **Note**
>
> Some of the following variable descriptions refer to "enabling" or "disabling" a variable. These variables can be enabled with the `SET` statement by setting them to `ON` or `1`, or disabled by setting them to `OFF` or `0`. In MySQL 5.7, boolean variables can be set at startup to the values `ON`, `TRUE`, `OFF`, and `FALSE` (not case sensitive), as well as `1` and `0`. See Section 4.2.3.2, "Program Option Modifiers".

Some system variables control the size of buffers or caches. For a given buffer, the server might need to allocate internal data structures. These structures typically are allocated from the total memory allocated to the buffer, and the amount of space required might be platform dependent. This means that when you assign a value to a system variable that controls a buffer size, the amount of space actually available might differ from the value assigned. In some cases, the amount might be less than the value assigned. It is also possible that the server will adjust a value upward. For example, if you assign a value of 0 to a variable for which the minimal value is 1024, the server will set the value to 1024.

Values for buffer sizes, lengths, and stack sizes are given in bytes unless otherwise specified.

Some system variables take file name values. Unless otherwise specified, the default file location is the data directory if the value is a relative path name. To specify the location explicitly, use an absolute path name. Suppose that the data directory is `/var/mysql/data`. If a file-valued variable is given as a relative path name, it will be located under `/var/mysql/data`. If the value is an absolute path name, its location is as given by the path name.

- `autocommit`

Command-Line Format	`--autocommit[=#]`
Option-File Format	`autocommit`
System Variable Name	`autocommit`
Variable Scope	Global, Session
Dynamic Variable	Yes

	Permitted Values	
	Type	boolean
	Default	ON

The autocommit mode. If set to 1, all changes to a table take effect immediately. If set to 0, you must use COMMIT to accept a transaction or ROLLBACK to cancel it. If autocommit is 0 and you change it to 1, MySQL performs an automatic COMMIT of any open transaction. Another way to begin a transaction is to use a START TRANSACTION or BEGIN statement. See Section 13.3.1, "START TRANSACTION, COMMIT, and ROLLBACK Syntax".

By default, client connections begin with autocommit set to 1. To cause clients to begin with a default of 0, set the global autocommit value by starting the server with the --autocommit=0 option. To set the variable using an option file, include these lines:

```
[mysqld]
autocommit=0
```

- automatic_sp_privileges

System Variable Name	automatic_sp_privileges	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	boolean
	Default	TRUE

When this variable has a value of 1 (the default), the server automatically grants the EXECUTE and ALTER ROUTINE privileges to the creator of a stored routine, if the user cannot already execute and alter or drop the routine. (The ALTER ROUTINE privilege is required to drop the routine.) The server also automatically drops those privileges from the creator when the routine is dropped. If automatic_sp_privileges is 0, the server does not automatically add or drop these privileges.

The creator of a routine is the account used to execute the CREATE statement for it. This might not be the same as the account named as the DEFINER in the routine definition.

See also Section 18.2.2, "Stored Routines and MySQL Privileges".

- back_log

System Variable Name	back_log	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	numeric
	Default	-1 (autosized)
	Range	1 .. 65535

The number of outstanding connection requests MySQL can have. This comes into play when the main MySQL thread gets very many connection requests in a very short time. It then takes some time (although very little) for the main thread to check the connection and start a new thread. The back_log value indicates how many requests can be stacked during this short time before MySQL momentarily stops answering new requests. You need to increase this only if you expect a large number of connections in a short period of time.

In other words, this value is the size of the listen queue for incoming TCP/IP connections. Your operating system has its own limit on the size of this queue. The manual page for the Unix listen() system call should have more details. Check your OS documentation for the maximum value for this variable. back_log cannot be set higher than your operating system limit.

The default value is based on the following formula, capped to a limit of 900:

```
50 + (max_connections / 5)
```

- basedir

Command-Line Format	--basedir=path	
	-b	
Option-File Format	basedir	
System Variable Name	basedir	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	directory name

The MySQL installation base directory. This variable can be set with the --basedir option. Relative path names for other variables usually are resolved relative to the base directory.

- big_tables

Command-Line Format	--big-tables	
Option-File Format	big-tables	
System Variable Name	big_tables	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	boolean
	Default	OFF

If set to 1, all temporary tables are stored on disk rather than in memory. This is a little slower, but the error The table *tbl_name* is full does not occur for SELECT operations that require a large temporary table. The default value for a new connection is 0 (use in-memory temporary tables). Normally, you should never need to set this variable, because in-memory tables are automatically converted to disk-based tables as required.

- bind_address

Command-Line Format	--bind-address=addr

Option-File Format	bind-address	
System Variable Name	bind_address	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	string
	Default	*

The value of the --bind-address option.

This variable has no effect for the embedded server (libmysqld) and as of MySQL 5.7.2 is no longer visible within the embedded server.

- block_encryption_mode

Introduced	5.7.4	
Command-Line Format	--block_encryption_mode=#	
Option-File Format	block_encryption_mode=#	
System Variable Name	block_encryption_mode	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	string
	Default	aes-128-ecb

This variable controls the block encryption mode for block-based algorithms such as AES. It affects encryption for AES_ENCRYPT() and AES_DECRYPT().

block_encryption_mode takes a value in aes-*keylen*-*mode* format, where *keylen* is the key length in bits and *mode* is the encryption mode. The value is not case sensitive. Permitted *keylen* values are 128, 192, and 256. Permitted encryption modes depend on whether MySQL was built using OpenSSL or yaSSL:

- For OpenSSL, permitted *mode* values are: ECB, CBC, CFB1, CFB8, CFB128, OFB
- For yaSSL, permitted *mode* values are: ECB, CBC

For example, this statement causes the AES encryption functions to use a key length of 256 bits and the CBC mode:

```
SET block_encryption_mode = 'aes-256-cbc';
```

An error occurs for attempts to set block_encryption_mode to a value containing an unsupported key length or a mode that the SSL library does not support.

This variable was added in MySQL 5.7.4.

- bulk_insert_buffer_size

Command-Line Format	`--bulk_insert_buffer_size=#`	
Option-File Format	`bulk_insert_buffer_size`	
System Variable Name	`bulk_insert_buffer_size`	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Platform Bit Size	`32`
	Type	`numeric`
	Default	`8388608`
	Range	`0 .. 4294967295`
	Permitted Values	
	Platform Bit Size	`64`
	Type	`numeric`
	Default	`8388608`
	Range	`0 .. 18446744073709547520`

`MyISAM` uses a special tree-like cache to make bulk inserts faster for `INSERT ... SELECT`, `INSERT ... VALUES (...), (...), ...`, and `LOAD DATA INFILE` when adding data to nonempty tables. This variable limits the size of the cache tree in bytes per thread. Setting it to 0 disables this optimization. The default value is 8MB.

- `character_set_client`

System Variable Name	`character_set_client`	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	`string`

The character set for statements that arrive from the client. The session value of this variable is set using the character set requested by the client when the client connects to the server. (Many clients support a `--default-character-set` option to enable this character set to be specified explicitly. See also Section 10.1.4, “Connection Character Sets and Collations”.) The global value of the variable is used to set the session value in cases when the client-requested value is unknown or not available, or the server is configured to ignore client requests:

- The client is from a version of MySQL older than MySQL 4.1, and thus does not request a character set.

- The client requests a character set not known to the server. For example, a Japanese-enabled client requests `sjis` when connecting to a server not configured with `sjis` support.

- `mysqld` was started with the `--skip-character-set-client-handshake` option, which causes it to ignore client character set configuration. This reproduces MySQL 4.0 behavior and is useful should you wish to upgrade the server without upgrading all the clients.

ucs2, utf16, utf16le, and utf32 cannot be used as a client character set, which means that they also do not work for SET NAMES or SET CHARACTER SET.

- character_set_connection

System Variable Name	character_set_connection	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	string

The character set used for literals that do not have a character set introducer and for number-to-string conversion.

- character_set_database

System Variable Name	character_set_database	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
Footnote	This option is dynamic, but only the server should set this information. You should not set the value of this variable manually.	
	Permitted Values	
	Type	string

The character set used by the default database. The server sets this variable whenever the default database changes. If there is no default database, the variable has the same value as character_set_server.

- character_set_filesystem

Command-Line Format	--character-set-filesystem=name	
Option-File Format	character-set-filesystem	
System Variable Name	character_set_filesystem	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	string
	Default	binary

The file system character set. This variable is used to interpret string literals that refer to file names, such as in the LOAD DATA INFILE and SELECT ... INTO OUTFILE statements and the LOAD_FILE() function. Such file names are converted from character_set_client to character_set_filesystem before the file opening attempt occurs. The default value is binary, which means that no conversion occurs. For systems on which multi-byte file names are permitted, a different value may be more appropriate. For example, if the system represents file names using UTF-8, set character_set_filesystem to 'utf8'.

- character_set_results

System Variable Name	character_set_results	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	string

The character set used for returning query results such as result sets or error messages to the client.

- character_set_server

Command-Line Format	--character-set-server	
Option-File Format	character-set-server	
System Variable Name	character_set_server	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	string
	Default	latin1

The server's default character set.

- character_set_system

System Variable Name	character_set_system	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	string
	Default	utf8

The character set used by the server for storing identifiers. The value is always utf8.

- character_sets_dir

Command-Line Format	--character-sets-dir=path	
Option-File Format	character-sets-dir	
System Variable Name	character_sets_dir	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	directory name

The directory where character sets are installed.

- collation_connection

System Variable Name	collation_connection	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	string

The collation of the connection character set.

- collation_database

System Variable Name	collation_database	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
Footnote	This option is dynamic, but only the server should set this information. You should not set the value of this variable manually.	
	Permitted Values	
	Type	string

The collation used by the default database. The server sets this variable whenever the default database changes. If there is no default database, the variable has the same value as collation_server.

- collation_server

Command-Line Format	--collation-server	
Option-File Format	collation-server	
System Variable Name	collation_server	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	string
	Default	latin1_swedish_ci

The server's default collation.

- completion_type

Command-Line Format	--completion_type=#	
Option-File Format	completion_type	
System Variable Name	completion_type	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	enumeration
	Default	NO_CHAIN

	Valid Values	NO_CHAIN
		CHAIN
		RELEASE
		0
		1
		2

The transaction completion type. This variable can take the values shown in the following table. The variable can be assigned using either the name values or corresponding integer values.

Value	Description
NO_CHAIN (or 0)	COMMIT and ROLLBACK are unaffected. This is the default value.
CHAIN (or 1)	COMMIT and ROLLBACK are equivalent to COMMIT AND CHAIN and ROLLBACK AND CHAIN, respectively. (A new transaction starts immediately with the same isolation level as the just-terminated transaction.)
RELEASE (or 2)	COMMIT and ROLLBACK are equivalent to COMMIT RELEASE and ROLLBACK RELEASE, respectively. (The server disconnects after terminating the transaction.)

completion_type affects transactions that begin with START TRANSACTION or BEGIN and end with COMMIT or ROLLBACK. It does not apply to implicit commits resulting from execution of the statements listed in Section 13.3.3, “Statements That Cause an Implicit Commit”. It also does not apply for XA COMMIT, XA ROLLBACK, or when autocommit=1.

- concurrent_insert

Command-Line Format	--concurrent_insert[=#]	
Option-File Format	concurrent_insert	
System Variable Name	concurrent_insert	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	enumeration
	Default	AUTO
	Valid Values	NEVER
		AUTO
		ALWAYS
		0
		1
		2

If AUTO (the default), MySQL permits INSERT and SELECT statements to run concurrently for MyISAM tables that have no free blocks in the middle of the data file. If you start mysqld with --skip-new, this variable is set to NEVER.

This variable can take the values shown in the following table. The variable can be assigned using either the name values or corresponding integer values.

Value	Description
NEVER (or 0)	Disables concurrent inserts
AUTO (or 1)	(Default) Enables concurrent insert for MyISAM tables that do not have holes
ALWAYS (or 2)	Enables concurrent inserts for all MyISAM tables, even those that have holes. For a table with a hole, new rows are inserted at the end of the table if it is in use by another thread. Otherwise, MySQL acquires a normal write lock and inserts the row into the hole.

See also Section 8.10.3, “Concurrent Inserts”.

- connect_timeout

Command-Line Format	--connect_timeout=#	
Option-File Format	connect_timeout	
System Variable Name	connect_timeout	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	numeric
	Default	10

The number of seconds that the mysqld server waits for a connect packet before responding with Bad handshake. The default value is 10 seconds.

Increasing the connect_timeout value might help if clients frequently encounter errors of the form Lost connection to MySQL server at '*XXX*', system error: *errno*.

- core_file

System Variable Name	core_file	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	boolean
	Default	OFF

Whether to write a core file if the server crashes. This variable is set by the --core-file option.

- datadir

Command-Line Format	--datadir=path
	-h
Option-File Format	datadir

System Variable Name	`datadir`	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	`directory name`

The MySQL data directory. This variable can be set with the `--datadir` option.

- `date_format`

  This variable is unused. It is deprecated and will be removed in a future MySQL release.

- `datetime_format`

  This variable is unused. It is deprecated and will be removed in a future MySQL release.

- `debug`

Command-Line Format	`--debug[=debug_options]`	
Option-File Format	`debug`	
System Variable Name	`debug`	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type (Unix)	`string`
	Default	`d:t:i:o,/tmp/mysqld.trace`
	Permitted Values	
	Type (Windows)	`string`
	Default	`d:t:i:O,\mysqld.trace`

This variable indicates the current debugging settings. It is available only for servers built with debugging support. The initial value comes from the value of instances of the `--debug` option given at server startup. The global and session values may be set at runtime; the `SUPER` privilege is required, even for the session value.

Assigning a value that begins with `+` or `-` cause the value to added to or subtracted from the current value:

```
mysql> SET debug = 'T';
mysql> SELECT @@debug;
+---------+
| @@debug |
+---------+
| T       |
+---------+

mysql> SET debug = '+P';
mysql> SELECT @@debug;
+---------+
| @@debug |
+---------+
| P:T     |
+---------+

mysql> SET debug = '-P';
mysql> SELECT @@debug;
+---------+
| @@debug |
+---------+
| T       |
+---------+
```

For more information, see Section 22.4.3, “The DBUG Package”.

- `debug_sync`

System Variable Name	`debug_sync`	
Variable Scope	Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	`string`

This variable is the user interface to the Debug Sync facility. Use of Debug Sync requires that MySQL be configured with the `-DENABLE_DEBUG_SYNC=1` option (see Section 2.8.4, “MySQL Source-Configuration Options”). If Debug Sync is not compiled in, this system variable is not available.

The global variable value is read only and indicates whether the facility is enabled. By default, Debug Sync is disabled and the value of `debug_sync` is `OFF`. If the server is started with `--debug-sync-timeout=N`, where `N` is a timeout value greater than 0, Debug Sync is enabled and the value of `debug_sync` is `ON - current signal` followed by the signal name. Also, `N` becomes the default timeout for individual synchronization points.

The session value can be read by any user and will have the same value as the global variable. The session value can be set by users that have the `SUPER` privilege to control synchronization points.

For a description of the Debug Sync facility and how to use synchronization points, see MySQL Internals: Test Synchronization.

- `default_authentication_plugin`

Introduced	5.7.2	
Command-Line Format	`--default-authentication-plugin=plugin_name`	
Option-File Format	`default-authentication-plugin=plugin_name`	
System Variable Name	`default_authentication_plugin`	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	`enumeration`
	Default	`mysql_native_password`
	Valid Values	`mysql_native_password`
		`sha256_password`

The default authentication plugin. Permitted values are mysql_native_password (use MySQL native passwords; this is the default) and sha256_password (use SHA-256 passwords). For more information about these plugins, see Section 6.3.9.1, “The Native Authentication Plugin”, and Section 6.3.9.4, “The SHA-256 Authentication Plugin”.

Note

If you use this variable to change the default authentication plugin to a value other than mysql_native_password, clients older than MySQL 5.5.6 will no longer be able to connect because they will not understand the resulting change to the authentication protocol.

The value of default_authentication_plugin affects these aspects of server operation:

- It determines which authentication plugin the server assigns to new accounts created by CREATE USER and GRANT statements that do not name a plugin explicitly with an IDENTIFIED WITH clause.

- It sets the old_passwords system variable at startup to the value that is consistent with the password hashing method required by the default plugin. The old_passwords value affects hashing of passwords specified in the IDENTIFIED BY clause of CREATE USER and GRANT, and passwords specified as the argument to the PASSWORD() function.

- For an account created with either of the following statements, the server associates the account with the default authentication plugin and assigns the account the given password, hashed according to the value of old_passwords.

```
CREATE USER ... IDENTIFIED BY 'cleartext password';
GRANT ...  IDENTIFIED BY 'cleartext password';
```

- For an account created with either of the following statements, the statement fails if the password hash is not encrypted using the hash format required by the default authentication plugin. Otherwise, the server associates the account with the default authentication plugin and assigns the account the given password hash.

```
CREATE USER ... IDENTIFIED BY PASSWORD 'encrypted password';
GRANT ...  IDENTIFIED BY PASSWORD 'encrypted password';
```

This variable was added in MySQL 5.7.2. Earlier in MySQL 5.7, use the --default-authentication-plugin command-line option instead, which is used the same way at server startup, but cannot be accessed at runtime.

- default_password_lifetime

Introduced	5.7.4	
Command-Line Format	--default_password_lifetime=#	
Option-File Format	default_password_lifetime=#	
System Variable Name	default_password_lifetime	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	integer

	Default	360
	Range	0 .. 65535

This variable defines the global automatic password expiration policy. It applies to accounts that use MySQL built-in authentication methods (accounts that use an authentication plugin of mysql_native_password, mysql_old_password, or sha256_password).

If the value of default_password_lifetime is a positive integer *N*, it indicates the permitted password lifetime; passwords must be changed every *N* days. A value of 0 disables automatic password expiration. The default is 360; passwords must be changed approximately once per year.

The global password expiration policy can be overridden as desired for individual accounts using the ALTER USER statement. See Section 6.3.6, “Password Expiration Policy”.

This variable was added in MySQL 5.7.4.

- default_storage_engine

Command-Line Format	--default-storage-engine=name	
Option-File Format	default-storage-engine	
System Variable Name	default_storage_engine	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	enumeration
	Default	InnoDB

The default storage engine. This variable sets the storage engine for permanent tables only. To set the storage engine for TEMPORARY tables, set the default_tmp_storage_engine system variable.

To see which storage engines are available and enabled, use the SHOW ENGINES statement or query the INFORMATION_SCHEMA ENGINES table.

default_storage_engine should be used in preference to storage_engine, which is deprecated and was removed in MySQL 5.7.5.

If you disable the default storage engine at server startup, you must set the default engine for both permanent and TEMPORARY tables to a different engine or the server will not start.

- default_tmp_storage_engine

Command-Line Format	--default_tmp_storage_engine=name	
Option-File Format	default_tmp_storage_engine	
System Variable Name	default_tmp_storage_engine	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	enumeration
	Default	InnoDB

The default storage engine for `TEMPORARY` tables (created with `CREATE TEMPORARY TABLE`). To set the storage engine for permanent tables, set the `default_storage_engine` system variable. Also see the discussion of that variable regarding possible values.

If you disable the default storage engine at server startup, you must set the default engine for both permanent and `TEMPORARY` tables to a different engine or the server will not start.

- `default_week_format`

Command-Line Format	`--default_week_format=#`	
Option-File Format	`default_week_format`	
System Variable Name	`default_week_format`	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	`numeric`
	Default	`0`
	Range	`0 .. 7`

The default mode value to use for the `WEEK()` function. See Section 12.7, “Date and Time Functions”.

- `delay_key_write`

Command-Line Format	`--delay-key-write[=name]`	
Option-File Format	`delay-key-write`	
System Variable Name	`delay_key_write`	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	`enumeration`
	Default	`ON`
	Valid Values	`ON`
		`OFF`
		`ALL`

This option applies only to `MyISAM` tables. It can have one of the following values to affect handling of the `DELAY_KEY_WRITE` table option that can be used in `CREATE TABLE` statements.

Option	Description
`OFF`	`DELAY_KEY_WRITE` is ignored.
`ON`	MySQL honors any `DELAY_KEY_WRITE` option specified in `CREATE TABLE` statements. This is the default value.
`ALL`	All new opened tables are treated as if they were created with the `DELAY_KEY_WRITE` option enabled.

If DELAY_KEY_WRITE is enabled for a table, the key buffer is not flushed for the table on every index update, but only when the table is closed. This speeds up writes on keys a lot, but if you use this feature, you should add automatic checking of all MyISAM tables by starting the server with the --myisam-recover-options option (for example, --myisam-recover-options=BACKUP,FORCE). See Section 5.1.3, “Server Command Options”, and Section 14.3.1, “MyISAM Startup Options”.

> **Warning**
>
> If you enable external locking with --external-locking, there is no protection against index corruption for tables that use delayed key writes.

- delayed_insert_limit

Deprecated	5.6.7	
Command-Line Format	--delayed_insert_limit=#	
Option-File Format	delayed_insert_limit	
System Variable Name	delayed_insert_limit	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Platform Bit Size	32
	Type	numeric
	Default	100
	Range	1 .. 4294967295
	Permitted Values	
	Platform Bit Size	64
	Type	numeric
	Default	100
	Range	1 .. 18446744073709547520

In MySQL 5.7, this system variable is deprecated (because DELAYED inserts are not supported), and will be removed in a future release.

- delayed_insert_timeout

Deprecated	5.6.7	
Command-Line Format	--delayed_insert_timeout=#	
Option-File Format	delayed_insert_timeout	
System Variable Name	delayed_insert_timeout	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	numeric

	Default	`300`

In MySQL 5.7, this system variable is deprecated (because `DELAYED` inserts are not supported), and will be removed in a future release.

- `delayed_queue_size`

Deprecated	5.6.7	
Command-Line Format	`--delayed_queue_size=#`	
Option-File Format	`delayed_queue_size`	
System Variable Name	`delayed_queue_size`	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Platform Bit Size	`32`
	Type	`numeric`
	Default	`1000`
	Range	`1 .. 4294967295`
	Permitted Values	
	Platform Bit Size	`64`
	Type	`numeric`
	Default	`1000`
	Range	`1 .. 18446744073709547520`

In MySQL 5.7, this system variable is deprecated (because `DELAYED` inserts are not supported), and will be removed in a future release.

- `disconnect_on_expired_password`

Introduced	5.7.1	
Command-Line Format	`--disconnect_on_expired_password=#`	
Option-File Format	`disconnect_on_expired_password`	
System Variable Name	`disconnect_on_expired_password`	
Variable Scope	Session	
Dynamic Variable	No	
	Permitted Values	
	Type	`boolean`
	Default	`ON`

This variable controls how the server handles clients with expired passwords:

  - If the client indicates that it can handle expires passwords, the value of disconnect_on_expired_password is irrelevant. The server permits the client to connect but puts it in sandbox mode.

  - If the client does not indicate that it can handle expires passwords, the server handles the client according to the value of disconnect_on_expired_password:

    - If disconnect_on_expired_password: is enabled, the server disconnects the client.

    - If disconnect_on_expired_password: is disabled, the server permits the client to connect but puts it in sandbox mode.

  For more information about the interaction of client and server settings relating to expired-password handling, see Section 6.3.7, “Password Expiration and Sandbox Mode”.

- div_precision_increment

Command-Line Format	--div_precision_increment=#	
Option-File Format	div_precision_increment	
System Variable Name	div_precision_increment	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	numeric
	Default	4
	Range	0 .. 30

  This variable indicates the number of digits by which to increase the scale of the result of division operations performed with the / operator. The default value is 4. The minimum and maximum values are 0 and 30, respectively. The following example illustrates the effect of increasing the default value.

```
mysql> SELECT 1/7;
+--------+
| 1/7    |
+--------+
| 0.1429 |
+--------+
mysql> SET div_precision_increment = 12;
mysql> SELECT 1/7;
+----------------+
| 1/7            |
+----------------+
| 0.142857142857 |
+----------------+
```

- end_markers_in_json

System Variable Name	end_markers_in_json
Variable Scope	Global, Session
Dynamic Variable	Yes
	Permitted Values

	Type	boolean
	Default	OFF

Whether optimizer JSON output should add end markers.

- eq_range_index_dive_limit

System Variable Name	eq_range_index_dive_limit	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values (<= 5.7.3)	
	Type	numeric
	Default	10
	Range	0 .. 4294967295
	Permitted Values (>= 5.7.4)	
	Type	numeric
	Default	200
	Range	0 .. 4294967295

This variable indicates the number of equality ranges in an equality comparison condition when the optimizer should switch from using index dives to index statistics in estimating the number of qualifying rows. It applies to evaluation of expressions that have either of these equivalent forms, where the optimizer uses a nonunique index to look up *col_name* values:

```
col_name IN(val1, ..., valN)
col_name = val1 OR ... OR col_name = valN
```

In both cases, the expression contains *N* equality ranges. The optimizer can make row estimates using index dives or index statistics. If eq_range_index_dive_limit is greater than 0, the optimizer uses existing index statistics instead of index dives if there are eq_range_index_dive_limit or more equality ranges. Thus, to permit use of index dives for up to *N* equality ranges, set eq_range_index_dive_limit to *N* + 1. Set eq_range_index_dive_limit to 0 to disable use of index statistics and always use index dives regardless of *N*.

For more information, see Equality Range Optimization of Many-Valued Comparisons.

To update table index statistics for best estimates, use ANALYZE TABLE.

- error_count

The number of errors that resulted from the last statement that generated messages. This variable is read only. See Section 13.7.5.16, "SHOW ERRORS Syntax".

- event_scheduler

Command-Line Format	--event-scheduler[=value]
Option-File Format	event-scheduler
System Variable Name	event_scheduler
Variable Scope	Global

Dynamic Variable	Yes	
	Permitted Values	
	Type	`enumeration`
	Default	`OFF`
	Valid Values	`ON`
		`OFF`
		`DISABLED`

This variable indicates the status of the Event Scheduler; possible values are `ON`, `OFF`, and `DISABLED`, with the default being `OFF`. This variable and its effects on the Event Scheduler's operation are discussed in greater detail in the Overview section of the Events chapter [2386].

- `expire_logs_days`

Command-Line Format	`--expire_logs_days=#`	
Option-File Format	`expire_logs_days`	
System Variable Name	`expire_logs_days`	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	`numeric`
	Default	`0`
	Range	`0 .. 99`

The number of days for automatic binary log file removal. The default is 0, which means “no automatic removal.” Possible removals happen at startup and when the binary log is flushed. Log flushing occurs as indicated in Section 5.2, “MySQL Server Logs”.

To remove binary log files manually, use the `PURGE BINARY LOGS` statement. See Section 13.4.1.1, “`PURGE BINARY LOGS` Syntax”.

- `explicit_defaults_for_timestamp`

Command-Line Format	`--explicit_defaults_for_timestamp=#`	
Option-File Format	`explicit_defaults_for_timestamp`	
System Variable Name	`explicit_defaults_for_timestamp`	
Variable Scope	Session	
Dynamic Variable	No	
	Permitted Values	
	Type	`boolean`
	Default	`FALSE`

In MySQL, the `TIMESTAMP` data type differs in nonstandard ways from other data types:

- TIMESTAMP columns not explicitly declared with the NULL attribute are assigned the NOT NULL attribute. (Columns of other data types, if not explicitly declared as NOT NULL, permit NULL values.) Setting such a column to NULL sets it to the current timestamp.

- The first TIMESTAMP column in a table, if not declared with the NULL attribute or an explicit DEFAULT or ON UPDATE clause, is automatically assigned the DEFAULT CURRENT_TIMESTAMP and ON UPDATE CURRENT_TIMESTAMP attributes.

- TIMESTAMP columns following the first one, if not declared with the NULL attribute or an explicit DEFAULT clause, are automatically assigned DEFAULT '0000-00-00 00:00:00' (the “zero” timestamp). For inserted rows that specify no explicit value for such a column, the column is assigned '0000-00-00 00:00:00' and no warning occurs.

Those nonstandard behaviors remain the default for TIMESTAMP but as of MySQL 5.6.6 are deprecated and this warning appears at startup:

```
[Warning] TIMESTAMP with implicit DEFAULT value is deprecated.
Please use --explicit_defaults_for_timestamp server option (see
documentation for more details).
```

As indicated by the warning, to turn off the nonstandard behaviors, enable the new explicit_defaults_for_timestamp system variable at server startup. With this variable enabled, the server handles TIMESTAMP as follows instead:

- TIMESTAMP columns not explicitly declared as NOT NULL permit NULL values. Setting such a column to NULL sets it to NULL, not the current timestamp.

- No TIMESTAMP column is assigned the DEFAULT CURRENT_TIMESTAMP or ON UPDATE CURRENT_TIMESTAMP attributes automatically. Those attributes must be explicitly specified.

- TIMESTAMP columns declared as NOT NULL and without an explicit DEFAULT clause are treated as having no default value. For inserted rows that specify no explicit value for such a column, the result depends on the SQL mode. If strict SQL mode is enabled, an error occurs. If strict SQL mode is not enabled, the column is assigned the implicit default of '0000-00-00 00:00:00' and a warning occurs. This is similar to how MySQL treats other temporal types such as DATETIME.

Note

explicit_defaults_for_timestamp is itself deprecated because its only purpose is to permit control over now-deprecated TIMESTAMP behaviors that will be removed in a future MySQL release. When that removal occurs, explicit_defaults_for_timestamp will have no purpose and will be removed as well.

- external_user

System Variable Name	external_user	
Variable Scope	Session	
Dynamic Variable	No	
	Permitted Values	
	Type	string

The external user name used during the authentication process, as set by the plugin used to authenticate the client. With native (built-in) MySQL authentication, or if the plugin does not set the value, this variable is NULL. See Section 6.3.10, “Proxy Users”.

- flush

Command-Line Format	--flush	
Option-File Format	flush	
System Variable Name	flush	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	boolean
	Default	OFF

If ON, the server flushes (synchronizes) all changes to disk after each SQL statement. Normally, MySQL does a write of all changes to disk only after each SQL statement and lets the operating system handle the synchronizing to disk. See Section C.5.4.2, “What to Do If MySQL Keeps Crashing”. This variable is set to ON if you start mysqld with the --flush option.

- flush_time

Command-Line Format	--flush_time=#	
Option-File Format	flush_time	
System Variable Name	flush_time	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type (Windows)	numeric
	Default	0
	Min Value	0

If this is set to a nonzero value, all tables are closed every flush_time seconds to free up resources and synchronize unflushed data to disk. This option is best used only on systems with minimal resources.

- foreign_key_checks

If set to 1 (the default), foreign key constraints for InnoDB tables are checked. If set to 0, they are ignored. Typically you leave this setting enabled during normal operation, to enforce referential integrity. Disabling foreign key checking can be useful for reloading InnoDB tables in an order different from that required by their parent/child relationships. See Section 14.2.6.6, “InnoDB and FOREIGN KEY Constraints”.

Setting foreign_key_checks to 0 also affects data definition statements: DROP SCHEMA drops a schema even if it contains tables that have foreign keys that are referred to by tables outside the schema, and DROP TABLE drops tables that have foreign keys that are referred to by other tables.

Note

Setting foreign_key_checks to 1 does not trigger a scan of the existing table data. Therefore, rows added to the table while foreign_key_checks = 0 will not be verified for consistency.

- ft_boolean_syntax

Command-Line Format	--ft_boolean_syntax=name	
Option-File Format	ft_boolean_syntax	
System Variable Name	ft_boolean_syntax	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	string
	Default	+ -><()~*:""&|

The list of operators supported by boolean full-text searches performed using IN BOOLEAN MODE. See Section 12.9.2, "Boolean Full-Text Searches".

The default variable value is '+ -><()~*:""&|'. The rules for changing the value are as follows:

- Operator function is determined by position within the string.
- The replacement value must be 14 characters.
- Each character must be an ASCII nonalphanumeric character.
- Either the first or second character must be a space.
- No duplicates are permitted except the phrase quoting operators in positions 11 and 12. These two characters are not required to be the same, but they are the only two that may be.
- Positions 10, 13, and 14 (which by default are set to ":", "&", and "|") are reserved for future extensions.

- ft_max_word_len

Command-Line Format	--ft_max_word_len=#	
Option-File Format	ft_max_word_len	
System Variable Name	ft_max_word_len	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	numeric

	Min Value	10

The maximum length of the word to be included in a MyISAM FULLTEXT index.

> **Note**
>
> FULLTEXT indexes on MyISAM tables must be rebuilt after changing this variable. Use REPAIR TABLE *tbl_name* QUICK.

- ft_min_word_len

Command-Line Format	--ft_min_word_len=#	
Option-File Format	ft_min_word_len	
System Variable Name	ft_min_word_len	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	numeric
	Default	4
	Min Value	1

The minimum length of the word to be included in a MyISAM FULLTEXT index.

> **Note**
>
> FULLTEXT indexes on MyISAM tables must be rebuilt after changing this variable. Use REPAIR TABLE *tbl_name* QUICK.

- ft_query_expansion_limit

Command-Line Format	--ft_query_expansion_limit=#	
Option-File Format	ft_query_expansion_limit	
System Variable Name	ft_query_expansion_limit	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	numeric
	Default	20
	Range	0 .. 1000

The number of top matches to use for full-text searches performed using WITH QUERY EXPANSION.

- ft_stopword_file

Command-Line Format	--ft_stopword_file=file_name
Option-File Format	ft_stopword_file

System Variable Name	ft_stopword_file	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	file name

The file from which to read the list of stopwords for full-text searches on MyISAM tables. The server looks for the file in the data directory unless an absolute path name is given to specify a different directory. All the words from the file are used; comments are *not* honored. By default, a built-in list of stopwords is used (as defined in the storage/myisam/ft_static.c file). Setting this variable to the empty string ('') disables stopword filtering. See also Section 12.9.4, "Full-Text Stopwords".

> **Note**
>
> FULLTEXT indexes on MyISAM tables must be rebuilt after changing this variable or the contents of the stopword file. Use REPAIR TABLE *tbl_name* QUICK.

- general_log

Command-Line Format	--general-log	
Option-File Format	general-log	
System Variable Name	general_log	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	boolean
	Default	OFF

Whether the general query log is enabled. The value can be 0 (or OFF) to disable the log or 1 (or ON) to enable the log. The default value depends on whether the --general_log option is given. The destination for log output is controlled by the log_output system variable; if that value is NONE, no log entries are written even if the log is enabled.

- general_log_file

Command-Line Format	--general-log-file=file_name	
Option-File Format	general_log_file	
System Variable Name	general_log_file	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	file name
	Default	host_name.log

The name of the general query log file. The default value is *host_name*.log, but the initial value can be changed with the --general_log_file option.

- group_concat_max_len

Command-Line Format	--group_concat_max_len=#	
Option-File Format	group_concat_max_len	
System Variable Name	group_concat_max_len	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Platform Bit Size	32
	Type	numeric
	Default	1024
	Range	4 .. 4294967295
	Permitted Values	
	Platform Bit Size	64
	Type	numeric
	Default	1024
	Range	4 .. 18446744073709547520

The maximum permitted result length in bytes for the GROUP_CONCAT() function. The default is 1024.

- have_compress

  YES if the zlib compression library is available to the server, NO if not. If not, the COMPRESS() and UNCOMPRESS() functions cannot be used.

- have_crypt

  YES if the crypt() system call is available to the server, NO if not. If not, the ENCRYPT() function cannot be used.

- have_dynamic_loading

  YES if mysqld supports dynamic loading of plugins, NO if not.

- have_geometry

  YES if the server supports spatial data types, NO if not.

- have_openssl

  This variable is an alias for have_ssl.

- have_profiling

  YES if statement profiling capability is present, NO if not. If present, the profiling system variable controls whether this capability is enabled or disabled. See Section 13.7.5.30, “SHOW PROFILES Syntax”.

  This variable is deprecated and will be removed in a future MySQL release.

- have_query_cache

YES if mysqld supports the query cache, NO if not.

- have_rtree_keys

  YES if RTREE indexes are available, NO if not. (These are used for spatial indexes in MyISAM tables.)

- have_ssl

  YES if mysqld supports SSL connections, NO if not. DISABLED indicates that the server was compiled with SSL support, but but was not started with the appropriate --ssl-*xxx* options. For more information, see Section 6.3.11.2, "Configuring MySQL for SSL".

- have_symlink

  YES if symbolic link support is enabled, NO if not. This is required on Unix for support of the DATA DIRECTORY and INDEX DIRECTORY table options. If the server is started with the --skip-symbolic-links option, the value is DISABLED.

  This variable has no meaning on Windows.

- host_cache_size

System Variable Name	host_cache_size	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	numeric
	Default	-1 (autosized)
	Range	0 .. 65536

  The size of the internal host cache (see Section 8.11.5.2, "DNS Lookup Optimization and the Host Cache"). Setting the size to 0 disables the host cache. Changing the cache size at runtime implicitly causes a FLUSH HOSTS operation to clear the host cache and truncate the host_cache table.

  The default value is 128, plus 1 for a value of max_connections up to 500, plus 1 for every increment of 20 over 500 in the max_connections value, capped to a limit of 2000.

  Use of --skip-host-cache is similar to setting the host_cache_size system variable to 0, but host_cache_size is more flexible because it can also be used to resize, enable, or disable the host cache at runtime, not just at server startup.

  If you start the server with --skip-host-cache, that does not prevent changes to the value of host_cache_size, but such changes have no effect and the cache is not re-enabled even if host_cache_size is set larger than 0.

- hostname

System Variable Name	hostname	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	string

The server sets this variable to the server host name at startup.

- identity

  This variable is a synonym for the last_insert_id variable. It exists for compatibility with other database systems. You can read its value with SELECT @@identity, and set it using SET identity.

- ignore_db_dirs

System Variable Name	ignore_db_dirs	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	string

  A comma-separated list of names that are not considered as database directories in the data directory. The value is set from any instances of --ignore-db-dir given at server startup.

- init_connect

Command-Line Format	--init-connect=name	
Option-File Format	init_connect	
System Variable Name	init_connect	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	string

  A string to be executed by the server for each client that connects. The string consists of one or more SQL statements, separated by semicolon characters. For example, each client session begins by default with autocommit mode enabled. For older servers (before MySQL 5.5.8), there is no global autocommit system variable to specify that autocommit should be disabled by default, but as a workaround init_connect can be used to achieve the same effect:

```
SET GLOBAL init_connect='SET autocommit=0';
```

  The init_connect variable can also be set on the command line or in an option file. To set the variable as just shown using an option file, include these lines:

```
[mysqld]
init_connect='SET autocommit=0'
```

  The content of init_connect is not executed for users that have the SUPER privilege. This is done so that an erroneous value for init_connect does not prevent all clients from connecting. For example, the value might contain a statement that has a syntax error, thus causing client connections to fail. Not executing init_connect for users that have the SUPER privilege enables them to open a connection and fix the init_connect value.

- init_file

Command-Line Format	--init-file=file_name	
Option-File Format	init-file	
System Variable Name	init_file	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	file name

The name of the file specified with the --init-file option when you start the server. This should be a file containing SQL statements that you want the server to execute when it starts. Each statement must be on a single line and should not include comments. No statement terminator such as ;, \g, or \G should be given at the end of each statement.

- innodb_*xxx*

  InnoDB system variables are listed in Section 14.2.13, "InnoDB Startup Options and System Variables". These variables control many aspects of storage, memory use, and I/O patterns for InnoDB tables, and are especially important now that InnoDB is the default storage engine.

- insert_id

  The value to be used by the following INSERT or ALTER TABLE statement when inserting an AUTO_INCREMENT value. This is mainly used with the binary log.

- interactive_timeout

Command-Line Format	--interactive_timeout=#	
Option-File Format	interactive_timeout	
System Variable Name	interactive_timeout	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	numeric
	Default	28800
	Min Value	1

The number of seconds the server waits for activity on an interactive connection before closing it. An interactive client is defined as a client that uses the CLIENT_INTERACTIVE option to mysql_real_connect(). See also wait_timeout.

- join_buffer_size

Command-Line Format	--join_buffer_size=#
Option-File Format	join_buffer_size
System Variable Name	join_buffer_size
Variable Scope	Global, Session

Dynamic Variable	Yes	
	Permitted Values	
	Platform Bit Size	32
	Type (Other)	numeric
	Default	262144
	Range	128 .. 4294967295
	Permitted Values	
	Platform Bit Size	64
	Type (Other)	numeric
	Default	262144
	Range	128 .. 18446744073709547520
	Permitted Values	
	Type (Windows)	numeric
	Default	262144
	Range	128 .. 4294967295

The minimum size of the buffer that is used for plain index scans, range index scans, and joins that do not use indexes and thus perform full table scans. Normally, the best way to get fast joins is to add indexes. Increase the value of `join_buffer_size` to get a faster full join when adding indexes is not possible. One join buffer is allocated for each full join between two tables. For a complex join between several tables for which indexes are not used, multiple join buffers might be necessary.

Unless Batched Key Access (BKA) is used, there is no gain from setting the buffer larger than required to hold each matching row, and all joins allocate at least the minimum size, so use caution in setting this variable to a large value globally. It is better to keep the global setting small and change to a larger setting only in sessions that are doing large joins. Memory allocation time can cause substantial performance drops if the global size is larger than needed by most queries that use it.

When BKA is used, the value of `join_buffer_size` defines how large the batch of keys is in each request to the storage engine. The larger the buffer, the more sequential access will be to the right hand table of a join operation, which can significantly improve performance.

The default is 256KB. The maximum permissible setting for `join_buffer_size` is 4GB–1. Larger values are permitted for 64-bit platforms (except 64-bit Windows, for which large values are truncated to 4GB–1 with a warning).

For additional information about join buffering, see Section 8.2.1.10, “Nested-Loop Join Algorithms”. For information about Batched Key Access, see Section 8.2.1.14, “Block Nested-Loop and Batched Key Access Joins”.

- `keep_files_on_create`

Command-Line Format	`--keep_files_on_create=#`

Option-File Format	keep_files_on_create	
System Variable Name	keep_files_on_create	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	boolean
	Default	OFF

If a MyISAM table is created with no DATA DIRECTORY option, the .MYD file is created in the database directory. By default, if MyISAM finds an existing .MYD file in this case, it overwrites it. The same applies to .MYI files for tables created with no INDEX DIRECTORY option. To suppress this behavior, set the keep_files_on_create variable to ON (1), in which case MyISAM will not overwrite existing files and returns an error instead. The default value is OFF (0).

If a MyISAM table is created with a DATA DIRECTORY or INDEX DIRECTORY option and an existing .MYD or .MYI file is found, MyISAM always returns an error. It will not overwrite a file in the specified directory.

- key_buffer_size

Command-Line Format	--key_buffer_size=#	
Option-File Format	key_buffer_size	
System Variable Name	key_buffer_size	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Platform Bit Size	32
	Type	numeric
	Default	8388608
	Range	8 .. 4294967295
	Permitted Values	
	Platform Bit Size	64
	Type	numeric
	Default	8388608
	Range	8 .. OS_PER_PROCESS_LIMIT

Index blocks for MyISAM tables are buffered and are shared by all threads. key_buffer_size is the size of the buffer used for index blocks. The key buffer is also known as the key cache.

The maximum permissible setting for key_buffer_size is 4GB–1 on 32-bit platforms. Larger values are permitted for 64-bit platforms. The effective maximum size might be less, depending on your available physical RAM and per-process RAM limits imposed by your operating system or hardware platform. The value of this variable indicates the amount of memory requested. Internally, the server allocates as much memory as possible up to this amount, but the actual allocation might be less.

You can increase the value to get better index handling for all reads and multiple writes; on a system whose primary function is to run MySQL using the MyISAM storage engine, 25% of the machine's total memory is an acceptable value for this variable. However, you should be aware that, if you make the value too large (for example, more than 50% of the machine's total memory), your system might start to page and become extremely slow. This is because MySQL relies on the operating system to perform file system caching for data reads, so you must leave some room for the file system cache. You should also consider the memory requirements of any other storage engines that you may be using in addition to MyISAM.

For even more speed when writing many rows at the same time, use LOCK TABLES. See Section 8.2.2.1, "Speed of INSERT Statements".

You can check the performance of the key buffer by issuing a SHOW STATUS statement and examining the Key_read_requests, Key_reads, Key_write_requests, and Key_writes status variables. (See Section 13.7.5, "SHOW Syntax".) The Key_reads/Key_read_requests ratio should normally be less than 0.01. The Key_writes/Key_write_requests ratio is usually near 1 if you are using mostly updates and deletes, but might be much smaller if you tend to do updates that affect many rows at the same time or if you are using the DELAY_KEY_WRITE table option.

The fraction of the key buffer in use can be determined using key_buffer_size in conjunction with the Key_blocks_unused status variable and the buffer block size, which is available from the key_cache_block_size system variable:

```
1 - ((Key_blocks_unused * key_cache_block_size) / key_buffer_size)
```

This value is an approximation because some space in the key buffer is allocated internally for administrative structures. Factors that influence the amount of overhead for these structures include block size and pointer size. As block size increases, the percentage of the key buffer lost to overhead tends to decrease. Larger blocks results in a smaller number of read operations (because more keys are obtained per read), but conversely an increase in reads of keys that are not examined (if not all keys in a block are relevant to a query).

It is possible to create multiple MyISAM key caches. The size limit of 4GB applies to each cache individually, not as a group. See Section 8.9.2, "The MyISAM Key Cache".

- key_cache_age_threshold

Command-Line Format	--key_cache_age_threshold=#	
Option-File Format	key_cache_age_threshold	
System Variable Name	key_cache_age_threshold	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Platform Bit Size	32
	Type	numeric
	Default	300
	Range	100 .. 4294967295
	Permitted Values	

	Platform Bit Size	64
	Type	numeric
	Default	300
	Range	100 .. 18446744073709547520

This value controls the demotion of buffers from the hot sublist of a key cache to the warm sublist. Lower values cause demotion to happen more quickly. The minimum value is 100. The default value is 300. See Section 8.9.2, “The MyISAM Key Cache”.

- key_cache_block_size

Command-Line Format	--key_cache_block_size=#	
Option-File Format	key_cache_block_size	
System Variable Name	key_cache_block_size	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	numeric
	Default	1024
	Range	512 .. 16384

The size in bytes of blocks in the key cache. The default value is 1024. See Section 8.9.2, “The MyISAM Key Cache”.

- key_cache_division_limit

Command-Line Format	--key_cache_division_limit=#	
Option-File Format	key_cache_division_limit	
System Variable Name	key_cache_division_limit	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	numeric
	Default	100
	Range	1 .. 100

The division point between the hot and warm sublists of the key cache buffer list. The value is the percentage of the buffer list to use for the warm sublist. Permissible values range from 1 to 100. The default value is 100. See Section 8.9.2, “The MyISAM Key Cache”.

- large_files_support

System Variable Name	large_files_support
Variable Scope	Global
Dynamic Variable	No

Whether mysqld was compiled with options for large file support.

- large_pages

Command-Line Format	--large-pages	
Option-File Format	large-pages	
System Variable Name	large_pages	
Variable Scope	Global	
Dynamic Variable	No	
Platform Specific	Linux	
	Permitted Values	
	Type (Linux)	boolean
	Default	FALSE

Whether large page support is enabled (via the --large-pages option). See Section 8.11.4.2, “Enabling Large Page Support”.

- large_page_size

System Variable Name	large_page_size	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type (Linux)	numeric
	Default	0

If large page support is enabled, this shows the size of memory pages. Currently, large memory pages are supported only on Linux; on other platforms, the value of this variable is always 0. See Section 8.11.4.2, “Enabling Large Page Support”.

- last_insert_id

The value to be returned from LAST_INSERT_ID(). This is stored in the binary log when you use LAST_INSERT_ID() in a statement that updates a table. Setting this variable does not update the value returned by the mysql_insert_id() C API function.

- lc_messages

Command-Line Format	--lc-messages=name	
Option-File Format	lc-messages	
System Variable Name	lc_messages	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	string

The locale to use for error messages. The server converts the value to a language name and combines it with the value of the lc_messages_dir to produce the location for the error message file. See Section 10.2, “Setting the Error Message Language”.

- lc_messages_dir

Command-Line Format	--lc-messages-dir=path	
Option-File Format	lc-messages-dir	
System Variable Name	lc_messages_dir	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	directory name

The directory where error messages are located. The value is used together with the value of lc_messages to produce the location for the error message file. See Section 10.2, “Setting the Error Message Language”.

- lc_time_names

System Variable Name	lc_time_names	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	string

This variable specifies the locale that controls the language used to display day and month names and abbreviations. This variable affects the output from the DATE_FORMAT(), DAYNAME() and MONTHNAME() functions. Locale names are POSIX-style values such as 'ja_JP' or 'pt_BR'. The default value is 'en_US' regardless of your system's locale setting. For further information, see Section 10.7, “MySQL Server Locale Support”.

- license

System Variable Name	license	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	string
	Default	GPL

The type of license the server has.

- local_infile

System Variable Name	local_infile
Variable Scope	Global

Dynamic Variable	Yes	
	Permitted Values	
	Type	boolean

Whether LOCAL is supported for LOAD DATA INFILE statements. If this variable is disabled, clients cannot use LOCAL in LOAD DATA statements. See Section 6.1.6, "Security Issues with LOAD DATA LOCAL".

- lock_wait_timeout

Command-Line Format	--lock_wait_timeout=#	
Option-File Format	lock_wait_timeout	
System Variable Name	lock_wait_timeout	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	numeric
	Default	31536000
	Range	1 .. 31536000

This variable specifies the timeout in seconds for attempts to acquire metadata locks. The permissible values range from 1 to 31536000 (1 year). The default is 31536000.

This timeout applies to all statements that use metadata locks. These include DML and DDL operations on tables, views, stored procedures, and stored functions, as well as LOCK TABLES, FLUSH TABLES WITH READ LOCK, and HANDLER statements.

This timeout does not apply to implicit accesses to system tables in the mysql database, such as grant tables modified by GRANT or REVOKE statements or table logging statements. The timeout does apply to system tables accessed directly, such as with SELECT or UPDATE.

The timeout value applies separately for each metadata lock attempt. A given statement can require more than one lock, so it is possible for the statement to block for longer than the lock_wait_timeout value before reporting a timeout error. When lock timeout occurs, ER_LOCK_WAIT_TIMEOUT is reported.

lock_wait_timeout does not apply to delayed inserts, which always execute with a timeout of 1 year. This is done to avoid unnecessary timeouts because a session that issues a delayed insert receives no notification of delayed insert timeouts.

- locked_in_memory

System Variable Name	locked_in_memory
Variable Scope	Global
Dynamic Variable	No

Whether mysqld was locked in memory with --memlock.

- log_bin_trust_function_creators

Command-Line Format	--log-bin-trust-function-creators

Option-File Format	log-bin-trust-function-creators	
System Variable Name	log_bin_trust_function_creators	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	boolean
	Default	FALSE

This variable applies when binary logging is enabled. It controls whether stored function creators can be trusted not to create stored functions that will cause unsafe events to be written to the binary log. If set to 0 (the default), users are not permitted to create or alter stored functions unless they have the SUPER privilege in addition to the CREATE ROUTINE or ALTER ROUTINE privilege. A setting of 0 also enforces the restriction that a function must be declared with the DETERMINISTIC characteristic, or with the READS SQL DATA or NO SQL characteristic. If the variable is set to 1, MySQL does not enforce these restrictions on stored function creation. This variable also applies to trigger creation. See Section 18.7, "Binary Logging of Stored Programs".

- log_error

Command-Line Format	--log-error[=name]	
Option-File Format	log-error	
System Variable Name	log_error	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	file name

The location of the error log, or stderr if the server is writing error message to the standard error output. See Section 5.2.2, "The Error Log".

- log_error_verbosity

Introduced	5.7.2	
Command-Line Format	--log_error_verbosity=#	
Option-File Format	log_error_verbosity=#	
System Variable Name	log_error_verbosity	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	numeric
	Default	3
	Range	1 .. 3

This variable controls verbosity of the server in writing error, warning, and note messages to the error log. The following table shows the permitted values. The default is 3.

Verbosity Value	Message Types Logged
1	Errors only
2	Errors and warnings
3	Errors, warnings, and notes

log_error_verbosity was added in MySQL 5.7.2. It is preferred over, and should be used instead of, the older log_warnings system variable. See the description of log_warnings for information about how that variable relates to log_error_verbosity.

- log_output

Command-Line Format	--log-output=name	
Option-File Format	log-output	
System Variable Name	log_output	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	set
	Default	FILE
	Valid Values	TABLE
		FILE
		NONE

The destination for general query log and slow query log output. The value can be a comma-separated list of one or more of the words TABLE (log to tables), FILE (log to files), or NONE (do not log to tables or files). The default value is FILE. NONE, if present, takes precedence over any other specifiers. If the value is NONE log entries are not written even if the logs are enabled. If the logs are not enabled, no logging occurs even if the value of log_output is not NONE. For more information, see Section 5.2.1, “Selecting General Query and Slow Query Log Output Destinations”.

- log_queries_not_using_indexes

Command-Line Format	--log-queries-not-using-indexes	
Option-File Format	log-queries-not-using-indexes	
System Variable Name	log_queries_not_using_indexes	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	boolean
	Default	OFF

Whether queries that do not use indexes are logged to the slow query log. See Section 5.2.5, “The Slow Query Log”.

- log_timestamps

Introduced	5.7.2	
Command-Line Format	--log_timestamps=#	
Option-File Format	log_timestamps=#	
System Variable Name	log_timestamps	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	enumeration
	Default	UTC
	Valid Values	UTC
		SYSTEM

This variable controls the timestamp time zone of error log messages, and of general query log and slow query log messages written to files. It does not affect the time zone of general query log and slow query log messages written to tables (mysql.general_log, mysql.slow_log). Rows retrieved from those tables can be converted from the local system time zone to any desired time zone with CONVERT_TZ() or by setting the session time_zone system variable.

Permitted log_timestamps values are UTC (the default) and SYSTEM (local system time zone).

Timestamps are written using ISO 8601 / RFC 3339 format: YYYY-MM-DDThh:mm:ss.uuuuuu plus a tail value of Z signifying Zulu time (UTC) or ±hh:mm (an offset from UTC).

This variable was added in MySQL 5.7.2. Before 5.7.2, timestamps in log messages were written using the local system time zone by default, not UTC. If you want the previous log message time zone default, set log_timestamps=SYSTEM.

- log_throttle_queries_not_using_indexes

System Variable Name	log_throttle_queries_not_using_indexes	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	numeric
	Default	0

If log_queries_not_using_indexes is enabled, the log_throttle_queries_not_using_indexes variable limits the number of such queries per minute that can be written to the slow query log. A value of 0 (the default) means "no limit". For more information, see Section 5.2.5, "The Slow Query Log".

- log_slow_admin_statements

Introduced	5.7.1
System Variable Name	log_slow_admin_statements
Variable Scope	Global
Dynamic Variable	Yes

	Permitted Values	
	Type	boolean
	Default	OFF

Include slow administrative statements in the statements written to the slow query log. Administrative statements include ALTER TABLE, ANALYZE TABLE, CHECK TABLE, CREATE INDEX, DROP INDEX, OPTIMIZE TABLE, and REPAIR TABLE.

This variable was added in MySQL 5.7.1.

- log_warnings

Deprecated	5.7.2	
Command-Line Format	--log-warnings[=#]	
	-W [#]	
Option-File Format	log-warnings[=#]	
System Variable Name	log_warnings	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Platform Bit Size	32
	Type	numeric
	Default	1
	Range	0 .. 4294967295
	Permitted Values	
	Platform Bit Size	64
	Type	numeric
	Default	1
	Range	0 .. 18446744073709547520

Whether to produce additional warning messages to the error log. Before MySQL 5.7.2, this variable is enabled (1) by default and can be disabled by setting it to 0. The server logs messages about statements that are unsafe for statement-based logging if the value is greater than 0. Aborted connections and access-denied errors for new connection attempts are logged if the value is greater than 1.

As of MySQL 5.7.2, information items previously governed by log_warnings are governed by log_error_verbosity, which is preferred over, and should be used instead of, the older log_warnings system variable. (The log_warnings system variable and --log-warnings command-line option are deprecated and will be removed in a future MySQL release.) The log_warnings and log_error_verbosity variables are related as follows:

- Suppression of all log_warnings items, previously achieved with log_warnings=0, is now achieved with log_error_verbosity=1 (errors only).

  - Items previously printed for `log_warnings=1` or higher now count as warnings and are printed for `log_error_verbosity=2` or higher.

  - Items previously printed for `log_warnings=2` now count as notes and are printed for `log_error_verbosity=3`.

  As of MySQL 5.7.2, the default log level is controlled by `log_error_verbosity`, which has a default of 3. This is a change from the default logging level before 5.7.2: The default for `log_warnings` is 1, which corresponds to `log_error_verbosity=2`. To achieve a logging level similar to the previous default, set `log_error_verbosity=2`.

  In MySQL 5.7.2 and up, use of `log_warnings` is still permitted but maps onto use of `log_error_verbosity` as follows:

  - Setting `log_warnings=0` is equivalent to `log_error_verbosity=1` (errors only).

  - Setting `log_warnings=1` is equivalent to `log_error_verbosity=2` (errors, warnings).

  - Setting `log_warnings=2` (or higher) is equivalent to `log_error_verbosity=3` (errors, warnings, notes), and the server sets `log_warnings` to 2 if a larger value is specified.

- `long_query_time`

Command-Line Format	`--long_query_time=#`	
Option-File Format	`long_query_time`	
System Variable Name	`long_query_time`	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	`numeric`
	Default	`10`
	Min Value	`0`

  If a query takes longer than this many seconds, the server increments the `Slow_queries` status variable. If the slow query log is enabled, the query is logged to the slow query log file. This value is measured in real time, not CPU time, so a query that is under the threshold on a lightly loaded system might be above the threshold on a heavily loaded one. The minimum and default values of `long_query_time` are 0 and 10, respectively. The value can be specified to a resolution of microseconds. For logging to a file, times are written including the microseconds part. For logging to tables, only integer times are written; the microseconds part is ignored. See Section 5.2.5, “The Slow Query Log”.

- `low_priority_updates`

Command-Line Format	`--low-priority-updates`
Option-File Format	`low-priority-updates`
System Variable Name	`low_priority_updates`
Variable Scope	Global, Session
Dynamic Variable	Yes

	Permitted Values	
	Type	`boolean`
	Default	`FALSE`

If set to `1`, all `INSERT`, `UPDATE`, `DELETE`, and `LOCK TABLE WRITE` statements wait until there is no pending `SELECT` or `LOCK TABLE READ` on the affected table. This affects only storage engines that use only table-level locking (such as `MyISAM`, `MEMORY`, and `MERGE`).

- `lower_case_file_system`

System Variable Name	`lower_case_file_system`	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	`boolean`

This variable describes the case sensitivity of file names on the file system where the data directory is located. `OFF` means file names are case sensitive, `ON` means they are not case sensitive. This variable is read only because it reflects a file system attribute and setting it would have no effect on the file system.

- `lower_case_table_names`

Command-Line Format	`--lower_case_table_names[=#]`	
Option-File Format	`lower_case_table_names`	
System Variable Name	`lower_case_table_names`	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	`numeric`
	Default	`0`
	Range	`0 .. 2`

If set to 0, table names are stored as specified and comparisons are case sensitive. If set to 1, table names are stored in lowercase on disk and comparisons are not case sensitive. If set to 2, table names are stored as given but compared in lowercase. This option also applies to database names and table aliases. For additional information, see Section 9.2.2, “Identifier Case Sensitivity”.

You should *not* set this variable to 0 if you are running MySQL on a system that has case-insensitive file names (such as Windows or Mac OS X). If you set this variable to 0 on such a system and access `MyISAM` tablenames using different lettercases, index corruption may result. On Windows the default value is 1. On Mac OS X, the default value is 2.

If you are using `InnoDB` tables, you should set this variable to 1 on all platforms to force names to be converted to lowercase.

The setting of this variable in MySQL 5.7 affects the behavior of replication filtering options with regard to case sensitivity. (Bug #51639) See Section 16.2.3, “How Servers Evaluate Replication Filtering Rules”, for more information.

- max_allowed_packet

Command-Line Format	--max_allowed_packet=#	
Option-File Format	max_allowed_packet	
System Variable Name	max_allowed_packet	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	numeric
	Default	4194304
	Range	1024 .. 1073741824

The maximum size of one packet or any generated/intermediate string, or any parameter sent by the mysql_stmt_send_long_data() C API function. The default is 4MB.

The packet message buffer is initialized to net_buffer_length bytes, but can grow up to max_allowed_packet bytes when needed. This value by default is small, to catch large (possibly incorrect) packets.

You must increase this value if you are using large BLOB columns or long strings. It should be as big as the largest BLOB you want to use. The protocol limit for max_allowed_packet is 1GB. The value should be a multiple of 1024; nonmultiples are rounded down to the nearest multiple.

When you change the message buffer size by changing the value of the max_allowed_packet variable, you should also change the buffer size on the client side if your client program permits it. The default max_allowed_packet value built in to the client library is 1GB, but individual client programs might override this. For example, mysql and mysqldump have defaults of 16MB and 24MB, respectively. They also enable you to change the client-side value by setting max_allowed_packet on the command line or in an option file.

The session value of this variable is read only.

- max_connect_errors

Command-Line Format	--max_connect_errors=#	
Option-File Format	max_connect_errors	
System Variable Name	max_connect_errors	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Platform Bit Size	32
	Type	numeric
	Default	100
	Range	1 .. 4294967295
	Permitted Values	

	Platform Bit Size	64
	Type	numeric
	Default	100
	Range	1 .. 18446744073709547520

If more than this many successive connection requests from a host are interrupted without a successful connection, the server blocks that host from further connections. You can unblock blocked hosts by flushing the host cache. To do so, issue a `FLUSH HOSTS` statement or execute a `mysqladmin flush-hosts` command. If a connection is established successfully within fewer than `max_connect_errors` attempts after a previous connection was interrupted, the error count for the host is cleared to zero. However, once a host is blocked, flushing the host cache is the only way to unblock it. The default is 100.

- `max_connections`

Command-Line Format	`--max_connections=#`	
Option-File Format	`max_connections`	
System Variable Name	`max_connections`	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	numeric
	Default	151
	Range	1 .. 100000

The maximum permitted number of simultaneous client connections. By default, this is 151. See Section C.5.2.7, “`Too many connections`”, for more information.

Increasing this value increases the number of file descriptors that `mysqld` requires. If the required number of descriptors are not available, the server reduces the value of `max_connections`. See Section 8.4.3.1, “How MySQL Opens and Closes Tables”, for comments on file descriptor limits.

Connections refused because the `max_connections` limit is reached increment the `Connection_errors_max_connections` status variable.

- `max_delayed_threads`

Deprecated	5.6.7	
Command-Line Format	`--max_delayed_threads=#`	
Option-File Format	`max_delayed_threads`	
System Variable Name	`max_delayed_threads`	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	numeric
	Default	20

	Range	0 .. 16384

In MySQL 5.7, this system variable is deprecated (because DELAYED inserts are not supported), and will be removed in a future release.

- max_error_count

Command-Line Format	--max_error_count=#	
Option-File Format	max_error_count	
System Variable Name	max_error_count	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	numeric
	Default	64
	Range	0 .. 65535

The maximum number of error, warning, and note messages to be stored for display by the SHOW ERRORS and SHOW WARNINGS statements. This is the same as the number of condition areas in the diagnostics area, and thus the number of conditions that can be inspected by GET DIAGNOSTICS.

- max_heap_table_size

Command-Line Format	--max_heap_table_size=#	
Option-File Format	max_heap_table_size	
System Variable Name	max_heap_table_size	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Platform Bit Size	32
	Type	numeric
	Default	16777216
	Range	16384 .. 4294967295
	Permitted Values	
	Platform Bit Size	64
	Type	numeric
	Default	16777216
	Range	16384 .. 1844674407370954752

This variable sets the maximum size to which user-created MEMORY tables are permitted to grow. The value of the variable is used to calculate MEMORY table MAX_ROWS values. Setting this variable has no effect on any existing MEMORY table, unless the table is re-created with a statement such as CREATE TABLE or altered with ALTER TABLE or TRUNCATE TABLE. A server restart also sets the maximum size of existing MEMORY tables to the global max_heap_table_size value.

This variable is also used in conjunction with tmp_table_size to limit the size of internal in-memory tables. See Section 8.4.4, “How MySQL Uses Internal Temporary Tables”.

max_heap_table_size is not replicated. See Section 16.4.1.21, “Replication and MEMORY Tables”, and Section 16.4.1.34, “Replication and Variables”, for more information.

- max_insert_delayed_threads

Deprecated	5.6.7	
System Variable Name	max_insert_delayed_threads	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	numeric

This variable is a synonym for max_delayed_threads.

In MySQL 5.7, this system variable is deprecated (because DELAYED inserts are not supported), and will be removed in a future release.

- max_join_size

Command-Line Format	--max_join_size=#	
Option-File Format	max_join_size	
System Variable Name	max_join_size	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	numeric
	Default	18446744073709551615
	Range	1 .. 18446744073709551615

Do not permit statements that probably need to examine more than max_join_size rows (for single-table statements) or row combinations (for multiple-table statements) or that are likely to do more than max_join_size disk seeks. By setting this value, you can catch statements where keys are not used properly and that would probably take a long time. Set it if your users tend to perform joins that lack a WHERE clause, that take a long time, or that return millions of rows.

Setting this variable to a value other than DEFAULT resets the value of sql_big_selects to 0. If you set the sql_big_selects value again, the max_join_size variable is ignored.

If a query result is in the query cache, no result size check is performed, because the result has previously been computed and it does not burden the server to send it to the client.

- max_length_for_sort_data

Command-Line Format	--max_length_for_sort_data=#
Option-File Format	max_length_for_sort_data

System Variable Name	`max_length_for_sort_data`	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	`numeric`
	Default	`1024`
	Range	`4 .. 8388608`

The cutoff on the size of index values that determines which `filesort` algorithm to use. See Section 8.2.1.15, “`ORDER BY` Optimization”.

- `max_prepared_stmt_count`

Command-Line Format	`--max_prepared_stmt_count=#`	
Option-File Format	`max_prepared_stmt_count`	
System Variable Name	`max_prepared_stmt_count`	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	`numeric`
	Default	`16382`
	Range	`0 .. 1048576`

This variable limits the total number of prepared statements in the server. It can be used in environments where there is the potential for denial-of-service attacks based on running the server out of memory by preparing huge numbers of statements. If the value is set lower than the current number of prepared statements, existing statements are not affected and can be used, but no new statements can be prepared until the current number drops below the limit. The default value is 16,382. The permissible range of values is from 0 to 1 million. Setting the value to 0 disables prepared statements.

- `max_relay_log_size`

Command-Line Format	`--max_relay_log_size=#`	
Option-File Format	`max_relay_log_size`	
System Variable Name	`max_relay_log_size`	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	`numeric`
	Default	`0`
	Range	`0 .. 1073741824`

If a write by a replication slave to its relay log causes the current log file size to exceed the value of this variable, the slave rotates the relay logs (closes the current file and opens the next one). If `max_relay_log_size` is 0, the server uses `max_binlog_size` for both the binary log and the relay log. If `max_relay_log_size` is greater than 0, it constrains the size of the relay log, which enables you to have different sizes for the two logs. You must set `max_relay_log_size` to between 4096 bytes and 1GB (inclusive), or to 0. The default value is 0. See Section 16.2.1, “Replication Implementation Details”.

- `max_seeks_for_key`

Command-Line Format	`--max_seeks_for_key=#`	
Option-File Format	`max_seeks_for_key`	
System Variable Name	`max_seeks_for_key`	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Platform Bit Size	`32`
	Type	`numeric`
	Default	`4294967295`
	Range	`1 .. 4294967295`
	Permitted Values	
	Platform Bit Size	`64`
	Type	`numeric`
	Default	`18446744073709547520`
	Range	`1 .. 18446744073709547520`

Limit the assumed maximum number of seeks when looking up rows based on a key. The MySQL optimizer assumes that no more than this number of key seeks are required when searching for matching rows in a table by scanning an index, regardless of the actual cardinality of the index (see Section 13.7.5.21, “`SHOW INDEX` Syntax”). By setting this to a low value (say, 100), you can force MySQL to prefer indexes instead of table scans.

- `max_sort_length`

Command-Line Format	`--max_sort_length=#`	
Option-File Format	`max_sort_length`	
System Variable Name	`max_sort_length`	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	`numeric`
	Default	`1024`
	Range	`4 .. 8388608`

The number of bytes to use when sorting data values. Only the first `max_sort_length` bytes of each value are used; the rest are ignored.

`max_sort_length` is ignored for integer, decimal, floating-point, and temporal data types.

- max_sp_recursion_depth

Command-Line Format	--max_sp_recursion_depth[=#]	
Option-File Format	max_sp_recursion_depth	
System Variable Name	max_sp_recursion_depth	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	numeric
	Default	0
	Max Value	255

The number of times that any given stored procedure may be called recursively. The default value for this option is 0, which completely disables recursion in stored procedures. The maximum value is 255.

Stored procedure recursion increases the demand on thread stack space. If you increase the value of max_sp_recursion_depth, it may be necessary to increase thread stack size by increasing the value of thread_stack at server startup.

- max_statement_time

Introduced	5.7.4	
Command-Line Format	--max_statement_time=#	
Option-File Format	max_statement_time=#	
System Variable Name	max_statement_time	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	numeric
	Default	0

The execution timeout for SELECT statements, in milliseconds. If the value is 0, timeouts are not enabled.

max_statement_time applies as follows:

- The global max_statement_time value provides the default for the session value for new connections. The session value applies to SELECT statements executed within the session that include no MAX_STATEMENT_TIME = *N* option or for which *N* is 0.

- max_statement_time applies to read-only SELECT statements. Statements that are not read only are those that invoke a stored function that modifies data as a side effect.

- max_statement_time is ignored for SELECT statements in stored programs.

This variable was added in MySQL 5.7.4.

- max_tmp_tables

This variable is unused. It is deprecated and will be removed in a future MySQL release.

- max_user_connections

Command-Line Format	--max_user_connections=#	
Option-File Format	max_user_connections	
System Variable Name	max_user_connections	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	numeric
	Default	0
	Range	0 .. 4294967295

The maximum number of simultaneous connections permitted to any given MySQL user account. A value of 0 (the default) means "no limit."

This variable has a global value that can be set at server startup or runtime. It also has a read-only session value that indicates the effective simultaneous-connection limit that applies to the account associated with the current session. The session value is initialized as follows:

- If the user account has a nonzero MAX_USER_CONNECTIONS resource limit, the session max_user_connections value is set to that limit.
- Otherwise, the session max_user_connections value is set to the global value.

Account resource limits are specified using the GRANT statement. See Section 6.3.4, "Setting Account Resource Limits", and Section 13.7.1.4, "GRANT Syntax".

- max_write_lock_count

Command-Line Format	--max_write_lock_count=#	
Option-File Format	max_write_lock_count	
System Variable Name	max_write_lock_count	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Platform Bit Size	32
	Type	numeric
	Default	4294967295
	Range	1 .. 4294967295
	Permitted Values	
	Platform Bit Size	64
	Type	numeric

	Default	18446744073709547520
	Range	1 .. 18446744073709547520

After this many write locks, permit some pending read lock requests to be processed in between.

- `metadata_locks_cache_size`

Deprecated	5.7.4	
System Variable Name	`metadata_locks_cache_size`	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	`numeric`
	Default	`1024`
	Range	`1 .. 1048576`

The size of the metadata locks cache. The server uses this cache to avoid creation and destruction of synchronization objects. This is particularly helpful on systems where such operations are expensive, such as Windows XP.

In MySQL 5.7.4, metadata locking implementation changes make this variable unnecessary, so it is deprecated and will be removed in a future MySQL release.

- `metadata_locks_hash_instances`

Deprecated	5.7.4	
System Variable Name	`metadata_locks_hash_instances`	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	`numeric`
	Default	`8`
	Range	`1 .. 1024`

The set of metadata locks can be partitioned into separate hashes to permit connections accessing different objects to use different locking hashes and reduce contention. The `metadata_locks_hash_instances` system variable specifies the number of hashes (default 8).

In MySQL 5.7.4, metadata locking implementation changes make this variable unnecessary, so it is deprecated and will be removed in a future MySQL release.

- `min_examined_row_limit`

Command-Line Format	`--min-examined-row-limit=#`
Option-File Format	`min-examined-row-limit`
System Variable Name	`min_examined_row_limit`
Variable Scope	Global, Session

Dynamic Variable	Yes	
	Permitted Values	
	Platform Bit Size	32
	Type	numeric
	Default	0
	Range	0 .. 4294967295
	Permitted Values	
	Platform Bit Size	64
	Type	numeric
	Default	0
	Range	0 .. 18446744073709547520

Queries that examine fewer than this number of rows are not logged to the slow query log.

- myisam_data_pointer_size

Command-Line Format	--myisam_data_pointer_size=#	
Option-File Format	myisam_data_pointer_size	
System Variable Name	myisam_data_pointer_size	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	numeric
	Default	6
	Range	2 .. 7

The default pointer size in bytes, to be used by CREATE TABLE for MyISAM tables when no MAX_ROWS option is specified. This variable cannot be less than 2 or larger than 7. The default value is 6. See Section C.5.2.12, "The table is full".

- myisam_max_sort_file_size

Command-Line Format	--myisam_max_sort_file_size=#	
Option-File Format	myisam_max_sort_file_size	
System Variable Name	myisam_max_sort_file_size	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Platform Bit Size	32
	Type	numeric
	Default	2147483648

	Permitted Values	
	Platform Bit Size	64
	Type	numeric
	Default	9223372036854775807

The maximum size of the temporary file that MySQL is permitted to use while re-creating a MyISAM index (during REPAIR TABLE, ALTER TABLE, or LOAD DATA INFILE). If the file size would be larger than this value, the index is created using the key cache instead, which is slower. The value is given in bytes.

The default value is 2GB. If MyISAM index files exceed this size and disk space is available, increasing the value may help performance. The space must be available in the file system containing the directory where the original index file is located.

- myisam_mmap_size

Command-Line Format	--myisam_mmap_size=#	
Option-File Format	myisam_mmap_size	
System Variable Name	myisam_mmap_size	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Platform Bit Size	32
	Type	numeric
	Default	4294967295
	Range	7 .. 4294967295
	Permitted Values	
	Platform Bit Size	64
	Type	numeric
	Default	18446744073709547520
	Range	7 .. 18446744073709547520

The maximum amount of memory to use for memory mapping compressed MyISAM files. If many compressed MyISAM tables are used, the value can be decreased to reduce the likelihood of memory-swapping problems.

- myisam_recover_options

System Variable Name	myisam_recover_options
Variable Scope	Global
Dynamic Variable	No

The value of the --myisam-recover-options option. See Section 5.1.3, “Server Command Options”.

- `myisam_repair_threads`

Command-Line Format	`--myisam_repair_threads=#`	
Option-File Format	`myisam_repair_threads`	
System Variable Name	`myisam_repair_threads`	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Platform Bit Size	`32`
	Type	`numeric`
	Default	`1`
	Range	`1 .. 4294967295`
	Permitted Values	
	Platform Bit Size	`64`
	Type	`numeric`
	Default	`1`
	Range	`1 .. 18446744073709547520`

If this value is greater than 1, `MyISAM` table indexes are created in parallel (each index in its own thread) during the `Repair by sorting` process. The default value is 1.

> **Note**
>
> Multi-threaded repair is still *beta-quality* code.

- `myisam_sort_buffer_size`

Command-Line Format	`--myisam_sort_buffer_size=#`	
Option-File Format	`myisam_sort_buffer_size`	
System Variable Name	`myisam_sort_buffer_size`	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Platform Bit Size	`32`
	Type (Other)	`numeric`
	Default	`8388608`
	Range	`4096 .. 4294967295`
	Permitted Values	
	Platform Bit Size	`64`

	Type (Other)	numeric
	Default	8388608
	Range	4096 .. 18446744073709547520
	Permitted Values	
	Type (Windows)	numeric
	Default	8388608
	Range	4096 .. 4294967295

The size of the buffer that is allocated when sorting MyISAM indexes during a REPAIR TABLE or when creating indexes with CREATE INDEX or ALTER TABLE.

The maximum permissible setting for myisam_sort_buffer_size is 4GB–1. Larger values are permitted for 64-bit platforms (except 64-bit Windows, for which large values are truncated to 4GB–1 with a warning).

- myisam_stats_method

Command-Line Format	--myisam_stats_method=name	
Option-File Format	myisam_stats_method	
System Variable Name	myisam_stats_method	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	enumeration
	Default	nulls_unequal
	Valid Values	nulls_equal
		nulls_unequal
		nulls_ignored

How the server treats NULL values when collecting statistics about the distribution of index values for MyISAM tables. This variable has three possible values, nulls_equal, nulls_unequal, and nulls_ignored. For nulls_equal, all NULL index values are considered equal and form a single value group that has a size equal to the number of NULL values. For nulls_unequal, NULL values are considered unequal, and each NULL forms a distinct value group of size 1. For nulls_ignored, NULL values are ignored.

The method that is used for generating table statistics influences how the optimizer chooses indexes for query execution, as described in Section 8.3.7, “InnoDB and MyISAM Index Statistics Collection”.

- myisam_use_mmap

Command-Line Format	--myisam_use_mmap
Option-File Format	myisam_use_mmap
System Variable Name	myisam_use_mmap
Variable Scope	Global

Dynamic Variable	Yes	
	Permitted Values	
	Type	boolean
	Default	OFF

Use memory mapping for reading and writing MyISAM tables.

- named_pipe

System Variable Name	named_pipe	
Variable Scope	Global	
Dynamic Variable	No	
Platform Specific	Windows	
	Permitted Values	
	Type (Windows)	boolean
	Default	OFF

(Windows only.) Indicates whether the server supports connections over named pipes.

- net_buffer_length

Command-Line Format	--net_buffer_length=#	
Option-File Format	net_buffer_length	
System Variable Name	net_buffer_length	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	numeric
	Default	16384
	Range	1024 .. 1048576

Each client thread is associated with a connection buffer and result buffer. Both begin with a size given by net_buffer_length but are dynamically enlarged up to max_allowed_packet bytes as needed. The result buffer shrinks to net_buffer_length after each SQL statement.

This variable should not normally be changed, but if you have very little memory, you can set it to the expected length of statements sent by clients. If statements exceed this length, the connection buffer is automatically enlarged. The maximum value to which net_buffer_length can be set is 1MB.

The session value of this variable is read only.

- net_read_timeout

Command-Line Format	--net_read_timeout=#
Option-File Format	net_read_timeout
System Variable Name	net_read_timeout

Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	numeric
	Default	30
	Min Value	1

The number of seconds to wait for more data from a connection before aborting the read. When the server is reading from the client, net_read_timeout is the timeout value controlling when to abort. When the server is writing to the client, net_write_timeout is the timeout value controlling when to abort. See also slave_net_timeout.

- net_retry_count

Command-Line Format	--net_retry_count=#	
Option-File Format	net_retry_count	
System Variable Name	net_retry_count	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Platform Bit Size	32
	Type	numeric
	Default	10
	Range	1 .. 4294967295
	Permitted Values	
	Platform Bit Size	64
	Type	numeric
	Default	10
	Range	1 .. 18446744073709547520

If a read or write on a communication port is interrupted, retry this many times before giving up. This value should be set quite high on FreeBSD because internal interrupts are sent to all threads.

- net_write_timeout

Command-Line Format	--net_write_timeout=#	
Option-File Format	net_write_timeout	
System Variable Name	net_write_timeout	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	

	Type	numeric
	Default	60
	Min Value	1

The number of seconds to wait for a block to be written to a connection before aborting the write. See also net_read_timeout.

- new

Command-Line Format	--new	
	-n	
Option-File Format	new	
System Variable Name	new	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
Disabled by	skip-new	
	Permitted Values	
	Type	boolean
	Default	FALSE

This variable was used in MySQL 4.0 to turn on some 4.1 behaviors, and is retained for backward compatibility. In MySQL 5.7, its value is always OFF.

- old

Command-Line Format	--old
Option-File Format	old
System Variable Name	old
Variable Scope	Global
Dynamic Variable	No

old is a compatibility variable. It is disabled by default, but can be enabled at startup to revert the server to behaviors present in older versions.

Currently, when old is enabled, it changes the default scope of index hints to that used prior to MySQL 5.1.17. That is, index hints with no FOR clause apply only to how indexes are used for row retrieval and not to resolution of ORDER BY or GROUP BY clauses. (See Section 13.2.9.3, “Index Hint Syntax”.) Take care about enabling this in a replication setup. With statement-based binary logging, having different modes for the master and slaves might lead to replication errors.

- old_alter_table

Command-Line Format	--old-alter-table
Option-File Format	old-alter-table
System Variable Name	old_alter_table
Variable Scope	Global, Session

Dynamic Variable	Yes	
	Permitted Values	
	Type	boolean
	Default	OFF

When this variable is enabled, the server does not use the optimized method of processing an ALTER TABLE operation. It reverts to using a temporary table, copying over the data, and then renaming the temporary table to the original, as used by MySQL 5.0 and earlier. For more information on the operation of ALTER TABLE, see Section 13.1.6, "ALTER TABLE Syntax".

- old_passwords

System Variable Name	old_passwords	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	enumeration
	Default	0
	Valid Values	0
		1
		2

This variable controls the password hashing method used by the PASSWORD() function. It also influences password hashing performed by CREATE USER and GRANT statements that specify a password using an IDENTIFIED BY clause.

The following table shows the permitted values of old_passwords, the password hashing method for each value, and which authentication plugins use passwords hashed with each method.

Value	Password Hashing Method	Associated Authentication Plugin
0	MySQL 4.1 native hashing	mysql_native_password
1	Pre-4.1 ("old") hashing	mysql_old_password
2	SHA-256 hashing	sha256_password

If old_passwords=1, PASSWORD(*str*) returns the same value as OLD_PASSWORD(*str*). The latter function is not affected by the value of old_passwords.

If you set old_passwords=2, follow the instructions for using the sha256_password plugin at Section 6.3.9.4, "The SHA-256 Authentication Plugin".

The server sets the global old_passwords value during startup to be consistent with the password hashing method required by the default authentication plugin. The default plugin is mysql_native_password unless the default_authentication_plugin system variable is set otherwise.

As of MySQL 5.7.1, when a client successfully connects to the server, the server sets the session old_passwords value appropriately for the account authentication method. For example, if the account uses the sha256_password authentication plugin, the server sets old_passwords=2.

For additional information about authentication plugins and hashing formats, see Section 6.3.8, “Pluggable Authentication”, and Section 6.1.2.4, “Password Hashing in MySQL”.

> **Note**
>
> Passwords that use the pre-4.1 hashing method are less secure than passwords that use the native password hashing method and should be avoided. Pre-4.1 passwords are deprecated and support for them will be removed in a future MySQL release. For account upgrade instructions, see Section 6.3.9.3, “Migrating Away from Pre-4.1 Password Hashing and the `mysql_old_password` Plugin”.

- `open_files_limit`

Command-Line Format	`--open-files-limit=#`	
Option-File Format	`open-files-limit`	
System Variable Name	`open_files_limit`	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	`numeric`
	Default	`(autosized)`
	Range	`0 .. platform dependent`

The number of files that the operating system permits `mysqld` to open. The value of this variable at runtime is the real value permitted by the system and might be different from the value you specify at server startup. The value is 0 on systems where MySQL cannot change the number of open files.

The effective `open_files_limit` value is based on the value specified at system startup (if any) and the values of `max_connections` and `table_open_cache`, using these formulas:

```
1) 10 + max_connections + (table_open_cache * 2)
2) max_connections * 5
3) open_files_limit value specified at startup, 5000 if none
```

The server attempts to obtain the number of file descriptors using the maximum of those three values. If that many descriptors cannot be obtained, the server attempts to obtain as many as the system will permit.

- `optimizer_prune_level`

Command-Line Format	`--optimizer_prune_level[=#]`	
Option-File Format	`optimizer_prune_level`	
System Variable Name	`optimizer_prune_level`	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	`boolean`
	Default	`1`

Controls the heuristics applied during query optimization to prune less-promising partial plans from the optimizer search space. A value of 0 disables heuristics so that the optimizer performs an exhaustive search. A value of 1 causes the optimizer to prune plans based on the number of rows retrieved by intermediate plans.

- `optimizer_search_depth`

Command-Line Format	`--optimizer_search_depth[=#]`	
Option-File Format	`optimizer_search_depth`	
System Variable Name	`optimizer_search_depth`	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	`numeric`
	Default	`62`
	Range	`0 .. 62`

The maximum depth of search performed by the query optimizer. Values larger than the number of relations in a query result in better query plans, but take longer to generate an execution plan for a query. Values smaller than the number of relations in a query return an execution plan quicker, but the resulting plan may be far from being optimal. If set to 0, the system automatically picks a reasonable value.

- `optimizer_switch`

Command-Line Format	`--optimizer_switch=value`	
Option-File Format	`optimizer_switch`	
System Variable Name	`optimizer_switch`	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	`set`
	Valid Values	`batched_key_access={on|off}`
		`block_nested_loop={on|off}`
		`engine_condition_pushdown={on|off}`
		`firstmatch={on|off}`
		`index_condition_pushdown={on|off}`
		`index_merge={on|off}`
		`index_merge_intersection={on|off}`
		`index_merge_sort_union={on|off}`
		`index_merge_union={on|off}`
		`loosescan={on|off}`
		`materialization={on|off}`

		mrr={on|off}
		mrr_cost_based={on|off}
		semijoin={on|off}
		subquery_materialization_cost_based={on|off}
		use_index_extensions={on|off}

The optimizer_switch system variable enables control over optimizer behavior. The value of this variable is a set of flags, each of which has a value of on or off to indicate whether the corresponding optimizer behavior is enabled or disabled. This variable has global and session values and can be changed at runtime. The global default can be set at server startup.

To see the current set of optimizer flags, select the variable value:

```
mysql> SELECT @@optimizer_switch\G
*************************** 1. row ***************************
@@optimizer_switch: index_merge=on,index_merge_union=on,
                    index_merge_sort_union=on,
                    index_merge_intersection=on,
                    engine_condition_pushdown=on,
                    index_condition_pushdown=on,
                    mrr=on,mrr_cost_based=on,
                    block_nested_loop=on,batched_key_access=off,
                    materialization=on,semijoin=on,loosescan=on,
                    firstmatch=on,
                    subquery_materialization_cost_based=on,
                    use_index_extensions=on
```

For more information about the syntax of this variable and the optimizer behaviors that it controls, see Section 8.8.6.2, “Controlling Switchable Optimizations”.

- optimizer_trace

System Variable Name	optimizer_trace	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	string

This variable controls optimizer tracing. For details, see MySQL Internals: Tracing the Optimizer.

- optimizer_trace_features

System Variable Name	optimizer_trace_features	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	string

This variable enables or disables selected optimizer tracing features. For details, see MySQL Internals: Tracing the Optimizer.

- optimizer_trace_limit

System Variable Name	optimizer_trace_limit	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	numeric
	Default	1

The maximum number of optimizer traces to display. For details, see MySQL Internals: Tracing the Optimizer.

- optimizer_trace_max_mem_size

System Variable Name	optimizer_trace_max_mem_size	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	numeric
	Default	16384

The maximum cumulative size of stored optimizer traces. For details, see MySQL Internals: Tracing the Optimizer.

- optimizer_trace_offset

System Variable Name	optimizer_trace_offset	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	numeric
	Default	-1

The offset of optimizer traces to display. For details, see MySQL Internals: Tracing the Optimizer.

- performance_schema_*xxx*

Performance Schema system variables are listed in Section 20.12, “Performance Schema System Variables”. These variables may be used to configure Performance Schema operation.

- pid_file

Command-Line Format	--pid-file=file_name
Option-File Format	pid-file
System Variable Name	pid_file
Variable Scope	Global
Dynamic Variable	No
	Permitted Values

	Type	file name

The path name of the process ID (PID) file. This variable can be set with the `--pid-file` option.

- `plugin_dir`

Command-Line Format	`--plugin_dir=path`	
Option-File Format	`plugin_dir`	
System Variable Name	`plugin_dir`	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	`directory name`
	Default	`BASEDIR/lib/plugin`

The path name of the plugin directory.

If the plugin directory is writable by the server, it may be possible for a user to write executable code to a file in the directory using `SELECT ... INTO DUMPFILE`. This can be prevented by making `plugin_dir` read only to the server or by setting `--secure-file-priv` to a directory where `SELECT` writes can be made safely.

- `port`

Command-Line Format	`--port=#`	
	`-P`	
Option-File Format	`port`	
System Variable Name	`port`	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	`numeric`
	Default	`3306`
	Range	`0 .. 65535`

The number of the port on which the server listens for TCP/IP connections. This variable can be set with the `--port` option.

- `preload_buffer_size`

Command-Line Format	`--preload_buffer_size=#`	
Option-File Format	`preload_buffer_size`	
System Variable Name	`preload_buffer_size`	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	

	Type	numeric
	Default	32768
	Range	1024 .. 1073741824

The size of the buffer that is allocated when preloading indexes.

- `profiling`

  If set to 0 or `OFF` (the default), statement profiling is disabled. If set to 1 or `ON`, statement profiling is enabled and the `SHOW PROFILE` and `SHOW PROFILES` statements provide access to profiling information. See Section 13.7.5.30, "`SHOW PROFILES` Syntax".

  This variable is deprecated and will be removed in a future MySQL release.

- `profiling_history_size`

  The number of statements for which to maintain profiling information if `profiling` is enabled. The default value is 15. The maximum value is 100. Setting the value to 0 effectively disables profiling. See Section 13.7.5.30, "`SHOW PROFILES` Syntax".

  This variable is deprecated and will be removed in a future MySQL release.

- `protocol_version`

System Variable Name	`protocol_version`	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	numeric

  The version of the client/server protocol used by the MySQL server.

- `proxy_user`

System Variable Name	`proxy_user`	
Variable Scope	Session	
Dynamic Variable	No	
	Permitted Values	
	Type	string

  If the current client is a proxy for another user, this variable is the proxy user account name. Otherwise, this variable is `NULL`. See Section 6.3.10, "Proxy Users".

- `pseudo_slave_mode`

System Variable Name	`pseudo_slave_mode`	
Variable Scope	Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	numeric

This variable is for internal server use.

- `pseudo_thread_id`

System Variable Name	`pseudo_thread_id`	
Variable Scope	Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	`numeric`

This variable is for internal server use.

- `query_alloc_block_size`

Command-Line Format	`--query_alloc_block_size=#`	
Option-File Format	`query_alloc_block_size`	
System Variable Name	`query_alloc_block_size`	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Platform Bit Size	`32`
	Type	`numeric`
	Default	`8192`
	Range	`1024 .. 4294967295`
	Block Size	`1024`
	Permitted Values	
	Platform Bit Size	`64`
	Type	`numeric`
	Default	`8192`
	Range	`1024 .. 18446744073709547520`
	Block Size	`1024`

The allocation size of memory blocks that are allocated for objects created during statement parsing and execution. If you have problems with memory fragmentation, it might help to increase this parameter.

- `query_cache_limit`

Command-Line Format	`--query_cache_limit=#`
Option-File Format	`query_cache_limit`
System Variable Name	`query_cache_limit`
Variable Scope	Global

Dynamic Variable	Yes	
	Permitted Values	
	Platform Bit Size	`32`
	Type	`numeric`
	Default	`1048576`
	Range	`0 .. 4294967295`
	Permitted Values	
	Platform Bit Size	`64`
	Type	`numeric`
	Default	`1048576`
	Range	`0 .. 18446744073709547520`

Do not cache results that are larger than this number of bytes. The default value is 1MB.

- `query_cache_min_res_unit`

Command-Line Format	`--query_cache_min_res_unit=#`	
Option-File Format	`query_cache_min_res_unit`	
System Variable Name	`query_cache_min_res_unit`	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Platform Bit Size	`32`
	Type	`numeric`
	Default	`4096`
	Range	`512 .. 4294967295`
	Permitted Values	
	Platform Bit Size	`64`
	Type	`numeric`
	Default	`4096`
	Range	`512 .. 18446744073709547520`

The minimum size (in bytes) for blocks allocated by the query cache. The default value is 4096 (4KB). Tuning information for this variable is given in Section 8.9.3.3, “Query Cache Configuration”.

- `query_cache_size`

Command-Line Format	`--query_cache_size=#`
Option-File Format	`query_cache_size`
System Variable Name	`query_cache_size`

Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Platform Bit Size	32
	Type	numeric
	Default	1048576
	Range	0 .. 4294967295
	Permitted Values	
	Platform Bit Size	64
	Type	numeric
	Default	1048576
	Range	0 .. 18446744073709547520

The amount of memory allocated for caching query results. By default, the query cache is disabled. This is achieved using a default value of 1M, with a default for query_cache_type of 0. (To reduce overhead significantly if you set the size to 0, you should also start the server with query_cache_type=0.

The permissible values are multiples of 1024; other values are rounded down to the nearest multiple. Note that query_cache_size bytes of memory are allocated even if query_cache_type is set to 0. See Section 8.9.3.3, “Query Cache Configuration”, for more information.

The query cache needs a minimum size of about 40KB to allocate its structures. (The exact size depends on system architecture.) If you set the value of query_cache_size too small, a warning will occur, as described in Section 8.9.3.3, “Query Cache Configuration”.

- query_cache_type

Command-Line Format	--query_cache_type=#	
Option-File Format	query_cache_type	
System Variable Name	query_cache_type	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	enumeration
	Default	0
	Valid Values	0
		1
		2

Set the query cache type. Setting the GLOBAL value sets the type for all clients that connect thereafter. Individual clients can set the SESSION value to affect their own use of the query cache. Possible values are shown in the following table.

Option	Description
`0` or `OFF`	Do not cache results in or retrieve results from the query cache. Note that this does not deallocate the query cache buffer. To do that, you should set `query_cache_size` to 0.
`1` or `ON`	Cache all cacheable query results except for those that begin with `SELECT SQL_NO_CACHE`.
`2` or `DEMAND`	Cache results only for cacheable queries that begin with `SELECT SQL_CACHE`.

This variable defaults to `OFF`.

If the server is started with `query_cache_type` set to 0, it does not acquire the query cache mutex at all, which means that the query cache cannot be enabled at runtime and there is reduced overhead in query execution.

- `query_cache_wlock_invalidate`

Command-Line Format	`--query_cache_wlock_invalidate`	
Option-File Format	`query_cache_wlock_invalidate`	
System Variable Name	`query_cache_wlock_invalidate`	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	`boolean`
	Default	`FALSE`

Normally, when one client acquires a `WRITE` lock on a `MyISAM` table, other clients are not blocked from issuing statements that read from the table if the query results are present in the query cache. Setting this variable to 1 causes acquisition of a `WRITE` lock for a table to invalidate any queries in the query cache that refer to the table. This forces other clients that attempt to access the table to wait while the lock is in effect.

- `query_prealloc_size`

Command-Line Format	`--query_prealloc_size=#`	
Option-File Format	`query_prealloc_size`	
System Variable Name	`query_prealloc_size`	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Platform Bit Size	`32`
	Type	`numeric`
	Default	`8192`
	Range	`8192 .. 4294967295`
	Block Size	`1024`

	Permitted Values	
	Platform Bit Size	64
	Type	numeric
	Default	8192
	Range	8192 .. 18446744073709547520
	Block Size	1024

The size of the persistent buffer used for statement parsing and execution. This buffer is not freed between statements. If you are running complex queries, a larger query_prealloc_size value might be helpful in improving performance, because it can reduce the need for the server to perform memory allocation during query execution operations.

- rand_seed1

  The rand_seed1 and rand_seed2 variables exist as session variables only, and can be set but not read. The variables—but not their values—are shown in the output of SHOW VARIABLES.

  The purpose of these variables is to support replication of the RAND() function. For statements that invoke RAND(), the master passes two values to the slave, where they are used to seed the random number generator. The slave uses these values to set the session variables rand_seed1 and rand_seed2 so that RAND() on the slave generates the same value as on the master.

- rand_seed2

  See the description for rand_seed1.

- range_alloc_block_size

Command-Line Format	--range_alloc_block_size=#	
Option-File Format	range_alloc_block_size	
System Variable Name	range_alloc_block_size	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Platform Bit Size	32
	Type	numeric
	Default	4096
	Range	4096 .. 4294967295
	Block Size	1024
	Permitted Values	
	Platform Bit Size	64
	Type	numeric

	Default	`4096`
	Range	`4096 .. 18446744073709547520`
	Block Size	`1024`

The size of blocks that are allocated when doing range optimization.

- `read_buffer_size`

Command-Line Format	`--read_buffer_size=#`	
Option-File Format	`read_buffer_size`	
System Variable Name	`read_buffer_size`	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	`numeric`
	Default	`131072`
	Range	`8200 .. 2147479552`

Each thread that does a sequential scan for a `MyISAM` table allocates a buffer of this size (in bytes) for each table it scans. If you do many sequential scans, you might want to increase this value, which defaults to 131072. The value of this variable should be a multiple of 4KB. If it is set to a value that is not a multiple of 4KB, its value will be rounded down to the nearest multiple of 4KB.

This option is also used in the following context for all search engines:

- For caching the indexes in a temporary file (not a temporary table), when sorting rows for `ORDER BY`.
- For bulk insert into partitions.
- For caching results of nested queries.

and in one other storage engine-specific way: to determine the memory block size for `MEMORY` tables.

The maximum permissible setting for `read_buffer_size` is 2GB.

For more information about memory use during different operations, see Section 8.11.4.1, “How MySQL Uses Memory”.

- `read_only`

Command-Line Format	`--read-only`	
Option-File Format	`read_only`	
System Variable Name	`read_only`	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	`boolean`
	Default	`false`

This variable is off by default. When it is enabled, the server permits no updates except from users that have the SUPER privilege or (on a slave server) from updates performed by slave threads. In replication setups, it can be useful to enable read_only on slave servers to ensure that slaves accept updates only from the master server and not from clients.

read_only does not apply to TEMPORARY tables, nor does it prevent the server from inserting rows into the log tables (see Section 5.2.1, “Selecting General Query and Slow Query Log Output Destinations”). This variable does not prevent the use of ANALYZE TABLE or OPTIMIZE TABLE statements because its purpose is to prevent changes to table structure or contents. Analysis and optimization do not qualify as such changes. This means, for example, that consistency checks on read-only slaves can be performed with mysqlcheck --all-databases --analyze.

read_only exists only as a GLOBAL variable, so changes to its value require the SUPER privilege. Changes to read_only on a master server are not replicated to slave servers. The value can be set on a slave server independent of the setting on the master.

Important

In MySQL 5.7, enabling read_only prevents the use of the SET PASSWORD statement by any user not having the SUPER privilege. This is not necessarily the case for all MySQL release series. When replicating from one MySQL release series to another (for example, from a MySQL 5.0 master to a MySQL 5.1 or later slave), you should check the documentation for the versions running on both master and slave to determine whether the behavior of read_only in this regard is or is not the same, and, if it is different, whether this has an impact on your applications.

The following conditions apply:

- If you attempt to enable read_only while you have any explicit locks (acquired with LOCK TABLES) or have a pending transaction, an error occurs.

- If you attempt to enable read_only while other clients hold explicit table locks or have pending transactions, the attempt blocks until the locks are released and the transactions end. While the attempt to enable read_only is pending, requests by other clients for table locks or to begin transactions also block until read_only has been set.

- read_only can be enabled while you hold a global read lock (acquired with FLUSH TABLES WITH READ LOCK) because that does not involve table locks.

In MySQL 5.7, attempts to set read_only block for active transactions that hold metadata locks until those transactions end.

- read_rnd_buffer_size

Command-Line Format	--read_rnd_buffer_size=#	
Option-File Format	read_rnd_buffer_size	
System Variable Name	read_rnd_buffer_size	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	numeric

	Default	262144
	Range	1 .. 2147483647

This variable is used for reads from MyISAM tables, and, for any storage engine, for Multi-Range Read optimization.

When reading rows from a MyISAM table in sorted order following a key-sorting operation, the rows are read through this buffer to avoid disk seeks. See Section 8.2.1.15, “ORDER BY Optimization”. Setting the variable to a large value can improve ORDER BY performance by a lot. However, this is a buffer allocated for each client, so you should not set the global variable to a large value. Instead, change the session variable only from within those clients that need to run large queries.

The maximum permissible setting for read_rnd_buffer_size is 2GB.

For more information about memory use during different operations, see Section 8.11.4.1, “How MySQL Uses Memory”. For information about Multi-Range Read optimization, see Section 8.2.1.13, “Multi-Range Read Optimization”.

- relay_log_purge

Command-Line Format	--relay_log_purge	
Option-File Format	relay_log_purge	
System Variable Name	relay_log_purge	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	boolean
	Default	TRUE

Disables or enables automatic purging of relay log files as soon as they are not needed any more. The default value is 1 (ON).

- relay_log_space_limit

Command-Line Format	--relay_log_space_limit=#	
Option-File Format	relay_log_space_limit	
System Variable Name	relay_log_space_limit	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Platform Bit Size	32
	Type	numeric
	Default	0
	Range	0 .. 4294967295
	Permitted Values	

	Platform Bit Size	64
	Type	numeric
	Default	0
	Range	0 .. 18446744073709547520

The maximum amount of space to use for all relay logs.

- report_host

Command-Line Format	--report-host=host_name	
Option-File Format	report-host	
System Variable Name	report_host	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	string

The value of the --report-host option.

- report_password

Command-Line Format	--report-password=name	
Option-File Format	report-password	
System Variable Name	report_password	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	string

The value of the --report-password option. Not the same as the password used for the MySQL replication user account.

- report_port

Command-Line Format	--report-port=#	
Option-File Format	report-port	
System Variable Name	report_port	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	numeric
	Default	[slave_port]
	Range	0 .. 65535

The value of the --report-port option.

- `report_user`

Command-Line Format	`--report-user=name`	
Option-File Format	`report-user`	
System Variable Name	`report_user`	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	`string`

The value of the `--report-user` option. Not the same as the name for the MySQL replication user account.

- `rpl_semi_sync_master_enabled`

System Variable Name	`rpl_semi_sync_master_enabled`	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	`boolean`
	Default	`OFF`

Controls whether semisynchronous replication is enabled on the master. To enable or disable the plugin, set this variable to `ON` or `OFF` (or 1 or 0), respectively. The default is `OFF`.

This variable is available only if the master-side semisynchronous replication plugin is installed.

- `rpl_semi_sync_master_timeout`

System Variable Name	`rpl_semi_sync_master_timeout`	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	`numeric`
	Default	`10000`

A value in milliseconds that controls how long the master waits on a commit for acknowledgment from a slave before timing out and reverting to asynchronous replication. The default value is 10000 (10 seconds).

This variable is available only if the master-side semisynchronous replication plugin is installed.

- `rpl_semi_sync_master_trace_level`

System Variable Name	`rpl_semi_sync_master_trace_level`
Variable Scope	Global
Dynamic Variable	Yes

	Permitted Values	
	Type	`numeric`
	Default	`32`

The semisynchronous replication debug trace level on the master. Currently, four levels are defined:

- 1 = general level (for example, time function failures)
- 16 = detail level (more verbose information)
- 32 = net wait level (more information about network waits)
- 64 = function level (information about function entry and exit)

This variable is available only if the master-side semisynchronous replication plugin is installed.

- `rpl_semi_sync_master_wait_for_slave_count`

Introduced	5.7.3	
System Variable Name	`rpl_semi_sync_master_wait_for_slave_count`	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	`numeric`
	Default	`1`
	Range	`1 .. 65535`

The number of slave acknowledgments the master must receive per transaction before proceeding. The default is to proceed after receiving a single acknowledgment. Performance is best for small values of this variable.

This variable is available only if the master-side semisynchronous replication plugin is installed. It was added in MySQL 5.7.3.

- `rpl_semi_sync_master_wait_no_slave`

System Variable Name	`rpl_semi_sync_master_wait_no_slave`	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	`boolean`
	Default	`ON`

With semisynchronous replication, for each transaction, the master waits until timeout for acknowledgment of receipt from some semisynchronous slave. If no response occurs during this period, the master reverts to normal replication. This variable controls whether the master waits for the timeout to expire before reverting to normal replication even if the slave count drops to zero during the timeout period.

If the value is ON (the default), it is permissible for the slave count to drop to zero during the timeout period (for example, if slaves disconnect). The master still waits for the timeout, so as long as some slave reconnects and acknowledges the transaction within the timeout interval, semisynchronous replication continues.

If the value is OFF, the master reverts to normal replication if the slave count drops to zero during the timeout period.

This variable is available only if the master-side semisynchronous replication plugin is installed.

- rpl_semi_sync_master_wait_point

Introduced	5.7.2	
System Variable Name	rpl_semi_sync_master_wait_point	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	enumeration
	Default	AFTER_SYNC
	Valid Values	AFTER_SYNC
		AFTER_COMMIT

This variable controls the point at which a semisynchronous replication master waits for slave acknowledgment of transaction receipt before returning a status to the client that committed the transaction. These values are permitted:

- AFTER_SYNC (the default): The master writes each transaction to its binary log and the slave, and syncs the binary log to disk. The master waits for slave acknowledgment of transaction receipt after the sync. Upon receiving acknowledgment, the master commits the transaction to the storage engine and returns a result to the client, which then can proceed.

- AFTER_COMMIT: The master writes each transaction to its binary log and the slave, syncs the binary log, and commits the transaction to the storage engine. The master waits for slave acknowledgment of transaction receipt after the commit. Upon receiving acknowledgment, the master returns a result to the client, which then can proceed.

The replication characteristics of these settings differ as follows:

- With AFTER_SYNC, all clients see the committed transaction at the same time: After it has been acknowledged by the slave and committed to the storage engine on the master. Thus, all clients see the same data on the master.

  In the event of master failure, all transactions committed on the master have been replicated to the slave (saved to its relay log). A crash of the master and failover to the slave is lossless because the slave is up to date.

- With AFTER_COMMIT, the client issuing the transaction gets a return status only after the server commits to the storage engine and receives slave acknowledgment. After the commit and before slave acknowledgment, other clients can see the committed transaction before the committing client.

If something goes wrong such that the slave does not process the transaction, then in the event of a master crash and failover to the slave, it is possible that such clients will see a loss of data relative to what they saw on the master.

This variable is available only if the master-side semisynchronous replication plugin is installed.

rpl_semi_sync_master_wait_point was added in MySQL 5.7.2. For older versions, semisynchronous master behavior is equivalent to a setting of AFTER_COMMIT.

This change introduces a version compatibility constraint because it increments the semisynchronous interface version: Servers for MySQL 5.7.2 and up do not work with semisynchronous replication plugins from older versions, nor do servers from older versions work with semisynchronous replication plugins for MySQL 5.7.2 and up.

- rpl_semi_sync_slave_enabled

System Variable Name	rpl_semi_sync_slave_enabled	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	boolean
	Default	OFF

Controls whether semisynchronous replication is enabled on the slave. To enable or disable the plugin, set this variable to ON or OFF (or 1 or 0), respectively. The default is OFF.

This variable is available only if the slave-side semisynchronous replication plugin is installed.

- rpl_semi_sync_slave_trace_level

System Variable Name	rpl_semi_sync_slave_trace_level	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	numeric
	Default	32

The semisynchronous replication debug trace level on the slave. See rpl_semi_sync_master_trace_level for the permissible values.

This variable is available only if the slave-side semisynchronous replication plugin is installed.

- secure_auth

Command-Line Format	--secure-auth
Option-File Format	secure-auth
System Variable Name	secure_auth
Variable Scope	Global
Dynamic Variable	Yes

	Permitted Values	
	Type	boolean
	Default	ON

If this variable is enabled, the server blocks connections by clients that attempt to use accounts that have passwords stored in the old (pre-4.1) format.

Enable this variable to prevent all use of passwords employing the old format (and hence insecure communication over the network). This variable enabled by default.

Server startup fails with an error if this variable is enabled and the privilege tables are in pre-4.1 format. See Section C.5.2.4, “Client does not support authentication protocol”.

Note

Passwords that use the pre-4.1 hashing method are less secure than passwords that use the native password hashing method and should be avoided. Pre-4.1 passwords are deprecated and support for them will be removed in a future MySQL release.

- secure_file_priv

Command-Line Format	--secure-file-priv=path	
Option-File Format	secure-file-priv	
System Variable Name	secure_file_priv	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	string

By default, this variable is empty. If set to the name of a directory, it limits the effect of the LOAD_FILE() function and the LOAD DATA and SELECT ... INTO OUTFILE statements to work only with files in that directory.

- server_id

Command-Line Format	--server-id=#	
Option-File Format	server-id	
System Variable Name	server_id	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	numeric
	Default	0
	Range	0 .. 4294967295

The server ID, used in replication to give each master and slave a unique identity. This variable is set by the --server-id [2162] option. For each server participating in replication, you should pick a positive integer in the range from 1 to 2^{32} – 1 to act as that server's ID.

- session_track_schema

Introduced	5.7.4	
Command-Line Format	--session_track_schema=#	
Option-File Format	session_track_schema=#	
System Variable Name	session_track_schema	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	boolean
	Default	ON

The server can track changes to the default schema (database) name within the current session and make this information available to the client when changes occur. This variable controls whether notification occurs.

If notification is enabled, any setting of the default schema is reported, even if the new schema name is the same as the old.

For information about obtaining session state-change information within client programs, see Section 21.8.7.64, "mysql_session_track_get_first()".

This variable was added in MySQL 5.7.4.

- session_track_state_change

Introduced	5.7.4	
Command-Line Format	--session_track_state_change=#	
Option-File Format	session_track_state_change=#	
System Variable Name	session_track_state_change	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	boolean
	Default	OFF

Whether the server tracks changes to the session state and notifies the client when changes to state information occur. Session state consists of these values:

- The default schema (database)
- Session-specific values for system variables
- User-defined variables
- Temporary tables
- Prepared statements

If notification is enabled, any assignments to session state values are reported, even if the new values are the same as the old.

The `session_track_state_change` variable controls only notification of when changes occur, not what the changes are. To receive notification for changes to the default schema name and session system variable values, use the `session_track_schema` and `session_track_system_variables` system variables.

For information about obtaining session state-change information within client programs, see Section 21.8.7.64, “`mysql_session_track_get_first()`”.

This variable was added in MySQL 5.7.4.

- `session_track_system_variables`

Introduced	5.7.4	
Command-Line Format	`--session_track_system_variables=#`	
Option-File Format	`session_track_system_variables=#`	
System Variable Name	`session_track_system_variables`	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	`string`
	Default	`time_zone, autocommit, character_set_client, character_set_results, character_set_connection`

The server can track changes to the session system variables and make this information available to the client when changes occur. The variable value is a comma-separated list of variables for which to track changes. By default, notification is enabled for `time_zone`, `autocommit`, `character_set_client`, `character_set_results`, and `character_set_connection`. (The latter three variables are those affected by `SET NAMES`.)

The special value `*` causes the server to track changes to all session variables. If given, this value must be specified by itself without specific system variable names.

Notification occurs for all assignments to tracked session system variables, even if the new values are the same as the old.

For information about obtaining session state-change information within client programs, see Section 21.8.7.64, “`mysql_session_track_get_first()`”.

This variable was added in MySQL 5.7.4.

- `sha256_password_private_key_path`

System Variable Name	`sha256_password_private_key_path`	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	`file name`

	Default	private_key.pem

The path name of the RSA private key file for the sha256_password authentication plugin. If the file is named as a relative path, it is interpreted relative to the server data directory. The file must be in PEM format. Because this file stores a private key, its access mode should be restricted so that only the MySQL server can read it.

For information about sha256_password, including instructions for creating the RSA key files, see Section 6.3.9.4, “The SHA-256 Authentication Plugin”.

This variable is available only if MySQL was built using OpenSSL.

- sha256_password_public_key_path

System Variable Name	sha256_password_public_key_path	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	file name
	Default	public_key.pem

The path name of the RSA public key file for the sha256_password authentication plugin. If the file is named as a relative path, it is interpreted relative to the server data directory. The file must be in PEM format. Because this file stores a public key, copies can be freely distributed to client users. (Clients that explicitly specify a public key when connecting to the server using RSA password encryption must use the same public key as that used by the server.)

For information about sha256_password, including instructions for creating the RSA key files and how clients specify the RSA public key, see Section 6.3.9.4, “The SHA-256 Authentication Plugin”.

This variable is available only if MySQL was built using OpenSSL.

- shared_memory

System Variable Name	shared_memory
Variable Scope	Global
Dynamic Variable	No
Platform Specific	Windows

(Windows only.) Whether the server permits shared-memory connections.

- shared_memory_base_name

System Variable Name	shared_memory_base_name
Variable Scope	Global
Dynamic Variable	No
Platform Specific	Windows

(Windows only.) The name of shared memory to use for shared-memory connections. This is useful when running multiple MySQL instances on a single physical machine. The default name is MYSQL. The name is case sensitive.

- `skip_external_locking`

Command-Line Format	`--skip-external-locking`	
Option-File Format	`skip_external_locking`	
System Variable Name	`skip_external_locking`	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	`boolean`
	Default	`ON`

This is `OFF` if `mysqld` uses external locking (system locking), `ON` if external locking is disabled. This affects only `MyISAM` table access.

This variable is set by the `--external-locking` or `--skip-external-locking` option. External locking has been disabled by default as of MySQL 4.0.

External locking affects only `MyISAM` table access. For more information, including conditions under which it can and cannot be used, see Section 8.10.5, “External Locking”.

- `skip_name_resolve`

Command-Line Format	`--skip-name-resolve`	
Option-File Format	`skip-name-resolve`	
System Variable Name	`skip_name_resolve`	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	`boolean`
	Default	`OFF`

This variable is set from the value of the `--skip-name-resolve` option. If it is `ON`, `mysqld` resolves host names when checking client connections. If `OFF`, `mysqld` uses only IP numbers and all `Host` column values in the grant tables must be IP addresses or `localhost`. See Section 8.11.5.2, “DNS Lookup Optimization and the Host Cache”.

- `skip_networking`

Command-Line Format	`--skip-networking`
Option-File Format	`skip-networking`
System Variable Name	`skip_networking`
Variable Scope	Global
Dynamic Variable	No

This is `ON` if the server permits only local (non-TCP/IP) connections. On Unix, local connections use a Unix socket file. On Windows, local connections use a named pipe or shared memory. This variable can be set to `ON` with the `--skip-networking` option.

- `skip_show_database`

Command-Line Format	`--skip-show-database`
Option-File Format	`skip-show-database`
System Variable Name	`skip_show_database`
Variable Scope	Global
Dynamic Variable	No

This prevents people from using the `SHOW DATABASES` statement if they do not have the `SHOW DATABASES` privilege. This can improve security if you have concerns about users being able to see databases belonging to other users. Its effect depends on the `SHOW DATABASES` privilege: If the variable value is `ON`, the `SHOW DATABASES` statement is permitted only to users who have the `SHOW DATABASES` privilege, and the statement displays all database names. If the value is `OFF`, `SHOW DATABASES` is permitted to all users, but displays the names of only those databases for which the user has the `SHOW DATABASES` or other privilege. (Note that *any* global privilege is considered a privilege for the database.)

- `slow_launch_time`

Command-Line Format	`--slow_launch_time=#`	
Option-File Format	`slow_launch_time`	
System Variable Name	`slow_launch_time`	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	`numeric`
	Default	`2`

If creating a thread takes longer than this many seconds, the server increments the `Slow_launch_threads` status variable.

- `slow_query_log`

Command-Line Format	`--slow-query-log`	
Option-File Format	`slow-query-log`	
System Variable Name	`slow_query_log`	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	`boolean`
	Default	`OFF`

Whether the slow query log is enabled. The value can be 0 (or `OFF`) to disable the log or 1 (or `ON`) to enable the log. The default value depends on whether the `--slow_query_log` option is given. The destination for log output is controlled by the `log_output` system variable; if that value is `NONE`, no log entries are written even if the log is enabled.

"Slow" is determined by the value of the `long_query_time` variable. See Section 5.2.5, "The Slow Query Log".

- `slow_query_log_file`

Command-Line Format	`--slow-query-log-file=file_name`	
Option-File Format	`slow_query_log_file`	
System Variable Name	`slow_query_log_file`	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	`file name`
	Default	`host_name-slow.log`

The name of the slow query log file. The default value is *`host_name`*`-slow.log`, but the initial value can be changed with the `--slow_query_log_file` option.

- `socket`

Command-Line Format	`--socket=name`	
Option-File Format	`socket`	
System Variable Name	`socket`	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	`file name`
	Default	`/tmp/mysql.sock`

On Unix platforms, this variable is the name of the socket file that is used for local client connections. The default is `/tmp/mysql.sock`. (For some distribution formats, the directory might be different, such as `/var/lib/mysql` for RPMs.)

On Windows, this variable is the name of the named pipe that is used for local client connections. The default value is `MySQL` (not case sensitive).

- `sort_buffer_size`

Command-Line Format	`--sort_buffer_size=#`	
Option-File Format	`sort_buffer_size`	
System Variable Name	`sort_buffer_size`	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Platform Bit Size	`32`

	Type (Other)	numeric
	Default	262144
	Range	32768 .. 4294967295
	Permitted Values	
	Platform Bit Size	64
	Type (Other)	numeric
	Default	262144
	Range	32768 .. 18446744073709551615
	Permitted Values	
	Type (Windows)	numeric
	Default	262144
	Range	32768 .. 4294967295

Each session that needs to do a sort allocates a buffer of this size. sort_buffer_size is not specific to any storage engine and applies in a general manner for optimization. See Section 8.2.1.15, "ORDER BY Optimization", for example.

If you see many Sort_merge_passes per second in SHOW GLOBAL STATUS output, you can consider increasing the sort_buffer_size value to speed up ORDER BY or GROUP BY operations that cannot be improved with query optimization or improved indexing.

The optimizer tries to work out how much space is needed but can allocate more, up to the limit. Setting it larger than required globally will slow down most queries that sort. It is best to increase it as a session setting, and only for the sessions that need a larger size. On Linux, there are thresholds of 256KB and 2MB where larger values may significantly slow down memory allocation, so you should consider staying below one of those values. Experiment to find the best value for your workload. See Section C.5.4.4, "Where MySQL Stores Temporary Files".

The maximum permissible setting for sort_buffer_size is 4GB–1. Larger values are permitted for 64-bit platforms (except 64-bit Windows, for which large values are truncated to 4GB–1 with a warning).

- sql_auto_is_null

System Variable Name	sql_auto_is_null	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	boolean
	Default	0

If this variable is set to 1, then after a statement that successfully inserts an automatically generated AUTO_INCREMENT value, you can find that value by issuing a statement of the following form:

```
SELECT * FROM tbl_name WHERE auto_col IS NULL
```

If the statement returns a row, the value returned is the same as if you invoked the LAST_INSERT_ID() function. For details, including the return value after a multiple-row insert, see Section 12.14, “Information Functions”. If no AUTO_INCREMENT value was successfully inserted, the SELECT statement returns no row.

The behavior of retrieving an AUTO_INCREMENT value by using an IS NULL comparison is used by some ODBC programs, such as Access. See Obtaining Auto-Increment Values. This behavior can be disabled by setting sql_auto_is_null to 0.

The default value of sql_auto_is_null is 0 in MySQL 5.7.

- sql_big_selects

System Variable Name	sql_big_selects	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	boolean
	Default	1

If set to 0, MySQL aborts SELECT statements that are likely to take a very long time to execute (that is, statements for which the optimizer estimates that the number of examined rows exceeds the value of max_join_size). This is useful when an inadvisable WHERE statement has been issued. The default value for a new connection is 1, which permits all SELECT statements.

If you set the max_join_size system variable to a value other than DEFAULT, sql_big_selects is set to 0.

- sql_buffer_result

System Variable Name	sql_buffer_result	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	boolean
	Default	0

If set to 1, sql_buffer_result forces results from SELECT statements to be put into temporary tables. This helps MySQL free the table locks early and can be beneficial in cases where it takes a long time to send results to the client. The default value is 0.

- sql_log_bin

System Variable Name	sql_log_bin	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	boolean

This variable controls whether logging to the binary log is done. The default value is 1 (do logging). To change logging for the current session, change the session value of this variable. The session user must have the SUPER privilege to set this variable.

In MySQL 5.7, it is not possible to set @@session.sql_log_bin within a transaction or subquery. (Bug #53437)

- sql_log_off

System Variable Name	sql_log_off	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	boolean
	Default	0

This variable controls whether logging to the general query log is done. The default value is 0 (do logging). To change logging for the current session, change the session value of this variable. The session user must have the SUPER privilege to set this option. The default value is 0.

- sql_mode

Command-Line Format	--sql-mode=name	
Option-File Format	sql-mode	
System Variable Name	sql_mode	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	set
	Default	NO_ENGINE_SUBSTITUTION
	Valid Values	ALLOW_INVALID_DATES
		ANSI_QUOTES
		ERROR_FOR_DIVISION_BY_ZERO
		HIGH_NOT_PRECEDENCE
		IGNORE_SPACE
		NO_AUTO_CREATE_USER
		NO_AUTO_VALUE_ON_ZERO
		NO_BACKSLASH_ESCAPES
		NO_DIR_IN_CREATE
		NO_ENGINE_SUBSTITUTION
		NO_FIELD_OPTIONS
		NO_KEY_OPTIONS
		NO_TABLE_OPTIONS
		NO_UNSIGNED_SUBTRACTION

		NO_ZERO_DATE
		NO_ZERO_IN_DATE
		ONLY_FULL_GROUP_BY
		PAD_CHAR_TO_FULL_LENGTH
		PIPES_AS_CONCAT
		REAL_AS_FLOAT
		STRICT_ALL_TABLES
		STRICT_TRANS_TABLES

The current server SQL mode, which can be set dynamically. See Section 5.1.7, “Server SQL Modes”.

> **Note**
>
> MySQL installation programs may configure the SQL mode during the installation process. For example, mysql_install_db creates a default option file named my.cnf in the base installation directory. This file contains a line that sets the SQL mode; see Section 4.4.3, “mysql_install_db — Initialize MySQL Data Directory”.
>
> If the SQL mode differs from the default or from what you expect, check for a setting in an option file that the server reads at startup.

- sql_notes

  If set to 1 (the default), warnings of Note level increment warning_count and the server records them. If set to 0, Note warnings do not increment warning_count and the server does not record them. mysqldump includes output to set this variable to 0 so that reloading the dump file does not produce warnings for events that do not affect the integrity of the reload operation.

- sql_quote_show_create

  If set to 1 (the default), the server quotes identifiers for SHOW CREATE TABLE and SHOW CREATE DATABASE statements. If set to 0, quoting is disabled. This option is enabled by default so that replication works for identifiers that require quoting. See Section 13.7.5.10, “SHOW CREATE TABLE Syntax”, and Section 13.7.5.6, “SHOW CREATE DATABASE Syntax”.

- sql_safe_updates

  If set to 1, MySQL aborts UPDATE or DELETE statements that do not use a key in the WHERE clause or a LIMIT clause. (Specifically, UPDATE statements must have a WHERE clause that uses a key or a LIMIT clause, or both. DELETE statements must have both.) This makes it possible to catch UPDATE or DELETE statements where keys are not used properly and that would probably change or delete a large number of rows. The default value is 0.

- sql_select_limit

System Variable Name	sql_select_limit	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	numeric

The maximum number of rows to return from SELECT statements. The default value for a new connection is the maximum number of rows that the server permits per table. Typical default values are (2^{32})–1 or (2^{64})–1. If you have changed the limit, the default value can be restored by assigning a value of DEFAULT.

If a SELECT has a LIMIT clause, the LIMIT takes precedence over the value of sql_select_limit.

- sql_warnings

  This variable controls whether single-row INSERT statements produce an information string if warnings occur. The default is 0. Set the value to 1 to produce an information string.

- ssl_ca

Command-Line Format	--ssl-ca=name	
Option-File Format	ssl-ca	
System Variable Name	ssl_ca	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	file name

  The path to a file with a list of trusted SSL CAs.

- ssl_capath

Command-Line Format	--ssl-capath=name	
Option-File Format	ssl-capath	
System Variable Name	ssl_capath	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	directory name

  The path to a directory that contains trusted SSL CA certificates in PEM format.

- ssl_cert

Command-Line Format	--ssl-cert=name	
Option-File Format	ssl-cert	
System Variable Name	ssl_cert	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	file name

  The name of the SSL certificate file to use for establishing a secure connection.

- `ssl_cipher`

Command-Line Format	`--ssl-cipher=name`	
Option-File Format	`ssl-cipher`	
System Variable Name	`ssl_cipher`	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	`string`

A list of permissible ciphers to use for SSL encryption.

- `ssl_crl`

Command-Line Format	`--ssl-crl=name`	
Option-File Format	`ssl-crl`	
System Variable Name	`ssl_crl`	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	`file name`

The path to a file containing certificate revocation lists in PEM format. Revocation lists work for MySQL distributions compiled against OpenSSL (but not yaSSL).

- `ssl_crlpath`

Command-Line Format	`--ssl-crlpath=name`	
Option-File Format	`ssl-crlpath`	
System Variable Name	`ssl_crlpath`	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	`directory name`

The path to a directory that contains files containing certificate revocation lists in PEM format. Revocation lists work for MySQL distributions compiled against OpenSSL (but not yaSSL).

- `ssl_key`

Command-Line Format	`--ssl-key=name`
Option-File Format	`ssl-key`
System Variable Name	`ssl_key`
Variable Scope	Global
Dynamic Variable	No
	Permitted Values

	Type	string

The name of the SSL key file to use for establishing a secure connection.

- storage_engine

Removed	5.7.5	
System Variable Name	storage_engine	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	enumeration
	Default	InnoDB

This variable is deprecated and was removed in MySQL 5.7.5. Use default_storage_engine instead.

- stored_program_cache

Command-Line Format	--stored-program-cache=#	
Option-File Format	stored_program_cache	
System Variable Name	stored_program_cache	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	numeric
	Default	256
	Range	256 .. 524288

Sets a soft upper limit for the number of cached stored routines per connection. The value of this variable is specified in terms of the number of stored routines held in each of the two caches maintained by the MySQL Server for, respectively, stored procedures and stored functions.

Whenever a stored routine is executed this cache size is checked before the first or top-level statement in the routine is parsed; if the number of routines of the same type (stored procedures or stored functions according to which is being executed) exceeds the limit specified by this variable, the corresponding cache is flushed and memory previously allocated for cached objects is freed. This allows the cache to be flushed safely, even when there are dependencies between stored routines.

- sync_frm

Command-Line Format	--sync-frm	
Option-File Format	sync_frm	
System Variable Name	sync_frm	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	

	Type	boolean
	Default	TRUE

If this variable is set to 1, when any nontemporary table is created its .frm file is synchronized to disk (using fdatasync()). This is slower but safer in case of a crash. The default is 1.

- system_time_zone

System Variable Name	system_time_zone	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	string

The server system time zone. When the server begins executing, it inherits a time zone setting from the machine defaults, possibly modified by the environment of the account used for running the server or the startup script. The value is used to set system_time_zone. Typically the time zone is specified by the TZ environment variable. It also can be specified using the --timezone option of the mysqld_safe script.

The system_time_zone variable differs from time_zone. Although they might have the same value, the latter variable is used to initialize the time zone for each client that connects. See Section 10.6, “MySQL Server Time Zone Support”.

- table_definition_cache

System Variable Name	table_definition_cache	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	numeric
	Default	-1 (autosized)
	Range	400 .. 524288

The number of table definitions (from .frm files) that can be stored in the definition cache. If you use a large number of tables, you can create a large table definition cache to speed up opening of tables. The table definition cache takes less space and does not use file descriptors, unlike the normal table cache. The minimum value is 400. The default value is based on the following formula, capped to a limit of 2000:

```
400 + (table_open_cache / 2)
```

For InnoDB, table_definition_cache acts as a soft limit for the number of open table instances in the InnoDB data dictionary cache. If the number of open table instances exceeds the table_definition_cache setting, the LRU mechanism begins to mark table instances for eviction and eventually removes them from the data dictionary cache. The limit helps address situations in which significant amounts of memory would be used to cache rarely used table instances until the next server restart. The number of table instances with cached metadata could be higher than the limit defined by table_definition_cache, because InnoDB system table instances and parent and child table instances with foreign key relationships are not placed on the LRU list and are not subject to eviction from memory.

Additionally, table_definition_cache defines a soft limit for the number of InnoDB file-per-table tablespaces that can be open at one time, which is also controlled by innodb_open_files. If both table_definition_cache and innodb_open_files are set, the highest setting is used. If neither variable is set, table_definition_cache, which has a higher default value, is used. If the number of open tablespace file handles exceeds the limit defined by table_definition_cache or innodb_open_files, the LRU mechanism searches the tablespace file LRU list for files that are fully flushed and are not currently being extended. This process is performed each time a new tablespace is opened. If there are no “inactive” tablespaces, no tablespace files are closed.

- table_open_cache

System Variable Name	table_open_cache	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	numeric
	Default	2000 (autosized)
	Range	1 .. 524288

The number of open tables for all threads. Increasing this value increases the number of file descriptors that mysqld requires. You can check whether you need to increase the table cache by checking the Opened_tables status variable. See Section 5.1.6, “Server Status Variables”. If the value of Opened_tables is large and you do not use FLUSH TABLES often (which just forces all tables to be closed and reopened), then you should increase the value of the table_open_cache variable. For more information about the table cache, see Section 8.4.3.1, “How MySQL Opens and Closes Tables”.

- table_open_cache_instances

System Variable Name	table_open_cache_instances	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	numeric
	Default	1

The number of open tables cache instances (default 1). To improve scalability by reducing contention among sessions, the open tables cache can be partitioned into several smaller cache instances of size table_open_cache / table_open_cache_instances . A session needs to lock only one instance to access it for DML statements. This segments cache access among instances, permitting higher performance for operations that use the cache when there are many sessions accessing tables. (DDL statements still require a lock on the entire cache, but such statements are much less frequent than DML statements.)

A value of 8 or 16 is recommended on systems that routinely use 16 or more cores.

- thread_cache_size

Command-Line Format	`--thread_cache_size=#`	
Option-File Format	`thread_cache_size`	
System Variable Name	`thread_cache_size`	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	`numeric`
	Default	`-1 (autosized)`
	Range	`0 .. 16384`

How many threads the server should cache for reuse. When a client disconnects, the client's threads are put in the cache if there are fewer than `thread_cache_size` threads there. Requests for threads are satisfied by reusing threads taken from the cache if possible, and only when the cache is empty is a new thread created. This variable can be increased to improve performance if you have a lot of new connections. Normally, this does not provide a notable performance improvement if you have a good thread implementation. However, if your server sees hundreds of connections per second you should normally set `thread_cache_size` high enough so that most new connections use cached threads. By examining the difference between the `Connections` and `Threads_created` status variables, you can see how efficient the thread cache is. For details, see Section 5.1.6, “Server Status Variables”.

The default value is based on the following formula, capped to a limit of 100:

```
8 + (max_connections / 100)
```

This variable has no effect for the embedded server (`libmysqld`) and as of MySQL 5.7.2 is no longer visible within the embedded server.

- `thread_concurrency`

Deprecated	5.6.1	
Removed	5.7.2	
Command-Line Format	`--thread_concurrency=#`	
Option-File Format	`thread_concurrency`	
System Variable Name	`thread_concurrency`	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	`numeric`
	Default	`10`
	Range	`1 .. 512`

This variable is specific to Solaris 8 and earlier systems, for which `mysqld` invokes the `thr_setconcurrency()` function with the variable value. This function enables applications to give the threads system a hint about the desired number of threads that should be run at the same time. Current Solaris versions document this as having no effect.

This variable was removed in MySQL 5.7.2.

- `thread_handling`

Command-Line Format	`--thread_handling=name`	
Option-File Format	`thread_handling`	
System Variable Name	`thread_handling`	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	`enumeration`
	Default	`one-thread-per-connection`
	Valid Values	`no-threads`
		`one-thread-per-connection`
		`dynamically-loaded`

The thread-handling model used by the server for connection threads. The permissible values are `no-threads` (the server uses a single thread to handle one connection) and `one-thread-per-connection` (the server uses one thread to handle each client connection). `no-threads` is useful for debugging under Linux; see Section 22.4, “Debugging and Porting MySQL”.

This variable has no effect for the embedded server (`libmysqld`) and as of MySQL 5.7.2 is no longer visible within the embedded server.

- `thread_stack`

Command-Line Format	`--thread_stack=#`	
Option-File Format	`thread_stack`	
System Variable Name	`thread_stack`	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Platform Bit Size	`32`
	Type	`numeric`
	Default	`196608`
	Range	`131072 .. 4294967295`
	Block Size	`1024`
	Permitted Values	
	Platform Bit Size	`64`
	Type	`numeric`
	Default	`262144`

	Range	131072 .. 18446744073709547520
	Block Size	1024

The stack size for each thread. Many of the limits detected by the `crash-me` test are dependent on this value. See Section 8.12.2, “The MySQL Benchmark Suite”. The default of 192KB (256KB for 64-bit systems) is large enough for normal operation. If the thread stack size is too small, it limits the complexity of the SQL statements that the server can handle, the recursion depth of stored procedures, and other memory-consuming actions.

- `time_format`

  This variable is unused. It is deprecated and will be removed in a future MySQL release.

- `time_zone`

System Variable Name	`time_zone`	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	`string`

  The current time zone. This variable is used to initialize the time zone for each client that connects. By default, the initial value of this is `'SYSTEM'` (which means, “use the value of `system_time_zone`”). The value can be specified explicitly at server startup with the `--default-time-zone` option. See Section 10.6, “MySQL Server Time Zone Support”.

- `timed_mutexes`

Deprecated	5.5.36	
Command-Line Format	`--timed_mutexes`	
Option-File Format	`timed_mutexes`	
System Variable Name	`timed_mutexes`	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	`boolean`
	Default	`OFF`

  In MySQL 5.7, this variable is deprecated; it has no use. It will be removed in a future MySQL release.

- `timestamp`

System Variable Name	`timestamp`	
Variable Scope	Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	`numeric`

Set the time for this client. This is used to get the original timestamp if you use the binary log to restore rows. *`timestamp_value`* should be a Unix epoch timestamp (a value like that returned by `UNIX_TIMESTAMP()`, not a value in `'YYYY-MM-DD hh:mm:ss'` format) or `DEFAULT`.

Setting `timestamp` to a constant value causes it to retain that value until it is changed again. Setting `timestamp` to `DEFAULT` causes its value to be the current date and time as of the time it is accessed.

In MySQL 5.7, `timestamp` is a `DOUBLE` rather than `BIGINT` because its value includes a microseconds part.

`SET timestamp` affects the value returned by `NOW()` but not by `SYSDATE()`. This means that timestamp settings in the binary log have no effect on invocations of `SYSDATE()`. The server can be started with the `--sysdate-is-now` option to cause `SYSDATE()` to be an alias for `NOW()`, in which case `SET timestamp` affects both functions.

- `tmp_table_size`

Command-Line Format	`--tmp_table_size=#`	
Option-File Format	`tmp_table_size`	
System Variable Name	`tmp_table_size`	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	`numeric`
	Default	`16777216`
	Range	`1024 .. 18446744073709551615`

The maximum size of internal in-memory temporary tables. (The actual limit is determined as the minimum of `tmp_table_size` and `max_heap_table_size`.) If an in-memory temporary table exceeds the limit, MySQL automatically converts it to an on-disk `MyISAM` table. Increase the value of `tmp_table_size` (and `max_heap_table_size` if necessary) if you do many advanced `GROUP BY` queries and you have lots of memory. This variable does not apply to user-created `MEMORY` tables.

You can compare the number of internal on-disk temporary tables created to the total number of internal temporary tables created by comparing the values of the `Created_tmp_disk_tables` and `Created_tmp_tables` variables.

See also Section 8.4.4, "How MySQL Uses Internal Temporary Tables".

- `tmpdir`

Command-Line Format	`--tmpdir=path`
	`-t`
Option-File Format	`tmpdir`
System Variable Name	`tmpdir`
Variable Scope	Global
Dynamic Variable	No
	Permitted Values

	Type	directory name

The directory used for temporary files and temporary tables. This variable can be set to a list of several paths that are used in round-robin fashion. Paths should be separated by colon characters (“:”) on Unix and semicolon characters (“;”) on Windows.

The multiple-directory feature can be used to spread the load between several physical disks. If the MySQL server is acting as a replication slave, you should not set tmpdir to point to a directory on a memory-based file system or to a directory that is cleared when the server host restarts. A replication slave needs some of its temporary files to survive a machine restart so that it can replicate temporary tables or LOAD DATA INFILE operations. If files in the temporary file directory are lost when the server restarts, replication fails. You can set the slave's temporary directory using the slave_load_tmpdir variable. In that case, the slave will not use the general tmpdir value and you can set tmpdir to a nonpermanent location.

- transaction_alloc_block_size

Command-Line Format	--transaction_alloc_block_size=#	
Option-File Format	transaction_alloc_block_size	
System Variable Name	transaction_alloc_block_size	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Platform Bit Size	32
	Type	numeric
	Default	8192
	Range	1024 .. 4294967295
	Block Size	1024
	Permitted Values	
	Platform Bit Size	64
	Type	numeric
	Default	8192
	Range	1024 .. 18446744073709547520
	Block Size	1024

The amount in bytes by which to increase a per-transaction memory pool which needs memory. See the description of transaction_prealloc_size.

- transaction_prealloc_size

Command-Line Format	--transaction_prealloc_size=#
Option-File Format	transaction_prealloc_size
System Variable Name	transaction_prealloc_size

Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Platform Bit Size	32
	Type	numeric
	Default	4096
	Range	1024 .. 4294967295
	Block Size	1024
	Permitted Values	
	Platform Bit Size	64
	Type	numeric
	Default	4096
	Range	1024 .. 18446744073709547520
	Block Size	1024

There is a per-transaction memory pool from which various transaction-related allocations take memory. The initial size of the pool in bytes is `transaction_prealloc_size`. For every allocation that cannot be satisfied from the pool because it has insufficient memory available, the pool is increased by `transaction_alloc_block_size` bytes. When the transaction ends, the pool is truncated to `transaction_prealloc_size` bytes.

By making `transaction_prealloc_size` sufficiently large to contain all statements within a single transaction, you can avoid many `malloc()` calls.

- `tx_isolation`

System Variable Name	`tx_isolation`	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	enumeration
	Default	REPEATABLE-READ
	Valid Values	READ-UNCOMMITTED
		READ-COMMITTED
		REPEATABLE-READ
		SERIALIZABLE

The default transaction isolation level. Defaults to `REPEATABLE-READ`.

This variable can be set directly, or indirectly using the `SET TRANSACTION` statement. See Section 13.3.6, “`SET TRANSACTION` Syntax”. If you set `tx_isolation` directly to an isolation level name that contains a space, the name should be enclosed within quotation marks, with the space replaced by a dash. For example:

```
SET tx_isolation = 'READ-COMMITTED';
```

Any unique prefix of a valid value may be used to set the value of this variable.

The default transaction isolation level can also be set at startup using the `--transaction-isolation` server option.

- `tx_read_only`

System Variable Name	`tx_read_only`	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	`boolean`
	Default	`OFF`

The default transaction access mode. The value can be `OFF` (read/write, the default) or `ON` (read only).

This variable can be set directly, or indirectly using the `SET TRANSACTION` statement. See Section 13.3.6, “`SET TRANSACTION` Syntax”.

To set the default transaction access mode at startup, use the `--transaction-read-only` server option.

- `unique_checks`

System Variable Name	`unique_checks`	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	`boolean`
	Default	`1`

If set to 1 (the default), uniqueness checks for secondary indexes in `InnoDB` tables are performed. If set to 0, storage engines are permitted to assume that duplicate keys are not present in input data. If you know for certain that your data does not contain uniqueness violations, you can set this to 0 to speed up large table imports to `InnoDB`.

Note that setting this variable to 0 does not *require* storage engines to ignore duplicate keys. An engine is still permitted to check for them and issue duplicate-key errors if it detects them.

- `updatable_views_with_limit`

Command-Line Format	`--updatable_views_with_limit=#`
Option-File Format	`updatable_views_with_limit`
System Variable Name	`updatable_views_with_limit`
Variable Scope	Global, Session

Dynamic Variable	Yes	
	Permitted Values	
	Type	boolean
	Default	1

This variable controls whether updates to a view can be made when the view does not contain all columns of the primary key defined in the underlying table, if the update statement contains a LIMIT clause. (Such updates often are generated by GUI tools.) An update is an UPDATE or DELETE statement. Primary key here means a PRIMARY KEY, or a UNIQUE index in which no column can contain NULL.

The variable can have two values:

- 1 or YES: Issue a warning only (not an error message). This is the default value.
- 0 or NO: Prohibit the update.

- validate_password_*xxx*

  The validate_password plugin implements a set of system variables having names of the form validate_password_*xxx*. These variables affect password testing by that plugin; see Password Validation Plugin Options and Variables.

- validate_user_plugins

System Variable Name	validate_user_plugins	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	boolean
	Default	ON

  If this variable is enabled (the default), the server checks each user account and produces a warning if conditions are found that would make the account unusable:

  - The account requires an authentication plugin that is not loaded.
  - The account requires the sha256_password authentication plugin but the server was started with neither SSL nor RSA enabled as required by this plugin.

  Enabling validate_user_plugins slows down server initialization and FLUSH PRIVILEGES. If you do not require the additional checking, you can disable this variable at startup to avoid the performance decrement.

  This variable was added in MySQL 5.7.1.

- version

  The version number for the server. The value might also include a suffix indicating server build or configuration information. -log indicates that one or more of the general log, slow query log, or binary log are enabled. -debug indicates that the server was built with debugging support enabled.

- version_comment

System Variable Name	version_comment	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	string

The CMake configuration program has a COMPILATION_COMMENT option that permits a comment to be specified when building MySQL. This variable contains the value of that comment. See Section 2.8.4, “MySQL Source-Configuration Options”.

- version_compile_machine

System Variable Name	version_compile_machine	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	string

The type of the server binary.

- version_compile_os

System Variable Name	version_compile_os	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	string

The type of operating system on which MySQL was built.

- wait_timeout

Command-Line Format	--wait_timeout=#	
Option-File Format	wait_timeout	
System Variable Name	wait_timeout	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type (Other)	numeric
	Default	28800
	Range	1 .. 31536000
	Permitted Values	
	Type (Windows)	numeric

	Default	28800
	Range	1 .. 2147483

The number of seconds the server waits for activity on a noninteractive connection before closing it.

On thread startup, the session wait_timeout value is initialized from the global wait_timeout value or from the global interactive_timeout value, depending on the type of client (as defined by the CLIENT_INTERACTIVE connect option to mysql_real_connect()). See also interactive_timeout.

- warning_count

  The number of errors, warnings, and notes that resulted from the last statement that generated messages. This variable is read only. See Section 13.7.5.39, "SHOW WARNINGS Syntax".

5.1.5 Using System Variables

The MySQL server maintains many system variables that indicate how it is configured. Section 5.1.4, "Server System Variables", describes the meaning of these variables. Each system variable has a default value. System variables can be set at server startup using options on the command line or in an option file. Most of them can be changed dynamically while the server is running by means of the SET statement, which enables you to modify operation of the server without having to stop and restart it. You can refer to system variable values in expressions.

The server maintains two kinds of system variables. Global variables affect the overall operation of the server. Session variables affect its operation for individual client connections. A given system variable can have both a global and a session value. Global and session system variables are related as follows:

- When the server starts, it initializes all global variables to their default values. These defaults can be changed by options specified on the command line or in an option file. (See Section 4.2.3, "Specifying Program Options".)

- The server also maintains a set of session variables for each client that connects. The client's session variables are initialized at connect time using the current values of the corresponding global variables. For example, the client's SQL mode is controlled by the session sql_mode value, which is initialized when the client connects to the value of the global sql_mode value.

System variable values can be set globally at server startup by using options on the command line or in an option file. When you use a startup option to set a variable that takes a numeric value, the value can be given with a suffix of K, M, or G (either uppercase or lowercase) to indicate a multiplier of 1024, 1024^2 or 1024^3; that is, units of kilobytes, megabytes, or gigabytes, respectively. Thus, the following command starts the server with a query cache size of 16 megabytes and a maximum packet size of one gigabyte:

```
mysqld --query_cache_size=16M --max_allowed_packet=1G
```

Within an option file, those variables are set like this:

```
[mysqld]
query_cache_size=16M
max_allowed_packet=1G
```

The lettercase of suffix letters does not matter; 16M and 16m are equivalent, as are 1G and 1g.

If you want to restrict the maximum value to which a system variable can be set at runtime with the SET statement, you can specify this maximum by using an option of the form --maximum-*var_name*=*value* at server startup. For example, to prevent the value of query_cache_size from being increased to more than 32MB at runtime, use the option --maximum-query_cache_size=32M.

Many system variables are dynamic and can be changed while the server runs by using the SET statement. For a list, see Section 5.1.5.2, "Dynamic System Variables". To change a system variable with SET, refer to it as *var_name*, optionally preceded by a modifier:

- To indicate explicitly that a variable is a global variable, precede its name by GLOBAL or @@global.. The SUPER privilege is required to set global variables.
- To indicate explicitly that a variable is a session variable, precede its name by SESSION, @@session., or @@. Setting a session variable requires no special privilege, but a client can change only its own session variables, not those of any other client.
- LOCAL and @@local. are synonyms for SESSION and @@session..
- If no modifier is present, SET changes the session variable.

A SET statement can contain multiple variable assignments, separated by commas. If you set several system variables, the most recent GLOBAL or SESSION modifier in the statement is used for following variables that have no modifier specified.

Examples:

```
SET sort_buffer_size=10000;
SET @@local.sort_buffer_size=10000;
SET GLOBAL sort_buffer_size=1000000, SESSION sort_buffer_size=1000000;
SET @@sort_buffer_size=1000000;
SET @@global.sort_buffer_size=1000000, @@local.sort_buffer_size=1000000;
```

The @@*var_name* syntax for system variables is supported for compatibility with some other database systems.

If you change a session system variable, the value remains in effect until your session ends or until you change the variable to a different value. The change is not visible to other clients.

If you change a global system variable, the value is remembered and used for new connections until the server restarts. (To make a global system variable setting permanent, you should set it in an option file.) The change is visible to any client that accesses that global variable. However, the change affects the corresponding session variable only for clients that connect after the change. The global variable change does not affect the session variable for any client that is currently connected (not even that of the client that issues the SET GLOBAL statement).

To prevent incorrect usage, MySQL produces an error if you use SET GLOBAL with a variable that can only be used with SET SESSION or if you do not specify GLOBAL (or @@global.) when setting a global variable.

To set a SESSION variable to the GLOBAL value or a GLOBAL value to the compiled-in MySQL default value, use the DEFAULT keyword. For example, the following two statements are identical in setting the session value of max_join_size to the global value:

```
SET max_join_size=DEFAULT;
SET @@session.max_join_size=@@global.max_join_size;
```

Not all system variables can be set to DEFAULT. In such cases, use of DEFAULT results in an error.

You can refer to the values of specific global or session system variables in expressions by using one of the @@-modifiers. For example, you can retrieve values in a SELECT statement like this:

```
SELECT @@global.sql_mode, @@session.sql_mode, @@sql_mode;
```

When you refer to a system variable in an expression as `@@var_name` (that is, when you do not specify `@@global.` or `@@session.`), MySQL returns the session value if it exists and the global value otherwise. (This differs from `SET @@var_name = value`, which always refers to the session value.)

Note

Some variables displayed by `SHOW VARIABLES` may not be available using `SELECT @@var_name` syntax; an `Unknown system variable` occurs. As a workaround in such cases, you can use `SHOW VARIABLES LIKE 'var_name'`.

Suffixes for specifying a value multiplier can be used when setting a variable at server startup, but not to set the value with `SET` at runtime. On the other hand, with `SET` you can assign a variable's value using an expression, which is not true when you set a variable at server startup. For example, the first of the following lines is legal at server startup, but the second is not:

```
shell> mysql --max_allowed_packet=16M
shell> mysql --max_allowed_packet=16*1024*1024
```

Conversely, the second of the following lines is legal at runtime, but the first is not:

```
mysql> SET GLOBAL max_allowed_packet=16M;
mysql> SET GLOBAL max_allowed_packet=16*1024*1024;
```

Note

Some system variables can be enabled with the `SET` statement by setting them to `ON` or `1`, or disabled by setting them to `OFF` or `0`. However, to set such a variable on the command line or in an option file, you must set it to `1` or `0`; setting it to `ON` or `OFF` will not work. For example, on the command line, `--delay_key_write=1` works but `--delay_key_write=ON` does not.

To display system variable names and values, use the `SHOW VARIABLES` statement:

```
mysql> SHOW VARIABLES;
+---------------------------------+-----------------------------------+
| Variable_name                   | Value                             |
+---------------------------------+-----------------------------------+
| auto_increment_increment        | 1                                 |
| auto_increment_offset           | 1                                 |
| automatic_sp_privileges         | ON                                |
| back_log                        | 50                                |
| basedir                         | /home/mysql/                      |
| binlog_cache_size               | 32768                             |
| bulk_insert_buffer_size         | 8388608                           |
| character_set_client            | latin1                            |
| character_set_connection        | latin1                            |
| character_set_database          | latin1                            |
| character_set_results           | latin1                            |
| character_set_server            | latin1                            |
| character_set_system            | utf8                              |
| character_sets_dir              | /home/mysql/share/mysql/charsets/ |
| collation_connection            | latin1_swedish_ci                 |
| collation_database              | latin1_swedish_ci                 |
| collation_server                | latin1_swedish_ci                 |
...
| innodb_autoextend_increment     | 8                                 |
| innodb_buffer_pool_size         | 8388608                           |
| innodb_checksums                | ON                                 |
| innodb_commit_concurrency       | 0                                  |
| innodb_concurrency_tickets      | 500                                |
| innodb_data_file_path           | ibdata1:10M:autoextend             |
| innodb_data_home_dir            |                                    |
...
| version                         | 5.1.6-alpha-log                    |
| version_comment                 | Source distribution                |
| version_compile_machine         | i686                               |
| version_compile_os              | suse-linux                         |
| wait_timeout                    | 28800                              |
+---------------------------------+------------------------------------+
```

With a LIKE clause, the statement displays only those variables that match the pattern. To obtain a specific variable name, use a LIKE clause as shown:

```
SHOW VARIABLES LIKE 'max_join_size';
SHOW SESSION VARIABLES LIKE 'max_join_size';
```

To get a list of variables whose name match a pattern, use the "%" wildcard character in a LIKE clause:

```
SHOW VARIABLES LIKE '%size%';
SHOW GLOBAL VARIABLES LIKE '%size%';
```

Wildcard characters can be used in any position within the pattern to be matched. Strictly speaking, because "_" is a wildcard that matches any single character, you should escape it as "_" to match it literally. In practice, this is rarely necessary.

For SHOW VARIABLES, if you specify neither GLOBAL nor SESSION, MySQL returns SESSION values.

The reason for requiring the GLOBAL keyword when setting GLOBAL-only variables but not when retrieving them is to prevent problems in the future. If we were to remove a SESSION variable that has the same name as a GLOBAL variable, a client with the SUPER privilege might accidentally change the GLOBAL variable rather than just the SESSION variable for its own connection. If we add a SESSION variable with the same name as a GLOBAL variable, a client that intends to change the GLOBAL variable might find only its own SESSION variable changed.

5.1.5.1 Structured System Variables

A structured variable differs from a regular system variable in two respects:

- Its value is a structure with components that specify server parameters considered to be closely related.
- There might be several instances of a given type of structured variable. Each one has a different name and refers to a different resource maintained by the server.

MySQL 5.7 supports one structured variable type, which specifies parameters governing the operation of key caches. A key cache structured variable has these components:

- key_buffer_size
- key_cache_block_size
- key_cache_division_limit
- key_cache_age_threshold

This section describes the syntax for referring to structured variables. Key cache variables are used for syntax examples, but specific details about how key caches operate are found elsewhere, in Section 8.9.2, "The MyISAM Key Cache".

To refer to a component of a structured variable instance, you can use a compound name in *instance_name.component_name* format. Examples:

```
hot_cache.key_buffer_size
hot_cache.key_cache_block_size
cold_cache.key_cache_block_size
```

For each structured system variable, an instance with the name of default is always predefined. If you refer to a component of a structured variable without any instance name, the default instance is used. Thus, default.key_buffer_size and key_buffer_size both refer to the same system variable.

Structured variable instances and components follow these naming rules:

- For a given type of structured variable, each instance must have a name that is unique *within* variables of that type. However, instance names need not be unique *across* structured variable types. For example, each structured variable has an instance named default, so default is not unique across variable types.

- The names of the components of each structured variable type must be unique across all system variable names. If this were not true (that is, if two different types of structured variables could share component member names), it would not be clear which default structured variable to use for references to member names that are not qualified by an instance name.

- If a structured variable instance name is not legal as an unquoted identifier, refer to it as a quoted identifier using backticks. For example, hot-cache is not legal, but `hot-cache` is.

- global, session, and local are not legal instance names. This avoids a conflict with notation such as @@global.*var_name* for referring to nonstructured system variables.

Currently, the first two rules have no possibility of being violated because the only structured variable type is the one for key caches. These rules will assume greater significance if some other type of structured variable is created in the future.

With one exception, you can refer to structured variable components using compound names in any context where simple variable names can occur. For example, you can assign a value to a structured variable using a command-line option:

```
shell> mysqld --hot_cache.key_buffer_size=64K
```

In an option file, use this syntax:

```
[mysqld]
hot_cache.key_buffer_size=64K
```

If you start the server with this option, it creates a key cache named hot_cache with a size of 64KB in addition to the default key cache that has a default size of 8MB.

Suppose that you start the server as follows:

```
shell> mysqld --key_buffer_size=256K \
         --extra_cache.key_buffer_size=128K \
         --extra_cache.key_cache_block_size=2048
```

In this case, the server sets the size of the default key cache to 256KB. (You could also have written --default.key_buffer_size=256K.) In addition, the server creates a second key cache named extra_cache that has a size of 128KB, with the size of block buffers for caching table index blocks set to 2048 bytes.

The following example starts the server with three different key caches having sizes in a 3:1:1 ratio:

```
shell> mysqld --key_buffer_size=6M \
         --hot_cache.key_buffer_size=2M \
         --cold_cache.key_buffer_size=2M
```

Structured variable values may be set and retrieved at runtime as well. For example, to set a key cache named hot_cache to a size of 10MB, use either of these statements:

```
mysql> SET GLOBAL hot_cache.key_buffer_size = 10*1024*1024;
mysql> SET @@global.hot_cache.key_buffer_size = 10*1024*1024;
```

To retrieve the cache size, do this:

```
mysql> SELECT @@global.hot_cache.key_buffer_size;
```

However, the following statement does not work. The variable is not interpreted as a compound name, but as a simple string for a LIKE pattern-matching operation:

```
mysql> SHOW GLOBAL VARIABLES LIKE 'hot_cache.key_buffer_size';
```

This is the exception to being able to use structured variable names anywhere a simple variable name may occur.

5.1.5.2 Dynamic System Variables

Many server system variables are dynamic and can be set at runtime using SET GLOBAL or SET SESSION. You can also obtain their values using SELECT. See Section 5.1.5, “Using System Variables”.

The following table shows the full list of all dynamic system variables. The last column indicates for each variable whether GLOBAL or SESSION (or both) apply. The table also lists session options that can be set with the SET statement. Section 5.1.4, “Server System Variables”, discusses these options.

Variables that have a type of “string” take a string value. Variables that have a type of “numeric” take a numeric value. Variables that have a type of “boolean” can be set to 0, 1, ON or OFF. (If you set them on the command line or in an option file, use the numeric values.) Variables that are marked as “enumeration” normally should be set to one of the available values for the variable, but can also be set to the number that corresponds to the desired enumeration value. For enumerated system variables, the first enumeration value corresponds to 0. This differs from ENUM columns, for which the first enumeration value corresponds to 1.

Table 5.3 Dynamic Variable Summary

Variable Name	Variable Type	Variable Scope
auto_increment_increment	numeric	GLOBAL | SESSION
auto_increment_offset	numeric	GLOBAL | SESSION
autocommit	boolean	GLOBAL | SESSION
automatic_sp_privileges	boolean	GLOBAL
big_tables	boolean	GLOBAL | SESSION
binlog_cache_size	numeric	GLOBAL
binlog_checksum	string	GLOBAL
binlog_direct_non_transactional_updates	boolean	GLOBAL | SESSION
binlog_format	enumeration	GLOBAL | SESSION

Variable Name	Variable Type	Variable Scope
binlog_max_flush_queue_time	numeric	GLOBAL
binlog_order_commits	boolean	GLOBAL
binlog_row_image=image_type	enumeration	GLOBAL | SESSION
binlog_rows_query_log_events	boolean	GLOBAL | SESSION
binlog_stmt_cache_size	numeric	GLOBAL
block_encryption_mode	string	GLOBAL | SESSION
bulk_insert_buffer_size	numeric	GLOBAL | SESSION
character_set_client	string	GLOBAL | SESSION
character_set_connection	string	GLOBAL | SESSION
character_set_database	string	GLOBAL | SESSION
character_set_filesystem	string	GLOBAL | SESSION
character_set_results	string	GLOBAL | SESSION
character_set_server	string	GLOBAL | SESSION
collation_connection	string	GLOBAL | SESSION
collation_database	string	GLOBAL | SESSION
collation_server	string	GLOBAL | SESSION
completion_type	numeric	GLOBAL | SESSION
concurrent_insert	boolean	GLOBAL
connect_timeout	numeric	GLOBAL
debug	string	GLOBAL | SESSION
debug_sync	string	SESSION
default_password_lifetime	integer	GLOBAL
default_storage_engine	enumeration	GLOBAL | SESSION
default_tmp_storage_engine	enumeration	GLOBAL | SESSION
default_week_format	numeric	GLOBAL | SESSION
delay_key_write	enumeration	GLOBAL
delayed_insert_limit	numeric	GLOBAL
delayed_insert_timeout	numeric	GLOBAL
delayed_queue_size	numeric	GLOBAL
div_precision_increment	numeric	GLOBAL | SESSION
end_markers_in_json	boolean	GLOBAL | SESSION
eq_range_index_dive_limit	numeric	GLOBAL | SESSION
event_scheduler	enumeration	GLOBAL
expire_logs_days	numeric	GLOBAL
flush	boolean	GLOBAL
flush_time	numeric	GLOBAL
foreign_key_checks	boolean	GLOBAL | SESSION
ft_boolean_syntax	string	GLOBAL

Variable Name	Variable Type	Variable Scope
general_log	boolean	GLOBAL
general_log_file	filename	GLOBAL
group_concat_max_len	numeric	GLOBAL | SESSION
gtid_next	enumeration	SESSION
gtid_purged	string	GLOBAL
host_cache_size	numeric	GLOBAL
identity	numeric	SESSION
init_connect	string	GLOBAL
init_slave	string	GLOBAL
innodb_adaptive_flushing	boolean	GLOBAL
innodb_adaptive_flushing_lwm	numeric	GLOBAL
innodb_adaptive_hash_index	boolean	GLOBAL
innodb_adaptive_max_sleep_delay	numeric	GLOBAL
innodb_api_bk_commit_interval	numeric	GLOBAL
innodb_api_trx_level	numeric	GLOBAL
innodb_autoextend_increment	numeric	GLOBAL
innodb_buffer_pool_dump_at_shutdown	boolean	GLOBAL
innodb_buffer_pool_dump_now	boolean	GLOBAL
innodb_buffer_pool_dump_pct	numeric	GLOBAL
innodb_buffer_pool_filename	filename	GLOBAL
innodb_buffer_pool_load_abort	boolean	GLOBAL
innodb_buffer_pool_load_now	boolean	GLOBAL
innodb_change_buffer_max_size	numeric	GLOBAL
innodb_change_buffering	enumeration	GLOBAL
innodb_checksum_algorithm	enumeration	GLOBAL
innodb_cmp_per_index_enabled	boolean	GLOBAL
innodb_commit_concurrency	numeric	GLOBAL
innodb_compression_failure_threshold_pct	numeric	GLOBAL
innodb_compression_level	numeric	GLOBAL
innodb_compression_pad_pct_max	numeric	GLOBAL
innodb_concurrency_tickets	numeric	GLOBAL
innodb_disable_sort_file_cache	boolean	GLOBAL
innodb_fast_shutdown	numeric	GLOBAL
innodb_file_format	string	GLOBAL
innodb_file_format_max	string	GLOBAL
innodb_file_per_table	boolean	GLOBAL
innodb_flush_log_at_timeout	numeric	GLOBAL
innodb_flush_log_at_trx_commit	enumeration	GLOBAL

Variable Name	Variable Type	Variable Scope
innodb_flush_neighbors	enumeration	GLOBAL
innodb_flushing_avg_loops	numeric	GLOBAL
innodb_ft_aux_table	string	GLOBAL
innodb_ft_enable_diag_print	boolean	GLOBAL
innodb_ft_enable_stopword	boolean	GLOBAL
innodb_ft_num_word_optimize	numeric	GLOBAL
innodb_ft_result_cache_limit	numeric	GLOBAL
innodb_ft_server_stopword_table	string	GLOBAL
innodb_ft_user_stopword_table	string	GLOBAL | SESSION
innodb_io_capacity	numeric	GLOBAL
innodb_io_capacity_max	numeric	GLOBAL
innodb_large_prefix	boolean	GLOBAL
innodb_lock_wait_timeout	numeric	GLOBAL | SESSION
innodb_log_compressed_pages	boolean	GLOBAL
innodb_log_write_ahead_size	numeric	GLOBAL
innodb_lru_scan_depth	numeric	GLOBAL
innodb_max_dirty_pages_pct	numeric	GLOBAL
innodb_max_dirty_pages_pct_lwm	numeric	GLOBAL
innodb_max_purge_lag	numeric	GLOBAL
innodb_max_purge_lag_delay	numeric	GLOBAL
innodb_monitor_disable	string	GLOBAL
innodb_monitor_enable	string	GLOBAL
innodb_monitor_reset	string	GLOBAL
innodb_monitor_reset_all	string	GLOBAL
innodb_old_blocks_pct	numeric	GLOBAL
innodb_old_blocks_time	numeric	GLOBAL
innodb_online_alter_log_max_size	numeric	GLOBAL
innodb_optimize_fulltext_only	boolean	GLOBAL
innodb_print_all_deadlocks	boolean	GLOBAL
innodb_purge_batch_size	numeric	GLOBAL
innodb_random_read_ahead	boolean	GLOBAL
innodb_read_ahead_threshold	numeric	GLOBAL
innodb_replication_delay	numeric	GLOBAL
innodb_rollback_segments	numeric	GLOBAL
innodb_spin_wait_delay	numeric	GLOBAL
innodb_stats_auto_recalc	boolean	GLOBAL
innodb_stats_method	enumeration	GLOBAL
innodb_stats_on_metadata	boolean	GLOBAL

Variable Name	Variable Type	Variable Scope
innodb_stats_persistent	boolean	GLOBAL
innodb_stats_persistent_sample_pages	numeric	GLOBAL
innodb_stats_sample_pages	numeric	GLOBAL
innodb_stats_transient_sample_pages	numeric	GLOBAL
innodb_status_output	boolean	GLOBAL
innodb_status_output_locks	boolean	GLOBAL
innodb_strict_mode	boolean	GLOBAL | SESSION
innodb_support_xa	boolean	GLOBAL | SESSION
innodb_sync_spin_loops	numeric	GLOBAL
innodb_table_locks	boolean	GLOBAL | SESSION
innodb_thread_concurrency	numeric	GLOBAL
innodb_thread_sleep_delay	numeric	GLOBAL
innodb_undo_logs	numeric	GLOBAL
insert_id	numeric	SESSION
interactive_timeout	numeric	GLOBAL | SESSION
join_buffer_size	numeric	GLOBAL | SESSION
keep_files_on_create	boolean	GLOBAL | SESSION
key_buffer_size	numeric	GLOBAL
key_cache_age_threshold	numeric	GLOBAL
key_cache_block_size	numeric	GLOBAL
key_cache_division_limit	numeric	GLOBAL
last_insert_id	numeric	SESSION
lc_messages	string	GLOBAL | SESSION
lc_time_names	string	GLOBAL | SESSION
local_infile	boolean	GLOBAL
lock_wait_timeout	numeric	GLOBAL | SESSION
log_bin_trust_function_creators	boolean	GLOBAL
log_error_verbosity	numeric	GLOBAL
log_output	set	GLOBAL
log_queries_not_using_indexes	boolean	GLOBAL
log_slow_admin_statements	boolean	GLOBAL
log_slow_slave_statements	boolean	GLOBAL
log_throttle_queries_not_using_indexes	numeric	GLOBAL
log_timestamps	enumeration	GLOBAL
log_warnings	numeric	GLOBAL
long_query_time	numeric	GLOBAL | SESSION
low_priority_updates	boolean	GLOBAL | SESSION
master_info_repository	string	GLOBAL

Variable Name	Variable Type	Variable Scope
master_verify_checksum	boolean	GLOBAL
max_allowed_packet	numeric	GLOBAL
max_binlog_cache_size	numeric	GLOBAL
max_binlog_size	numeric	GLOBAL
max_binlog_stmt_cache_size	numeric	GLOBAL
max_connect_errors	numeric	GLOBAL
max_connections	numeric	GLOBAL
max_delayed_threads	numeric	GLOBAL | SESSION
max_error_count	numeric	GLOBAL | SESSION
max_heap_table_size	numeric	GLOBAL | SESSION
max_insert_delayed_threads	numeric	GLOBAL | SESSION
max_join_size	numeric	GLOBAL | SESSION
max_length_for_sort_data	numeric	GLOBAL | SESSION
max_prepared_stmt_count	numeric	GLOBAL
max_relay_log_size	numeric	GLOBAL
max_seeks_for_key	numeric	GLOBAL | SESSION
max_sort_length	numeric	GLOBAL | SESSION
max_sp_recursion_depth	numeric	GLOBAL | SESSION
max_statement_time	numeric	GLOBAL | SESSION
max_user_connections	numeric	GLOBAL | SESSION
max_write_lock_count	numeric	GLOBAL
min_examined_row_limit	numeric	GLOBAL | SESSION
myisam_data_pointer_size	numeric	GLOBAL
myisam_max_sort_file_size	numeric	GLOBAL
myisam_repair_threads	numeric	GLOBAL | SESSION
myisam_sort_buffer_size	numeric	GLOBAL | SESSION
myisam_stats_method	enumeration	GLOBAL | SESSION
myisam_use_mmap	boolean	GLOBAL
net_buffer_length	numeric	GLOBAL | SESSION
net_read_timeout	numeric	GLOBAL | SESSION
net_retry_count	numeric	GLOBAL | SESSION
net_write_timeout	numeric	GLOBAL | SESSION
new	boolean	GLOBAL | SESSION
old_alter_table	boolean	GLOBAL | SESSION
old_passwords	boolean	GLOBAL | SESSION
optimizer_prune_level	boolean	GLOBAL | SESSION
optimizer_search_depth	numeric	GLOBAL | SESSION
optimizer_switch	set	GLOBAL | SESSION

Variable Name	Variable Type	Variable Scope
optimizer_trace	string	GLOBAL | SESSION
optimizer_trace_features	string	GLOBAL | SESSION
optimizer_trace_limit	numeric	GLOBAL | SESSION
optimizer_trace_max_mem_size	numeric	GLOBAL | SESSION
optimizer_trace_offset	numeric	GLOBAL | SESSION
preload_buffer_size	numeric	GLOBAL | SESSION
profiling	boolean	GLOBAL | SESSION
profiling_history_size	numeric	GLOBAL | SESSION
pseudo_slave_mode	numeric	SESSION
pseudo_thread_id	numeric	SESSION
query_alloc_block_size	numeric	GLOBAL | SESSION
query_cache_limit	numeric	GLOBAL
query_cache_min_res_unit	numeric	GLOBAL
query_cache_size	numeric	GLOBAL
query_cache_type	enumeration	GLOBAL | SESSION
query_cache_wlock_invalidate	boolean	GLOBAL | SESSION
query_prealloc_size	numeric	GLOBAL | SESSION
rand_seed1	numeric	SESSION
rand_seed2	numeric	SESSION
range_alloc_block_size	numeric	GLOBAL | SESSION
read_buffer_size	numeric	GLOBAL | SESSION
read_only	boolean	GLOBAL
read_rnd_buffer_size	numeric	GLOBAL | SESSION
relay_log_info_repository	string	GLOBAL
relay_log_purge	boolean	GLOBAL
rpl_semi_sync_master_enabled	boolean	GLOBAL
rpl_semi_sync_master_timeout	numeric	GLOBAL
rpl_semi_sync_master_trace_level	numeric	GLOBAL
rpl_semi_sync_master_wait_for_slave_count	numeric	GLOBAL
rpl_semi_sync_master_wait_no_slave	boolean	GLOBAL
rpl_semi_sync_master_wait_point	enumeration	GLOBAL
rpl_semi_sync_slave_enabled	boolean	GLOBAL
rpl_semi_sync_slave_trace_level	numeric	GLOBAL
rpl_stop_slave_timeout	integer	GLOBAL
secure_auth	boolean	GLOBAL
server_id [2162]	numeric	GLOBAL
session_track_schema	boolean	GLOBAL | SESSION
session_track_state_change	boolean	GLOBAL | SESSION

Variable Name	Variable Type	Variable Scope
session_track_system_variables	string	GLOBAL | SESSION
slave_allow_batching	boolean	GLOBAL
slave_checkpoint_group=#	numeric	GLOBAL
slave_checkpoint_period=#	numeric	GLOBAL
slave_compressed_protocol	boolean	GLOBAL
slave_exec_mode	enumeration	GLOBAL
slave_max_allowed_packet	numeric	GLOBAL
slave_net_timeout	numeric	GLOBAL
slave_parallel_type	enumeration	GLOBAL
slave_parallel_workers	numeric	GLOBAL
slave_pending_jobs_size_max	numeric	GLOBAL
slave_rows_search_algorithms=list	set	GLOBAL
slave_sql_verify_checksum	boolean	GLOBAL
slave_transaction_retries	numeric	GLOBAL
slow_launch_time	numeric	GLOBAL
slow_query_log	boolean	GLOBAL
slow_query_log_file	filename	GLOBAL
sort_buffer_size	numeric	GLOBAL | SESSION
sql_auto_is_null	boolean	GLOBAL | SESSION
sql_big_selects	boolean	GLOBAL | SESSION
sql_buffer_result	boolean	GLOBAL | SESSION
sql_log_bin	boolean	GLOBAL | SESSION
sql_log_off	boolean	GLOBAL | SESSION
sql_mode	set	GLOBAL | SESSION
sql_notes	boolean	GLOBAL | SESSION
sql_quote_show_create	boolean	GLOBAL | SESSION
sql_safe_updates	boolean	GLOBAL | SESSION
sql_select_limit	numeric	GLOBAL | SESSION
sql_slave_skip_counter	numeric	GLOBAL
sql_warnings	boolean	GLOBAL | SESSION
storage_engine	enumeration	GLOBAL | SESSION
stored_program_cache	numeric	GLOBAL
sync_binlog	numeric	GLOBAL
sync_frm	boolean	GLOBAL
sync_master_info	numeric	GLOBAL
sync_relay_log	numeric	GLOBAL
sync_relay_log_info	numeric	GLOBAL
table_definition_cache	numeric	GLOBAL

Variable Name	Variable Type	Variable Scope
table_open_cache	numeric	GLOBAL
thread_cache_size	numeric	GLOBAL
time_zone	string	GLOBAL | SESSION
timed_mutexes	boolean	GLOBAL
timestamp	numeric	SESSION
tmp_table_size	numeric	GLOBAL | SESSION
transaction_alloc_block_size	numeric	GLOBAL | SESSION
transaction_prealloc_size	numeric	GLOBAL | SESSION
tx_isolation	enumeration	GLOBAL | SESSION
tx_read_only	boolean	GLOBAL | SESSION
unique_checks	boolean	GLOBAL | SESSION
updatable_views_with_limit	boolean	GLOBAL | SESSION
validate_password_length	numeric	GLOBAL
validate_password_mixed_case_count	numeric	GLOBAL
validate_password_number_count	numeric	GLOBAL
validate_password_policy	enumeration	GLOBAL
validate_password_special_char_count	numeric	GLOBAL
wait_timeout	numeric	GLOBAL | SESSION

5.1.6 Server Status Variables

The server maintains many status variables that provide information about its operation. You can view these variables and their values by using the SHOW [GLOBAL | SESSION] STATUS statement (see Section 13.7.5.34, “SHOW STATUS Syntax”). The optional GLOBAL keyword aggregates the values over all connections, and SESSION shows the values for the current connection.

```
mysql> SHOW GLOBAL STATUS;
+-----------------------------------+------------+
| Variable_name                     | Value      |
+-----------------------------------+------------+
| Aborted_clients                   | 0          |
| Aborted_connects                  | 0          |
| Bytes_received                    | 155372598  |
| Bytes_sent                        | 1176560426 |
...
| Connections                       | 30023      |
| Created_tmp_disk_tables           | 0          |
| Created_tmp_files                 | 3          |
| Created_tmp_tables                | 2          |
...
| Threads_created                   | 217        |
| Threads_running                   | 88         |
| Uptime                            | 1389872    |
+-----------------------------------+------------+
```

Many status variables are reset to 0 by the FLUSH STATUS statement.

The following table lists all available server status variables:

Table 5.4 Status Variable Summary

Variable Name	Variable Type	Variable Scope
Aborted_clients	numeric	GLOBAL
Aborted_connects	numeric	GLOBAL
Binlog_cache_disk_use	numeric	GLOBAL
Binlog_cache_use	numeric	GLOBAL
Binlog_stmt_cache_disk_use	numeric	GLOBAL
Binlog_stmt_cache_use	numeric	GLOBAL
Bytes_received	numeric	GLOBAL | SESSION
Bytes_sent	numeric	GLOBAL | SESSION
Com_admin_commands	numeric	GLOBAL | SESSION
Com_alter_db	numeric	GLOBAL | SESSION
Com_alter_db_upgrade	numeric	GLOBAL | SESSION
Com_alter_event	numeric	GLOBAL | SESSION
Com_alter_function	numeric	GLOBAL | SESSION
Com_alter_procedure	numeric	GLOBAL | SESSION
Com_alter_server	numeric	GLOBAL | SESSION
Com_alter_table	numeric	GLOBAL | SESSION
Com_alter_tablespace	numeric	GLOBAL | SESSION
Com_alter_user	numeric	GLOBAL | SESSION
Com_analyze	numeric	GLOBAL | SESSION
Com_assign_to_keycache	numeric	GLOBAL | SESSION
Com_begin	numeric	GLOBAL | SESSION
Com_binlog	numeric	GLOBAL | SESSION
Com_call_procedure	numeric	GLOBAL | SESSION
Com_change_db	numeric	GLOBAL | SESSION
Com_change_master	numeric	GLOBAL | SESSION
Com_change_repl_filter	numeric	GLOBAL | SESSION
Com_check	numeric	GLOBAL | SESSION
Com_checksum	numeric	GLOBAL | SESSION
Com_commit	numeric	GLOBAL | SESSION
Com_create_db	numeric	GLOBAL | SESSION
Com_create_event	numeric	GLOBAL | SESSION
Com_create_function	numeric	GLOBAL | SESSION
Com_create_index	numeric	GLOBAL | SESSION
Com_create_procedure	numeric	GLOBAL | SESSION
Com_create_server	numeric	GLOBAL | SESSION
Com_create_table	numeric	GLOBAL | SESSION
Com_create_trigger	numeric	GLOBAL | SESSION

Variable Name	Variable Type	Variable Scope
Com_create_udf	numeric	GLOBAL | SESSION
Com_create_user	numeric	GLOBAL | SESSION
Com_create_view	numeric	GLOBAL | SESSION
Com_dealloc_sql	numeric	GLOBAL | SESSION
Com_delete	numeric	GLOBAL | SESSION
Com_delete_multi	numeric	GLOBAL | SESSION
Com_do	numeric	GLOBAL | SESSION
Com_drop_db	numeric	GLOBAL | SESSION
Com_drop_event	numeric	GLOBAL | SESSION
Com_drop_function	numeric	GLOBAL | SESSION
Com_drop_index	numeric	GLOBAL | SESSION
Com_drop_procedure	numeric	GLOBAL | SESSION
Com_drop_server	numeric	GLOBAL | SESSION
Com_drop_table	numeric	GLOBAL | SESSION
Com_drop_trigger	numeric	GLOBAL | SESSION
Com_drop_user	numeric	GLOBAL | SESSION
Com_drop_view	numeric	GLOBAL | SESSION
Com_empty_query	numeric	GLOBAL | SESSION
Com_execute_sql	numeric	GLOBAL | SESSION
Com_flush	numeric	GLOBAL | SESSION
Com_get_diagnostics	numeric	GLOBAL | SESSION
Com_grant	numeric	GLOBAL | SESSION
Com_ha_close	numeric	GLOBAL | SESSION
Com_ha_open	numeric	GLOBAL | SESSION
Com_ha_read	numeric	GLOBAL | SESSION
Com_help	numeric	GLOBAL | SESSION
Com_insert	numeric	GLOBAL | SESSION
Com_insert_select	numeric	GLOBAL | SESSION
Com_install_plugin	numeric	GLOBAL | SESSION
Com_kill	numeric	GLOBAL | SESSION
Com_load	numeric	GLOBAL | SESSION
Com_lock_tables	numeric	GLOBAL | SESSION
Com_optimize	numeric	GLOBAL | SESSION
Com_preload_keys	numeric	GLOBAL | SESSION
Com_prepare_sql	numeric	GLOBAL | SESSION
Com_purge	numeric	GLOBAL | SESSION
Com_purge_before_date	numeric	GLOBAL | SESSION
Com_release_savepoint	numeric	GLOBAL | SESSION

Variable Name	Variable Type	Variable Scope
Com_rename_table	numeric	GLOBAL | SESSION
Com_rename_user	numeric	GLOBAL | SESSION
Com_repair	numeric	GLOBAL | SESSION
Com_replace	numeric	GLOBAL | SESSION
Com_replace_select	numeric	GLOBAL | SESSION
Com_reset	numeric	GLOBAL | SESSION
Com_resignal	numeric	GLOBAL | SESSION
Com_revoke	numeric	GLOBAL | SESSION
Com_revoke_all	numeric	GLOBAL | SESSION
Com_rollback	numeric	GLOBAL | SESSION
Com_rollback_to_savepoint	numeric	GLOBAL | SESSION
Com_savepoint	numeric	GLOBAL | SESSION
Com_select	numeric	GLOBAL | SESSION
Com_set_option	numeric	GLOBAL | SESSION
Com_show_authors	numeric	GLOBAL | SESSION
Com_show_binlog_events	numeric	GLOBAL | SESSION
Com_show_binlogs	numeric	GLOBAL | SESSION
Com_show_charsets	numeric	GLOBAL | SESSION
Com_show_collations	numeric	GLOBAL | SESSION
Com_show_contributors	numeric	GLOBAL | SESSION
Com_show_create_db	numeric	GLOBAL | SESSION
Com_show_create_event	numeric	GLOBAL | SESSION
Com_show_create_func	numeric	GLOBAL | SESSION
Com_show_create_proc	numeric	GLOBAL | SESSION
Com_show_create_table	numeric	GLOBAL | SESSION
Com_show_create_trigger	numeric	GLOBAL | SESSION
Com_show_databases	numeric	GLOBAL | SESSION
Com_show_engine_logs	numeric	GLOBAL | SESSION
Com_show_engine_mutex	numeric	GLOBAL | SESSION
Com_show_engine_status	numeric	GLOBAL | SESSION
Com_show_errors	numeric	GLOBAL | SESSION
Com_show_events	numeric	GLOBAL | SESSION
Com_show_fields	numeric	GLOBAL | SESSION
Com_show_function_code	numeric	GLOBAL | SESSION
Com_show_function_status	numeric	GLOBAL | SESSION
Com_show_grants	numeric	GLOBAL | SESSION
Com_show_keys	numeric	GLOBAL | SESSION
Com_show_master_status	numeric	GLOBAL | SESSION

Variable Name	Variable Type	Variable Scope
Com_show_new_master	numeric	GLOBAL | SESSION
Com_show_open_tables	numeric	GLOBAL | SESSION
Com_show_plugins	numeric	GLOBAL | SESSION
Com_show_privileges	numeric	GLOBAL | SESSION
Com_show_procedure_code	numeric	GLOBAL | SESSION
Com_show_procedure_status	numeric	GLOBAL | SESSION
Com_show_processlist	numeric	GLOBAL | SESSION
Com_show_profile	numeric	GLOBAL | SESSION
Com_show_profiles	numeric	GLOBAL | SESSION
Com_show_relaylog_events	numeric	GLOBAL | SESSION
Com_show_slave_hosts	numeric	GLOBAL | SESSION
Com_show_slave_status	numeric	GLOBAL | SESSION
Com_show_status	numeric	GLOBAL | SESSION
Com_show_storage_engines	numeric	GLOBAL | SESSION
Com_show_table_status	numeric	GLOBAL | SESSION
Com_show_tables	numeric	GLOBAL | SESSION
Com_show_triggers	numeric	GLOBAL | SESSION
Com_show_variables	numeric	GLOBAL | SESSION
Com_show_warnings	numeric	GLOBAL | SESSION
Com_signal	numeric	GLOBAL | SESSION
Com_slave_start	numeric	GLOBAL | SESSION
Com_slave_stop	numeric	GLOBAL | SESSION
Com_stmt_close	numeric	GLOBAL | SESSION
Com_stmt_execute	numeric	GLOBAL | SESSION
Com_stmt_fetch	numeric	GLOBAL | SESSION
Com_stmt_prepare	numeric	GLOBAL | SESSION
Com_stmt_reprepare	numeric	GLOBAL | SESSION
Com_stmt_reset	numeric	GLOBAL | SESSION
Com_stmt_send_long_data	numeric	GLOBAL | SESSION
Com_truncate	numeric	GLOBAL | SESSION
Com_uninstall_plugin	numeric	GLOBAL | SESSION
Com_unlock_tables	numeric	GLOBAL | SESSION
Com_update	numeric	GLOBAL | SESSION
Com_update_multi	numeric	GLOBAL | SESSION
Com_xa_commit	numeric	GLOBAL | SESSION
Com_xa_end	numeric	GLOBAL | SESSION
Com_xa_prepare	numeric	GLOBAL | SESSION
Com_xa_recover	numeric	GLOBAL | SESSION

Variable Name	Variable Type	Variable Scope
Com_xa_rollback	numeric	GLOBAL | SESSION
Com_xa_start	numeric	GLOBAL | SESSION
Compression	numeric	SESSION
Connection_errors_accept	numeric	GLOBAL
Connection_errors_internal	numeric	GLOBAL
Connection_errors_max_connections	numeric	GLOBAL
Connection_errors_peer_addr	numeric	GLOBAL
Connection_errors_select	numeric	GLOBAL
Connection_errors_tcpwrap	numeric	GLOBAL
Connections	numeric	GLOBAL
Created_tmp_disk_tables	numeric	GLOBAL | SESSION
Created_tmp_files	numeric	GLOBAL
Created_tmp_tables	numeric	GLOBAL | SESSION
Delayed_errors	numeric	GLOBAL
Delayed_insert_threads	numeric	GLOBAL
Delayed_writes	numeric	GLOBAL
Flush_commands	numeric	GLOBAL
Handler_commit	numeric	GLOBAL | SESSION
Handler_delete	numeric	GLOBAL | SESSION
Handler_discover	numeric	GLOBAL | SESSION
Handler_external_lock	numeric	GLOBAL | SESSION
Handler_mrr_init	numeric	GLOBAL | SESSION
Handler_prepare	numeric	GLOBAL | SESSION
Handler_read_first	numeric	GLOBAL | SESSION
Handler_read_key	numeric	GLOBAL | SESSION
Handler_read_last	numeric	GLOBAL | SESSION
Handler_read_next	numeric	GLOBAL | SESSION
Handler_read_prev	numeric	GLOBAL | SESSION
Handler_read_rnd	numeric	GLOBAL | SESSION
Handler_read_rnd_next	numeric	GLOBAL | SESSION
Handler_rollback	numeric	GLOBAL | SESSION
Handler_savepoint	numeric	GLOBAL | SESSION
Handler_savepoint_rollback	numeric	GLOBAL | SESSION
Handler_update	numeric	GLOBAL | SESSION
Handler_write	numeric	GLOBAL | SESSION
Innodb_available_undo_logs	numeric	GLOBAL
Innodb_buffer_pool_bytes_data	numeric	GLOBAL
Innodb_buffer_pool_bytes_dirty	numeric	GLOBAL

Variable Name	Variable Type	Variable Scope
Innodb_buffer_pool_dump_status	numeric	GLOBAL
Innodb_buffer_pool_load_status	numeric	GLOBAL
Innodb_buffer_pool_pages_data	numeric	GLOBAL
Innodb_buffer_pool_pages_dirty	numeric	GLOBAL
Innodb_buffer_pool_pages_flushed	numeric	GLOBAL
Innodb_buffer_pool_pages_free	numeric	GLOBAL
Innodb_buffer_pool_pages_latched	numeric	GLOBAL
Innodb_buffer_pool_pages_misc	numeric	GLOBAL
Innodb_buffer_pool_pages_total	numeric	GLOBAL
Innodb_buffer_pool_read_ahead	numeric	GLOBAL
Innodb_buffer_pool_read_ahead_evicted	numeric	GLOBAL
Innodb_buffer_pool_read_requests	numeric	GLOBAL
Innodb_buffer_pool_reads	numeric	GLOBAL
Innodb_buffer_pool_wait_free	numeric	GLOBAL
Innodb_buffer_pool_write_requests	numeric	GLOBAL
Innodb_data_fsyncs	numeric	GLOBAL
Innodb_data_pending_fsyncs	numeric	GLOBAL
Innodb_data_pending_reads	numeric	GLOBAL
Innodb_data_pending_writes	numeric	GLOBAL
Innodb_data_read	numeric	GLOBAL
Innodb_data_reads	numeric	GLOBAL
Innodb_data_writes	numeric	GLOBAL
Innodb_data_written	numeric	GLOBAL
Innodb_dblwr_pages_written	numeric	GLOBAL
Innodb_dblwr_writes	numeric	GLOBAL
Innodb_have_atomic_builtins	numeric	GLOBAL
Innodb_log_waits	numeric	GLOBAL
Innodb_log_write_requests	numeric	GLOBAL
Innodb_log_writes	numeric	GLOBAL
Innodb_num_open_files	numeric	GLOBAL
Innodb_os_log_fsyncs	numeric	GLOBAL
Innodb_os_log_pending_fsyncs	numeric	GLOBAL
Innodb_os_log_pending_writes	numeric	GLOBAL
Innodb_os_log_written	numeric	GLOBAL
Innodb_page_size	numeric	GLOBAL
Innodb_pages_created	numeric	GLOBAL
Innodb_pages_read	numeric	GLOBAL
Innodb_pages_written	numeric	GLOBAL

Variable Name	Variable Type	Variable Scope
Innodb_row_lock_current_waits	numeric	GLOBAL
Innodb_row_lock_time	numeric	GLOBAL
Innodb_row_lock_time_avg	numeric	GLOBAL
Innodb_row_lock_time_max	numeric	GLOBAL
Innodb_row_lock_waits	numeric	GLOBAL
Innodb_rows_deleted	numeric	GLOBAL
Innodb_rows_inserted	numeric	GLOBAL
Innodb_rows_read	numeric	GLOBAL
Innodb_rows_updated	numeric	GLOBAL
Innodb_truncated_status_writes	numeric	GLOBAL
Key_blocks_not_flushed	numeric	GLOBAL
Key_blocks_unused	numeric	GLOBAL
Key_blocks_used	numeric	GLOBAL
Key_read_requests	numeric	GLOBAL
Key_reads	numeric	GLOBAL
Key_write_requests	numeric	GLOBAL
Key_writes	numeric	GLOBAL
Last_query_cost	numeric	SESSION
Last_query_partial_plans	numeric	SESSION
Max_statement_time_exceeded	numeric	GLOBAL | SESSION
Max_statement_time_set	numeric	GLOBAL | SESSION
Max_statement_time_set_failed	numeric	GLOBAL | SESSION
Max_used_connections	numeric	GLOBAL
Max_used_connections_time	datetime	GLOBAL
Ndb_conflict_fn_max	numeric	GLOBAL
Ndb_conflict_fn_old	numeric	GLOBAL
Ndb_number_of_data_nodes	numeric	GLOBAL
Not_flushed_delayed_rows	numeric	GLOBAL
Open_files	numeric	GLOBAL
Open_streams	numeric	GLOBAL
Open_table_definitions	numeric	GLOBAL
Open_tables	numeric	GLOBAL | SESSION
Opened_files	numeric	GLOBAL
Opened_table_definitions	numeric	GLOBAL | SESSION
Opened_tables	numeric	GLOBAL | SESSION
Performance_schema_accounts_lost	numeric	GLOBAL
Performance_schema_cond_classes_lost	numeric	GLOBAL
Performance_schema_cond_instances_lost	numeric	GLOBAL

Variable Name	Variable Type	Variable Scope
Performance_schema_digest_lost	numeric	GLOBAL
Performance_schema_file_classes_lost	numeric	GLOBAL
Performance_schema_file_handles_lost	numeric	GLOBAL
Performance_schema_file_instances_lost	numeric	GLOBAL
Performance_schema_hosts_lost	numeric	GLOBAL
Performance_schema_locker_lost	numeric	GLOBAL
Performance_schema_memory_classes_lost	numeric	GLOBAL
Performance_schema_metadata_lock_lost	numeric	GLOBAL
Performance_schema_mutex_classes_lost	numeric	GLOBAL
Performance_schema_mutex_instances_lost	numeric	GLOBAL
Performance_schema_nested_statement_lost	numeric	GLOBAL
Performance_schema_prepared_statements_lost	numeric	GLOBAL
Performance_schema_program_lost	numeric	GLOBAL
Performance_schema_rwlock_classes_lost	numeric	GLOBAL
Performance_schema_rwlock_instances_lost	numeric	GLOBAL
Performance_schema_session_connect_attrs_lost	numeric	GLOBAL
Performance_schema_socket_classes_lost	numeric	GLOBAL
Performance_schema_socket_instances_lost	numeric	GLOBAL
Performance_schema_stage_classes_lost	numeric	GLOBAL
Performance_schema_statement_classes_lost	numeric	GLOBAL
Performance_schema_table_handles_lost	numeric	GLOBAL
Performance_schema_table_instances_lost	numeric	GLOBAL
Performance_schema_thread_classes_lost	numeric	GLOBAL
Performance_schema_thread_instances_lost	numeric	GLOBAL
Performance_schema_users_lost	numeric	GLOBAL
Prepared_stmt_count	numeric	GLOBAL
Qcache_free_blocks	numeric	GLOBAL
Qcache_free_memory	numeric	GLOBAL
Qcache_hits	numeric	GLOBAL
Qcache_inserts	numeric	GLOBAL
Qcache_lowmem_prunes	numeric	GLOBAL
Qcache_not_cached	numeric	GLOBAL
Qcache_queries_in_cache	numeric	GLOBAL
Qcache_total_blocks	numeric	GLOBAL
Queries	numeric	GLOBAL | SESSION
Questions	numeric	GLOBAL | SESSION
Rpl_semi_sync_master_clients	numeric	GLOBAL
Rpl_semi_sync_master_net_avg_wait_time	numeric	GLOBAL

Variable Name	Variable Type	Variable Scope
Rpl_semi_sync_master_net_wait_time	numeric	GLOBAL
Rpl_semi_sync_master_net_waits	numeric	GLOBAL
Rpl_semi_sync_master_no_times	numeric	GLOBAL
Rpl_semi_sync_master_no_tx	numeric	GLOBAL
Rpl_semi_sync_master_status	boolean	GLOBAL
Rpl_semi_sync_master_timefunc_failures	numeric	GLOBAL
Rpl_semi_sync_master_tx_avg_wait_time	numeric	GLOBAL
Rpl_semi_sync_master_tx_wait_time	numeric	GLOBAL
Rpl_semi_sync_master_tx_waits	numeric	GLOBAL
Rpl_semi_sync_master_wait_pos_backtraverse	numeric	GLOBAL
Rpl_semi_sync_master_wait_sessions	numeric	GLOBAL
Rpl_semi_sync_master_yes_tx	numeric	GLOBAL
Rpl_semi_sync_slave_status	boolean	GLOBAL
Rsa_public_key	string	GLOBAL
Select_full_join	numeric	GLOBAL | SESSION
Select_full_range_join	numeric	GLOBAL | SESSION
Select_range	numeric	GLOBAL | SESSION
Select_range_check	numeric	GLOBAL | SESSION
Select_scan	numeric	GLOBAL | SESSION
Slave_heartbeat_period		GLOBAL
Slave_last_heartbeat		GLOBAL
Slave_open_temp_tables	numeric	GLOBAL
Slave_received_heartbeats		GLOBAL
Slave_retried_transactions	numeric	GLOBAL
Slave_running	boolean	GLOBAL
Slow_launch_threads	numeric	GLOBAL | SESSION
Slow_queries	numeric	GLOBAL | SESSION
Sort_merge_passes	numeric	GLOBAL | SESSION
Sort_range	numeric	GLOBAL | SESSION
Sort_rows	numeric	GLOBAL | SESSION
Sort_scan	numeric	GLOBAL | SESSION
Ssl_accept_renegotiates	numeric	GLOBAL
Ssl_accepts	numeric	GLOBAL
Ssl_callback_cache_hits	numeric	GLOBAL
Ssl_cipher	string	GLOBAL | SESSION
Ssl_cipher_list	string	GLOBAL | SESSION
Ssl_client_connects	numeric	GLOBAL
Ssl_connect_renegotiates	numeric	GLOBAL

Variable Name	Variable Type	Variable Scope
Ssl_ctx_verify_depth	numeric	GLOBAL
Ssl_ctx_verify_mode	numeric	GLOBAL
Ssl_default_timeout	numeric	GLOBAL | SESSION
Ssl_finished_accepts	numeric	GLOBAL
Ssl_finished_connects	numeric	GLOBAL
Ssl_server_not_after	numeric	GLOBAL | SESSION
Ssl_server_not_before	numeric	GLOBAL | SESSION
Ssl_session_cache_hits	numeric	GLOBAL
Ssl_session_cache_misses	numeric	GLOBAL
Ssl_session_cache_mode	string	GLOBAL
Ssl_session_cache_overflows	numeric	GLOBAL
Ssl_session_cache_size	numeric	GLOBAL
Ssl_session_cache_timeouts	numeric	GLOBAL
Ssl_sessions_reused	numeric	GLOBAL | SESSION
Ssl_used_session_cache_entries	numeric	GLOBAL
Ssl_verify_depth	numeric	GLOBAL | SESSION
Ssl_verify_mode	numeric	GLOBAL | SESSION
Ssl_version	string	GLOBAL | SESSION
Table_locks_immediate	numeric	GLOBAL
Table_locks_waited	numeric	GLOBAL
Table_open_cache_hits	numeric	GLOBAL | SESSION
Table_open_cache_misses	numeric	GLOBAL | SESSION
Table_open_cache_overflows	numeric	GLOBAL | SESSION
Tc_log_max_pages_used	numeric	GLOBAL
Tc_log_page_size	numeric	GLOBAL
Tc_log_page_waits	numeric	GLOBAL
Threads_cached	numeric	GLOBAL
Threads_connected	numeric	GLOBAL
Threads_created	numeric	GLOBAL
Threads_running	numeric	GLOBAL
Uptime	numeric	GLOBAL
Uptime_since_flush_status	numeric	GLOBAL

The status variables have the following meanings.

- Aborted_clients

  The number of connections that were aborted because the client died without closing the connection properly. See Section C.5.2.11, “Communication Errors and Aborted Connections”.

- Aborted_connects

The number of failed attempts to connect to the MySQL server. See Section C.5.2.11, “Communication Errors and Aborted Connections”.

For additional connection-related information, check the `Connection_errors_xxx` status variables and the `host_cache` table.

As of MySQL 5.7.3, `Aborted_connects` is not visible in the embedded server because for that server it is not updated and is not meaningful.

- `Binlog_cache_disk_use`

  The number of transactions that used the temporary binary log cache but that exceeded the value of `binlog_cache_size` and used a temporary file to store statements from the transaction.

  The number of nontransactional statements that caused the binary log transaction cache to be written to disk is tracked separately in the `Binlog_stmt_cache_disk_use` status variable.

- `Binlog_cache_use`

  The number of transactions that used the binary log cache.

- `Binlog_stmt_cache_disk_use`

  The number of nontransaction statements that used the binary log statement cache but that exceeded the value of `binlog_stmt_cache_size` and used a temporary file to store those statements.

- `Binlog_stmt_cache_use`

  The number of nontransactional statements that used the binary log statement cache.

- `Bytes_received`

  The number of bytes received from all clients.

- `Bytes_sent`

  The number of bytes sent to all clients.

- `Com_xxx`

  The `Com_xxx` statement counter variables indicate the number of times each *xxx* statement has been executed. There is one status variable for each type of statement. For example, `Com_delete` and `Com_update` count `DELETE` and `UPDATE` statements, respectively. `Com_delete_multi` and `Com_update_multi` are similar but apply to `DELETE` and `UPDATE` statements that use multiple-table syntax.

  If a query result is returned from query cache, the server increments the `Qcache_hits` status variable, not `Com_select`. See Section 8.9.3.4, “Query Cache Status and Maintenance”.

  All of the `Com_stmt_xxx` variables are increased even if a prepared statement argument is unknown or an error occurred during execution. In other words, their values correspond to the number of requests issued, not to the number of requests successfully completed.

  The `Com_stmt_xxx` status variables are as follows:

  - `Com_stmt_prepare`
  - `Com_stmt_execute`

  - Com_stmt_fetch
  - Com_stmt_send_long_data
  - Com_stmt_reset
  - Com_stmt_close

  Those variables stand for prepared statement commands. Their names refer to the COM_*xxx* command set used in the network layer. In other words, their values increase whenever prepared statement API calls such as mysql_stmt_prepare(), mysql_stmt_execute(), and so forth are executed. However, Com_stmt_prepare, Com_stmt_execute and Com_stmt_close also increase for PREPARE, EXECUTE, or DEALLOCATE PREPARE, respectively. Additionally, the values of the older statement counter variables Com_prepare_sql, Com_execute_sql, and Com_dealloc_sql increase for the PREPARE, EXECUTE, and DEALLOCATE PREPARE statements. Com_stmt_fetch stands for the total number of network round-trips issued when fetching from cursors.

  Com_stmt_reprepare indicates the number of times statements were automatically reprepared by the server after metadata changes to tables or views referred to by the statement. A reprepare operation increments Com_stmt_reprepare, and also Com_stmt_prepare.

  Com_change_repl_filter indicates the number of CHANGE REPLICATION FILTER statements executed. It was introduced in MySQL 5.7.3.

- Compression

  Whether the client connection uses compression in the client/server protocol.

- Connection_errors_*xxx*

  These variables provide information about errors that occur during the client connection process. They are global only and represent error counts aggregated across connections from all hosts. These variables track errors not accounted for by the host cache (see Section 8.11.5.2, “DNS Lookup Optimization and the Host Cache”), such as errors that are not associated with TCP connections, occur very early in the connection process (even before an IP address is known), or are not specific to any particular IP address (such as out-of-memory conditions).

  As of MySQL 5.7.3, the Connection_errors_*xxx* status variables are not visible in the embedded server because for that server they are not updated and are not meaningful.

  - Connection_errors_accept

    The number of errors that occurred during calls to accept() on the listening port.

  - Connection_errors_internal

    The number of connections refused due to internal errors in the server, such as failure to start a new thread or an out-of-memory condition.

  - Connection_errors_max_connections

    The number of connections refused because the server max_connections limit was reached.

  - Connection_errors_peer_addr

    The number of errors that occurred while searching for connecting client IP addresses.

  - `Connection_errors_select`

    The number of errors that occurred during calls to `select()` or `poll()` on the listening port. (Failure of this operation does not necessarily means a client connection was rejected.)

  - `Connection_errors_tcpwrap`

    The number of connections refused by the `libwrap` library.

- `Connections`

  The number of connection attempts (successful or not) to the MySQL server.

- `Created_tmp_disk_tables`

  The number of internal on-disk temporary tables created by the server while executing statements.

  If an internal temporary table is created initially as an in-memory table but becomes too large, MySQL automatically converts it to an on-disk table. The maximum size for in-memory temporary tables is the minimum of the `tmp_table_size` and `max_heap_table_size` values. If `Created_tmp_disk_tables` is large, you may want to increase the `tmp_table_size` or `max_heap_table_size` value to lessen the likelihood that internal temporary tables in memory will be converted to on-disk tables.

  You can compare the number of internal on-disk temporary tables created to the total number of internal temporary tables created by comparing the values of the `Created_tmp_disk_tables` and `Created_tmp_tables` variables.

  See also Section 8.4.4, “How MySQL Uses Internal Temporary Tables”.

- `Created_tmp_files`

  How many temporary files `mysqld` has created.

- `Created_tmp_tables`

  The number of internal temporary tables created by the server while executing statements.

  You can compare the number of internal on-disk temporary tables created to the total number of internal temporary tables created by comparing the values of the `Created_tmp_disk_tables` and `Created_tmp_tables` variables.

  See also Section 8.4.4, “How MySQL Uses Internal Temporary Tables”.

  Each invocation of the `SHOW STATUS` statement uses an internal temporary table and increments the global `Created_tmp_tables` value.

- `Delayed_errors`

  In MySQL 5.7, this status variable is deprecated (because `DELAYED` inserts are not supported), and will be removed in a future release.

- `Delayed_insert_threads`

  In MySQL 5.7, this status variable is deprecated (because `DELAYED` inserts are not supported), and will be removed in a future release.

- `Delayed_writes`

In MySQL 5.7, this status variable is deprecated (because DELAYED inserts are not supported), and will be removed in a future release.

- Flush_commands

  The number of times the server flushes tables, whether because a user executed a FLUSH TABLES statement or due to internal server operation. It is also incremented by receipt of a COM_REFRESH packet. This is in contrast to Com_flush, which indicates how many FLUSH statements have been executed, whether FLUSH TABLES, FLUSH LOGS, and so forth.

- Handler_commit

  The number of internal COMMIT statements.

- Handler_delete

  The number of times that rows have been deleted from tables.

- Handler_external_lock

  The server increments this variable for each call to its external_lock() function, which generally occurs at the beginning and end of access to a table instance. There might be differences among storage engines. This variable can be used, for example, to discover for a statement that accesses a partitioned table how many partitions were pruned before locking occurred: Check how much the counter increased for the statement, subtract 2 (2 calls for the table itself), then divide by 2 to get the number of partitions locked.

- Handler_mrr_init

  The number of times the server uses a storage engine's own Multi-Range Read implementation for table access.

- Handler_prepare

  A counter for the prepare phase of two-phase commit operations.

- Handler_read_first

  The number of times the first entry in an index was read. If this value is high, it suggests that the server is doing a lot of full index scans; for example, SELECT col1 FROM foo, assuming that col1 is indexed.

- Handler_read_key

  The number of requests to read a row based on a key. If this value is high, it is a good indication that your tables are properly indexed for your queries.

- Handler_read_last

  The number of requests to read the last key in an index. With ORDER BY, the server will issue a first-key request followed by several next-key requests, whereas with With ORDER BY DESC, the server will issue a last-key request followed by several previous-key requests.

- Handler_read_next

  The number of requests to read the next row in key order. This value is incremented if you are querying an index column with a range constraint or if you are doing an index scan.

- `Handler_read_prev`

  The number of requests to read the previous row in key order. This read method is mainly used to optimize `ORDER BY ... DESC`.

- `Handler_read_rnd`

  The number of requests to read a row based on a fixed position. This value is high if you are doing a lot of queries that require sorting of the result. You probably have a lot of queries that require MySQL to scan entire tables or you have joins that do not use keys properly.

- `Handler_read_rnd_next`

  The number of requests to read the next row in the data file. This value is high if you are doing a lot of table scans. Generally this suggests that your tables are not properly indexed or that your queries are not written to take advantage of the indexes you have.

- `Handler_rollback`

  The number of requests for a storage engine to perform a rollback operation.

- `Handler_savepoint`

  The number of requests for a storage engine to place a savepoint.

- `Handler_savepoint_rollback`

  The number of requests for a storage engine to roll back to a savepoint.

- `Handler_update`

  The number of requests to update a row in a table.

- `Handler_write`

  The number of requests to insert a row in a table.

- `Innodb_available_undo_logs`

  The total number of available `InnoDB` undo logs. Supplements the `innodb_undo_logs` system variable, which reports the number of active undo logs.

- `Innodb_buffer_pool_dump_status`

  The progress of an operation to record the pages held in the `InnoDB` buffer pool, triggered by the setting of `innodb_buffer_pool_dump_at_shutdown` or `innodb_buffer_pool_dump_now`.

- `Innodb_buffer_pool_load_status`

  The progress of an operation to warm up the `InnoDB` buffer pool by reading in a set of pages corresponding to an earlier point in time, triggered by the setting of `innodb_buffer_pool_load_at_startup` or `innodb_buffer_pool_load_now`. If the operation introduces too much overhead, you can cancel it by setting `innodb_buffer_pool_load_abort`.

- `Innodb_buffer_pool_bytes_data`

  The total number of bytes in the `InnoDB` buffer pool containing data. The number includes both dirty and clean pages. For more accurate memory usage calculations than with

Innodb_buffer_pool_pages_data, when compressed tables cause the buffer pool to hold pages of different sizes.

- Innodb_buffer_pool_pages_data

  The number of pages in the InnoDB buffer pool containing data. The number includes both dirty and clean pages.

- Innodb_buffer_pool_bytes_dirty

  The total current number of bytes held in dirty pages in the InnoDB buffer pool. For more accurate memory usage calculations than with Innodb_buffer_pool_pages_dirty, when compressed tables cause the buffer pool to hold pages of different sizes.

- Innodb_buffer_pool_pages_dirty

  The current number of dirty pages in the InnoDB buffer pool.

- Innodb_buffer_pool_pages_flushed

  The number of requests to flush pages from the InnoDB buffer pool.

- Innodb_buffer_pool_pages_free

  The number of free pages in the InnoDB buffer pool.

- Innodb_buffer_pool_pages_latched

  The number of latched pages in the InnoDB buffer pool. These are pages currently being read or written, or that cannot be flushed or removed for some other reason. Calculation of this variable is expensive, so it is available only when the UNIV_DEBUG system is defined at server build time.

- Innodb_buffer_pool_pages_misc

  The number of pages in the InnoDB buffer pool that are busy because they have been allocated for administrative overhead, such as row locks or the adaptive hash index. This value can also be calculated as Innodb_buffer_pool_pages_total – Innodb_buffer_pool_pages_free – Innodb_buffer_pool_pages_data.

- Innodb_buffer_pool_pages_total

  The total size of the InnoDB buffer pool, in pages.

- Innodb_buffer_pool_read_ahead

  The number of pages read into the InnoDB buffer pool by the read-ahead background thread.

- Innodb_buffer_pool_read_ahead_evicted

  The number of pages read into the InnoDB buffer pool by the read-ahead background thread that were subsequently evicted without having been accessed by queries.

- Innodb_buffer_pool_read_requests

  The number of logical read requests.

- Innodb_buffer_pool_reads

  The number of logical reads that InnoDB could not satisfy from the buffer pool, and had to read directly from disk.

- Innodb_buffer_pool_wait_free

  Normally, writes to the InnoDB buffer pool happen in the background. When InnoDB needs to read or create a page and no clean pages are available, InnoDB flushes some dirty pages first and waits for that operation to finish. This counter counts instances of these waits. If innodb_buffer_pool_size has been set properly, this value should be small.

- Innodb_buffer_pool_write_requests

  The number of writes done to the InnoDB buffer pool.

- Innodb_data_fsyncs

  The number of fsync() operations so far. The frequency of fsync() calls is influenced by the setting of the innodb_flush_method configuration option.

- Innodb_data_pending_fsyncs

  The current number of pending fsync() operations. The frequency of fsync() calls is influenced by the setting of the innodb_flush_method configuration option.

- Innodb_data_pending_reads

  The current number of pending reads.

- Innodb_data_pending_writes

  The current number of pending writes.

- Innodb_data_read

  The amount of data read since the server was started.

- Innodb_data_reads

  The total number of data reads.

- Innodb_data_writes

  The total number of data writes.

- Innodb_data_written

  The amount of data written so far, in bytes.

- Innodb_dblwr_pages_written

  The number of pages that have been written to the doublewrite buffer. See Section 14.2.10.1, "InnoDB Disk I/O".

- Innodb_dblwr_writes

  The number of doublewrite operations that have been performed. See Section 14.2.10.1, "InnoDB Disk I/O".

- Innodb_have_atomic_builtins

  Indicates whether the server was built with atomic instructions.

- Innodb_log_waits

The number of times that the log buffer was too small and a wait was required for it to be flushed before continuing.

- `Innodb_log_write_requests`

  The number of write requests for the `InnoDB` redo log.

- `Innodb_log_writes`

  The number of physical writes to the `InnoDB` redo log file.

- `Innodb_num_open_files`

  The number of files `InnoDB` currently holds open.

- `Innodb_os_log_fsyncs`

  The number of `fsync()` writes done to the `InnoDB` redo log files.

- `Innodb_os_log_pending_fsyncs`

  The number of pending `fsync()` operations for the `InnoDB` redo log files.

- `Innodb_os_log_pending_writes`

  The number of pending writes to the `InnoDB` redo log files.

- `Innodb_os_log_written`

  The number of bytes written to the `InnoDB` redo log files.

- `Innodb_page_size`

  The compiled-in `InnoDB` page size (default 16KB). Many values are counted in pages; the page size enables them to be easily converted to bytes.

- `Innodb_pages_created`

  The number of pages created by operations on `InnoDB` tables.

- `Innodb_pages_read`

  The number of pages read by operations on `InnoDB` tables.

- `Innodb_pages_written`

  The number of pages written by operations on `InnoDB` tables.

- `Innodb_row_lock_current_waits`

  The number of row locks currently being waited for by operations on `InnoDB` tables.

- `Innodb_row_lock_time`

  The total time spent in acquiring row locks for `InnoDB` tables, in milliseconds.

- `Innodb_row_lock_time_avg`

  The average time to acquire a row lock for `InnoDB` tables, in milliseconds.

- `Innodb_row_lock_time_max`

  The maximum time to acquire a row lock for `InnoDB` tables, in milliseconds.

- `Innodb_row_lock_waits`

  The number of times operations on `InnoDB` tables had to wait for a row lock.

- `Innodb_rows_deleted`

  The number of rows deleted from `InnoDB` tables.

- `Innodb_rows_inserted`

  The number of rows inserted into `InnoDB` tables.

- `Innodb_rows_read`

  The number of rows read from `InnoDB` tables.

- `Innodb_rows_updated`

  The number of rows updated in `InnoDB` tables.

- `Innodb_truncated_status_writes`

  The number of times output from the `SHOW ENGINE INNODB STATUS` statement has been truncated.

- `Key_blocks_not_flushed`

  The number of key blocks in the `MyISAM` key cache that have changed but have not yet been flushed to disk.

- `Key_blocks_unused`

  The number of unused blocks in the `MyISAM` key cache. You can use this value to determine how much of the key cache is in use; see the discussion of `key_buffer_size` in Section 5.1.4, “Server System Variables”.

- `Key_blocks_used`

  The number of used blocks in the `MyISAM` key cache. This value is a high-water mark that indicates the maximum number of blocks that have ever been in use at one time.

- `Key_read_requests`

  The number of requests to read a key block from the `MyISAM` key cache.

- `Key_reads`

  The number of physical reads of a key block from disk into the `MyISAM` key cache. If `Key_reads` is large, then your `key_buffer_size` value is probably too small. The cache miss rate can be calculated as `Key_reads`/`Key_read_requests`.

- `Key_write_requests`

  The number of requests to write a key block to the `MyISAM` key cache.

- `Key_writes`

The number of physical writes of a key block from the MyISAM key cache to disk.

- Last_query_cost

  The total cost of the last compiled query as computed by the query optimizer. This is useful for comparing the cost of different query plans for the same query. The default value of 0 means that no query has been compiled yet. The default value is 0. Last_query_cost has session scope.

  The Last_query_cost value can be computed accurately only for simple “flat” queries, not complex queries such as those with subqueries or UNION. For the latter, the value is set to 0.

- Last_query_partial_plans

  The number of iterations the query optimizer made in execution plan construction for the previous query. Last_query_cost has session scope.

- Max_statement_time_exceeded

  The number of SELECT statements for which the execution timeout was exceeded. This variable was added in MySQL 5.7.4.

- Max_statement_time_set

  The number of SELECT statements for which a nonzero execution timeout was set. This includes statements that include a nonzero MAX_STATEMENT_TIME option, and statements that include no such option but execute while the timeout indicated by the max_statement_time system variable is nonzero. This variable was added in MySQL 5.7.4.

- Max_statement_time_set_failed

  The number of SELECT statements for which the attempt to set an execution timeout failed. This variable was added in MySQL 5.7.4.

- Max_used_connections

  The maximum number of connections that have been in use simultaneously since the server started.

- Max_used_connections_time

  The time at which Max_used_connections reached its current value. This variable was added in MySQL 5.7.5.

- Not_flushed_delayed_rows

  In MySQL 5.7, this status variable is deprecated (because DELAYED inserts are not supported), and will be removed in a future release.

- Open_files

  The number of files that are open. This count includes regular files opened by the server. It does not include other types of files such as sockets or pipes. Also, the count does not include files that storage engines open using their own internal functions rather than asking the server level to do so.

- Open_streams

  The number of streams that are open (used mainly for logging).

- Open_table_definitions

The number of cached .frm files.

- Open_tables

  The number of tables that are open.

- Opened_files

  The number of files that have been opened with my_open() (a mysys library function). Parts of the server that open files without using this function do not increment the count.

- Opened_table_definitions

  The number of .frm files that have been cached.

- Opened_tables

  The number of tables that have been opened. If Opened_tables is big, your table_open_cache value is probably too small.

- Performance_schema_*xxx*

  Performance Schema status variables are listed in Section 20.13, “Performance Schema Status Variables”. These variables provide information about instrumentation that could not be loaded or created due to memory constraints.

- Prepared_stmt_count

  The current number of prepared statements. (The maximum number of statements is given by the max_prepared_stmt_count system variable.)

- Qcache_free_blocks

  The number of free memory blocks in the query cache.

- Qcache_free_memory

  The amount of free memory for the query cache.

- Qcache_hits

  The number of query cache hits.

- Qcache_inserts

  The number of queries added to the query cache.

- Qcache_lowmem_prunes

  The number of queries that were deleted from the query cache because of low memory.

- Qcache_not_cached

  The number of noncached queries (not cacheable, or not cached due to the query_cache_type setting).

- Qcache_queries_in_cache

  The number of queries registered in the query cache.

- `Qcache_total_blocks`

  The total number of blocks in the query cache.

- `Queries`

  The number of statements executed by the server. This variable includes statements executed within stored programs, unlike the `Questions` variable. It does not count `COM_PING` or `COM_STATISTICS` commands.

- `Questions`

  The number of statements executed by the server. This includes only statements sent to the server by clients and not statements executed within stored programs, unlike the `Queries` variable. This variable does not count `COM_PING`, `COM_STATISTICS`, `COM_STMT_PREPARE`, `COM_STMT_CLOSE`, or `COM_STMT_RESET` commands.

- `Rpl_semi_sync_master_clients`

  The number of semisynchronous slaves.

  This variable is available only if the master-side semisynchronous replication plugin is installed.

- `Rpl_semi_sync_master_net_avg_wait_time`

  The average time in microseconds the master waited for a slave reply.

  This variable is available only if the master-side semisynchronous replication plugin is installed.

- `Rpl_semi_sync_master_net_wait_time`

  The total time in microseconds the master waited for slave replies.

  This variable is available only if the master-side semisynchronous replication plugin is installed.

- `Rpl_semi_sync_master_net_waits`

  The total number of times the master waited for slave replies.

  This variable is available only if the master-side semisynchronous replication plugin is installed.

- `Rpl_semi_sync_master_no_times`

  The number of times the master turned off semisynchronous replication.

  This variable is available only if the master-side semisynchronous replication plugin is installed.

- `Rpl_semi_sync_master_no_tx`

  The number of commits that were not acknowledged successfully by a slave.

  This variable is available only if the master-side semisynchronous replication plugin is installed.

- `Rpl_semi_sync_master_status`

  Whether semisynchronous replication currently is operational on the master. The value is `ON` if the plugin has been enabled and a commit acknowledgment has occurred. It is `OFF` if the plugin is not enabled or the master has fallen back to asynchronous replication due to commit acknowledgment timeout.

This variable is available only if the master-side semisynchronous replication plugin is installed.

- Rpl_semi_sync_master_timefunc_failures

  The number of times the master failed when calling time functions such as gettimeofday().

  This variable is available only if the master-side semisynchronous replication plugin is installed.

- Rpl_semi_sync_master_tx_avg_wait_time

  The average time in microseconds the master waited for each transaction.

  This variable is available only if the master-side semisynchronous replication plugin is installed.

- Rpl_semi_sync_master_tx_wait_time

  The total time in microseconds the master waited for transactions.

  This variable is available only if the master-side semisynchronous replication plugin is installed.

- Rpl_semi_sync_master_tx_waits

  The total number of times the master waited for transactions.

  This variable is available only if the master-side semisynchronous replication plugin is installed.

- Rpl_semi_sync_master_wait_pos_backtraverse

  The total number of times the master waited for an event with binary coordinates lower than events waited for previously. This can occur when the order in which transactions start waiting for a reply is different from the order in which their binary log events are written.

  This variable is available only if the master-side semisynchronous replication plugin is installed.

- Rpl_semi_sync_master_wait_sessions

  The number of sessions currently waiting for slave replies.

  This variable is available only if the master-side semisynchronous replication plugin is installed.

- Rpl_semi_sync_master_yes_tx

  The number of commits that were acknowledged successfully by a slave.

  This variable is available only if the master-side semisynchronous replication plugin is installed.

- Rpl_semi_sync_slave_status

  Whether semisynchronous replication currently is operational on the slave. This is ON if the plugin has been enabled and the slave I/O thread is running, OFF otherwise.

  This variable is available only if the slave-side semisynchronous replication plugin is installed.

- Rsa_public_key

  The RSA public key value used by the sha256_password authentication plugin. The value is nonempty only if the server successfully initializes the private and public keys in the files named by the sha256_password_private_key_path and sha256_password_public_key_path system variables. The value of Rsa_public_key comes from the latter file.

For information about sha256_password, see Section 6.3.9.4, “The SHA-256 Authentication Plugin”.

This variable is available only if MySQL was built using OpenSSL.

- Select_full_join

  The number of joins that perform table scans because they do not use indexes. If this value is not 0, you should carefully check the indexes of your tables.

- Select_full_range_join

  The number of joins that used a range search on a reference table.

- Select_range

  The number of joins that used ranges on the first table. This is normally not a critical issue even if the value is quite large.

- Select_range_check

  The number of joins without keys that check for key usage after each row. If this is not 0, you should carefully check the indexes of your tables.

- Select_scan

  The number of joins that did a full scan of the first table.

- Slave_heartbeat_period

  Shows the replication heartbeat interval (in seconds) on a replication slave.

- Slave_last_heartbeat

  Shows when the most recent heartbeat signal was received by a replication slave, as a TIMESTAMP value.

- Slave_open_temp_tables

  The number of temporary tables that the slave SQL thread currently has open. If the value is greater than zero, it is not safe to shut down the slave; see Section 16.4.1.22, “Replication and Temporary Tables”.

- Slave_received_heartbeats

  This counter increments with each replication heartbeat received by a replication slave since the last time that the slave was restarted or reset, or a CHANGE MASTER TO statement was issued.

- Slave_retried_transactions

  The total number of times since startup that the replication slave SQL thread has retried transactions.

- Slave_running

  This is ON if this server is a replication slave that is connected to a replication master, and both the I/O and SQL threads are running; otherwise, it is OFF.

- Slow_launch_threads

  The number of threads that have taken more than slow_launch_time seconds to create.

This variable is not meaningful in the embedded server (libmysqld) and as of MySQL 5.7.2 is no longer visible within the embedded server.

- Slow_queries

  The number of queries that have taken more than long_query_time seconds. This counter increments regardless of whether the slow query log is enabled. For information about that log, see Section 5.2.5, “The Slow Query Log”.

- Sort_merge_passes

  The number of merge passes that the sort algorithm has had to do. If this value is large, you should consider increasing the value of the sort_buffer_size system variable.

- Sort_range

  The number of sorts that were done using ranges.

- Sort_rows

  The number of sorted rows.

- Sort_scan

  The number of sorts that were done by scanning the table.

- Ssl_accept_renegotiates

  The number of negotiates needed to establish the connection.

- Ssl_accepts

  The number of accepted SSL connections.

- Ssl_callback_cache_hits

  The number of callback cache hits.

- Ssl_cipher

  The current SSL cipher (empty for non-SSL connections).

- Ssl_cipher_list

  The list of possible SSL ciphers.

- Ssl_client_connects

  The number of SSL connection attempts to an SSL-enabled master.

- Ssl_connect_renegotiates

  The number of negotiates needed to establish the connection to an SSL-enabled master.

- Ssl_ctx_verify_depth

  The SSL context verification depth (how many certificates in the chain are tested).

- Ssl_ctx_verify_mode

The SSL context verification mode.

- `Ssl_default_timeout`

  The default SSL timeout.

- `Ssl_finished_accepts`

  The number of successful SSL connections to the server.

- `Ssl_finished_connects`

  The number of successful slave connections to an SSL-enabled master.

- `Ssl_server_not_after`

  The last date for which the SSL certificate is valid.

- `Ssl_server_not_before`

  The first date for which the SSL certificate is valid.

- `Ssl_session_cache_hits`

  The number of SSL session cache hits.

- `Ssl_session_cache_misses`

  The number of SSL session cache misses.

- `Ssl_session_cache_mode`

  The SSL session cache mode.

- `Ssl_session_cache_overflows`

  The number of SSL session cache overflows.

- `Ssl_session_cache_size`

  The SSL session cache size.

- `Ssl_session_cache_timeouts`

  The number of SSL session cache timeouts.

- `Ssl_sessions_reused`

  How many SSL connections were reused from the cache.

- `Ssl_used_session_cache_entries`

  How many SSL session cache entries were used.

- `Ssl_verify_depth`

  The verification depth for replication SSL connections.

- `Ssl_verify_mode`

The verification mode for replication SSL connections.

- `Ssl_version`

  The SSL protocol version of the connection.

- `Table_locks_immediate`

  The number of times that a request for a table lock could be granted immediately.

- `Table_locks_waited`

  The number of times that a request for a table lock could not be granted immediately and a wait was needed. If this is high and you have performance problems, you should first optimize your queries, and then either split your table or tables or use replication.

- `Table_open_cache_hits`

  The number of hits for open tables cache lookups.

- `Table_open_cache_misses`

  The number of misses for open tables cache lookups.

- `Table_open_cache_overflows`

  The number of overflows for the open tables cache. This is the number of times, after a table is opened or closed, a cache instance has an unused entry and the size of the instance is larger than `table_open_cache` / `table_open_cache_instances`.

- `Tc_log_max_pages_used`

  For the memory-mapped implementation of the log that is used by `mysqld` when it acts as the transaction coordinator for recovery of internal XA transactions, this variable indicates the largest number of pages used for the log since the server started. If the product of `Tc_log_max_pages_used` and `Tc_log_page_size` is always significantly less than the log size, the size is larger than necessary and can be reduced. (The size is set by the `--log-tc-size` option. Currently, this variable is unused: It is unneeded for binary log-based recovery, and the memory-mapped recovery log method is not used unless the number of storage engines capable of two-phase commit is greater than one. (`InnoDB` is the only applicable engine.)

- `Tc_log_page_size`

  The page size used for the memory-mapped implementation of the XA recovery log. The default value is determined using `getpagesize()`. Currently, this variable is unused for the same reasons as described for `Tc_log_max_pages_used`.

- `Tc_log_page_waits`

  For the memory-mapped implementation of the recovery log, this variable increments each time the server was not able to commit a transaction and had to wait for a free page in the log. If this value is large, you might want to increase the log size (with the `--log-tc-size` option). For binary log-based recovery, this variable increments each time the binary log cannot be closed because there are two-phase commits in progress. (The close operation waits until all such transactions are finished.)

- `Threads_cached`

  The number of threads in the thread cache.

This variable is not meaningful in the embedded server (libmysqld) and as of MySQL 5.7.2 is no longer visible within the embedded server.

- Threads_connected

  The number of currently open connections.

- Threads_created

  The number of threads created to handle connections. If Threads_created is big, you may want to increase the thread_cache_size value. The cache miss rate can be calculated as Threads_created/Connections.

- Threads_running

  The number of threads that are not sleeping.

- Uptime

  The number of seconds that the server has been up.

- Uptime_since_flush_status

  The number of seconds since the most recent FLUSH STATUS statement.

5.1.7 Server SQL Modes

The MySQL server can operate in different SQL modes, and can apply these modes differently for different clients, depending on the value of the sql_mode system variable. DBAs can set the global SQL mode to match site server operating requirements, and each application can set its session SQL mode to its own requirements.

Modes affect the SQL syntax MySQL supports and the data validation checks it performs. This makes it easier to use MySQL in different environments and to use MySQL together with other database servers.

- Setting the SQL Mode
- The Most Important SQL Modes
- Full List of SQL Modes
- Strict SQL Mode
- Combination SQL Modes
- SQL Mode Changes in MySQL 5.7

For answers to questions often asked about server SQL modes in MySQL, see Section B.3, “MySQL 5.7 FAQ: Server SQL Mode”.

When working with InnoDB tables, consider also the innodb_strict_mode system variable. It enables additional error checks for InnoDB tables.

Setting the SQL Mode

The default SQL mode in MySQL 5.7 is NO_ENGINE_SUBSTITUTION.

To set the SQL mode at server startup, use the `--sql-mode="modes"` option on the command line, or `sql-mode="modes"` in an option file such as `my.cnf` (Unix operating systems) or `my.ini` (Windows). *`modes`* is a list of different modes separated by commas. To clear the SQL mode explicitly, set it to an empty string using `--sql-mode=""` on the command line, or `sql-mode=""` in an option file.

> **Note**
>
> MySQL installation programs may configure the SQL mode during the installation process. For example, `mysql_install_db` creates a default option file named `my.cnf` in the base installation directory. This file contains a line that sets the SQL mode; see Section 4.4.3, "`mysql_install_db` — Initialize MySQL Data Directory".
>
> If the SQL mode differs from the default or from what you expect, check for a setting in an option file that the server reads at startup.

To change the SQL mode at runtime, set the global or session `sql_mode` system variable using a `SET` statement:

```
SET GLOBAL sql_mode = 'modes';
SET SESSION sql_mode = 'modes';
```

Setting the `GLOBAL` variable requires the `SUPER` privilege and affects the operation of all clients that connect from that time on. Setting the `SESSION` variable affects only the current client. Each client can change its session `sql_mode` value at any time.

To determine the current global or session `sql_mode` value, use the following statements:

```
SELECT @@GLOBAL.sql_mode;
SELECT @@SESSION.sql_mode;
```

> **Important**
>
> **SQL mode and user-defined partitioning.** Changing the server SQL mode after creating and inserting data into partitioned tables can cause major changes in the behavior of such tables, and could lead to loss or corruption of data. It is strongly recommended that you never change the SQL mode once you have created tables employing user-defined partitioning.
>
> When replicating partitioned tables, differing SQL modes on master and slave can also lead to problems. For best results, you should always use the same server SQL mode on the master and on the slave.
>
> See Section 17.6, "Restrictions and Limitations on Partitioning", for more information.

The Most Important SQL Modes

The most important `sql_mode` values are probably these:

- `ANSI`

  This mode changes syntax and behavior to conform more closely to standard SQL. It is one of the special combination modes listed at the end of this section.

- `STRICT_TRANS_TABLES`

If a value could not be inserted as given into a transactional table, abort the statement. For a nontransactional table, abort the statement if the value occurs in a single-row statement or the first row of a multiple-row statement. More details are given later in this section.

- `TRADITIONAL`

  Make MySQL behave like a "traditional" SQL database system. A simple description of this mode is "give an error instead of a warning" when inserting an incorrect value into a column. It is one of the special combination modes listed at the end of this section.

  Note

  The `INSERT` or `UPDATE` aborts as soon as the error is noticed. This may not be what you want if you are using a nontransactional storage engine, because data changes made prior to the error may not be rolled back, resulting in a "partially done" update.

When this manual refers to "strict mode," it means a mode with either or both `STRICT_TRANS_TABLES` or `STRICT_ALL_TABLES` enabled.

Full List of SQL Modes

The following list describes all supported SQL modes:

- `ALLOW_INVALID_DATES`

  Do not perform full checking of dates. Check only that the month is in the range from 1 to 12 and the day is in the range from 1 to 31. This is very convenient for Web applications where you obtain year, month, and day in three different fields and you want to store exactly what the user inserted (without date validation). This mode applies to `DATE` and `DATETIME` columns. It does not apply `TIMESTAMP` columns, which always require a valid date.

  The server requires that month and day values be legal, and not merely in the range 1 to 12 and 1 to 31, respectively. With strict mode disabled, invalid dates such as `'2004-04-31'` are converted to `'0000-00-00'` and a warning is generated. With strict mode enabled, invalid dates generate an error. To permit such dates, enable `ALLOW_INVALID_DATES`.

- `ANSI_QUOTES`

  Treat "`"`" as an identifier quote character (like the "`` ` ``" quote character) and not as a string quote character. You can still use "`` ` ``" to quote identifiers with this mode enabled. With `ANSI_QUOTES` enabled, you cannot use double quotation marks to quote literal strings, because it is interpreted as an identifier.

- `ERROR_FOR_DIVISION_BY_ZERO`

  As of MySQL 5.7.4, the `ERROR_FOR_DIVISION_BY_ZERO` mode does nothing. Instead, its pre-5.7.4 effect is included in the effects of strict SQL mode.

  Before MySQL 5.7.4, the `ERROR_FOR_DIVISION_BY_ZERO` mode affects handling of division by zero, which includes `MOD(N,0)`. For data-change operations (`INSERT`, `UPDATE`), its effect also depends on whether strict mode is enabled.

  - If this mode is not enabled, division by zero inserts `NULL` and produces no warning.

  - If this mode is enabled, division by zero inserts `NULL` and produces a warning.

  - If this mode and strict mode are enabled, division by zero produces an error, unless IGNORE is given as well. For INSERT IGNORE and UPDATE IGNORE, division by zero inserts NULL and produces a warning.

  For SELECT, division by zero returns NULL. Enabling ERROR_FOR_DIVISION_BY_ZERO causes a warning to be produced as well, regardless of whether strict mode is enabled.

- HIGH_NOT_PRECEDENCE

  The precedence of the NOT operator is such that expressions such as NOT a BETWEEN b AND c are parsed as NOT (a BETWEEN b AND c). In some older versions of MySQL, the expression was parsed as (NOT a) BETWEEN b AND c. The old higher-precedence behavior can be obtained by enabling the HIGH_NOT_PRECEDENCE SQL mode.

```
mysql> SET sql_mode = '';
mysql> SELECT NOT 1 BETWEEN -5 AND 5;
        -> 0
mysql> SET sql_mode = 'HIGH_NOT_PRECEDENCE';
mysql> SELECT NOT 1 BETWEEN -5 AND 5;
        -> 1
```

- IGNORE_SPACE

  Permit spaces between a function name and the “(” character. This causes built-in function names to be treated as reserved words. As a result, identifiers that are the same as function names must be quoted as described in Section 9.2, “Schema Object Names”. For example, because there is a COUNT() function, the use of count as a table name in the following statement causes an error:

```
mysql> CREATE TABLE count (i INT);
ERROR 1064 (42000): You have an error in your SQL syntax
```

  The table name should be quoted:

```
mysql> CREATE TABLE `count` (i INT);
Query OK, 0 rows affected (0.00 sec)
```

  The IGNORE_SPACE SQL mode applies to built-in functions, not to user-defined functions or stored functions. It is always permissible to have spaces after a UDF or stored function name, regardless of whether IGNORE_SPACE is enabled.

  For further discussion of IGNORE_SPACE, see Section 9.2.4, “Function Name Parsing and Resolution”.

- NO_AUTO_CREATE_USER

  Prevent the GRANT statement from automatically creating new users if it would otherwise do so, unless authentication information is specified. The statement must specify a nonempty password using IDENTIFIED BY or an authentication plugin using IDENTIFIED WITH.

- NO_AUTO_VALUE_ON_ZERO

  NO_AUTO_VALUE_ON_ZERO affects handling of AUTO_INCREMENT columns. Normally, you generate the next sequence number for the column by inserting either NULL or 0 into it. NO_AUTO_VALUE_ON_ZERO suppresses this behavior for 0 so that only NULL generates the next sequence number.

  This mode can be useful if 0 has been stored in a table's AUTO_INCREMENT column. (Storing 0 is not a recommended practice, by the way.) For example, if you dump the table with mysqldump and then reload it, MySQL normally generates new sequence numbers when it encounters the 0 values, resulting in a table with contents different from the one that was dumped. Enabling NO_AUTO_VALUE_ON_ZERO before reloading the dump file solves this problem. mysqldump now automatically includes in its output a statement that enables NO_AUTO_VALUE_ON_ZERO, to avoid this problem.

- NO_BACKSLASH_ESCAPES

  Disable the use of the backslash character (“\”) as an escape character within strings. With this mode enabled, backslash becomes an ordinary character like any other.

- NO_DIR_IN_CREATE

  When creating a table, ignore all INDEX DIRECTORY and DATA DIRECTORY directives. This option is useful on slave replication servers.

- NO_ENGINE_SUBSTITUTION

  Control automatic substitution of the default storage engine when a statement such as CREATE TABLE or ALTER TABLE specifies a storage engine that is disabled or not compiled in.

  Because storage engines can be pluggable at runtime, unavailable engines are treated the same way:

  With NO_ENGINE_SUBSTITUTION disabled, for CREATE TABLE the default engine is used and a warning occurs if the desired engine is unavailable. For ALTER TABLE, a warning occurs and the table is not altered.

  With NO_ENGINE_SUBSTITUTION enabled, an error occurs and the table is not created or altered if the desired engine is unavailable.

- NO_FIELD_OPTIONS

  Do not print MySQL-specific column options in the output of SHOW CREATE TABLE. This mode is used by mysqldump in portability mode.

- NO_KEY_OPTIONS

  Do not print MySQL-specific index options in the output of SHOW CREATE TABLE. This mode is used by mysqldump in portability mode.

- NO_TABLE_OPTIONS

  Do not print MySQL-specific table options (such as ENGINE) in the output of SHOW CREATE TABLE. This mode is used by mysqldump in portability mode.

- NO_UNSIGNED_SUBTRACTION

  By default, subtraction between integer operands produces an UNSIGNED result if any operand isUNSIGNED. When NO_UNSIGNED_SUBTRACTION is enabled, the subtraction result is signed, *even if any operand is unsigned*. For example, compare the type of column c2 in table t1 with that of column c2 in table t2:

```
mysql> SET sql_mode='';
mysql> CREATE TABLE test (c1 BIGINT UNSIGNED NOT NULL);
mysql> CREATE TABLE t1 SELECT c1 - 1 AS c2 FROM test;
mysql> DESCRIBE t1;
+-------+---------------------+------+-----+---------+-------+
| Field | Type                | Null | Key | Default | Extra |
+-------+---------------------+------+-----+---------+-------+
| c2    | bigint(21) unsigned |      |     | 0       |       |
+-------+---------------------+------+-----+---------+-------+

mysql> SET sql_mode='NO_UNSIGNED_SUBTRACTION';
mysql> CREATE TABLE t2 SELECT c1 - 1 AS c2 FROM test;
mysql> DESCRIBE t2;
+-------+------------+------+-----+---------+-------+
| Field | Type       | Null | Key | Default | Extra |
+-------+------------+------+-----+---------+-------+
| c2    | bigint(21) |      |     | 0       |       |
+-------+------------+------+-----+---------+-------+
```

Note that this means that BIGINT UNSIGNED is not 100% usable in all contexts. See Section 12.10, “Cast Functions and Operators”.

```
mysql> SET sql_mode = '';
mysql> SELECT CAST(0 AS UNSIGNED) - 1;
+-------------------------+
| CAST(0 AS UNSIGNED) - 1 |
+-------------------------+
|    18446744073709551615 |
+-------------------------+

mysql> SET sql_mode = 'NO_UNSIGNED_SUBTRACTION';
mysql> SELECT CAST(0 AS UNSIGNED) - 1;
+-------------------------+
| CAST(0 AS UNSIGNED) - 1 |
+-------------------------+
|                      -1 |
+-------------------------+
```

- NO_ZERO_DATE

  As of MySQL 5.7.4, the NO_ZERO_DATE mode does nothing. Instead, its pre-5.7.4 effect is included in the effects of strict SQL mode.

  Before MySQL 5.7.4, the NO_ZERO_DATE mode affects whether the server permits '0000-00-00' as a valid date. Its effect also depends on whether strict mode is enabled.

  - If this mode is not enabled, '0000-00-00' is permitted and inserts produce no warning.
  - If this mode is enabled, '0000-00-00' is permitted and inserts produce a warning.
  - If this mode and strict mode are enabled, '0000-00-00' is not permitted and inserts produce an error, unless IGNORE is given as well. For INSERT IGNORE and UPDATE IGNORE, '0000-00-00' is permitted and inserts produce a warning.

- NO_ZERO_IN_DATE

  As of MySQL 5.7.4, the NO_ZERO_IN_DATE mode does nothing. Instead, its pre-5.7.4 effect is included in the effects of strict SQL mode.

  Before MySQL 5.7.4, the NO_ZERO_IN_DATE mode affects whether the server permits dates in which the year part is nonzero but the month or day part is 0. (This mode affects dates such as '2010-00-01' or '2010-01-00', but not '0000-00-00'. To control whether the server permits '0000-00-00', use the NO_ZERO_DATE mode.) The effect of NO_ZERO_IN_DATE also depends on whether strict mode is enabled.

  - If this mode is not enabled, dates with zero parts are permitted and inserts produce no warning.
  - If this mode is enabled, dates with zero parts are inserted as '0000-00-00' and produce a warning.

- If this mode and strict mode are enabled, dates with zero parts are not permitted and inserts produce an error, unless IGNORE is given as well. For INSERT IGNORE and UPDATE IGNORE, dates with zero parts are inserted as '0000-00-00' and produce a warning.

- ONLY_FULL_GROUP_BY

  Do not permit queries for which the select list or HAVING list or ORDER BY list refers to nonaggregated columns that are not named in the GROUP BY clause.

  The following queries are invalid with ONLY_FULL_GROUP_BY enabled. The first is invalid because address in the select list is not named in the GROUP BY clause, and the second because max_age in the HAVING clause is not named in the GROUP BY clause:

```
mysql> SELECT name, address, MAX(age) FROM t GROUP BY name;
ERROR 1055 (42000): 't.address' isn't in GROUP BY
```

```
mysql> SELECT name, MAX(age) AS max_age FROM t GROUP BY name
    -> HAVING max_age < 30;
Empty set (0.00 sec)
ERROR 1463 (42000): Non-grouping field 'max_age' is used in HAVING clause
```

  In the second example, the query could be rewritten to use HAVING MAX(age) instead, so that the reference is to a column named in an aggregate function. (max_age fails because it *is* an aggregate function.)

  In addition, if a query has aggregate functions and no GROUP BY clause, it cannot have nonaggregated columns in the select list or ORDER BY list:

```
mysql> SELECT name, MAX(age) FROM t;
ERROR 1140 (42000): Mixing of GROUP columns (MIN(),MAX(),COUNT(),...)
with no GROUP columns is illegal if there is no GROUP BY clause
```

  For more information, see Section 12.17.3, “MySQL Extensions to GROUP BY”.

- PAD_CHAR_TO_FULL_LENGTH

  By default, trailing spaces are trimmed from CHAR column values on retrieval. If PAD_CHAR_TO_FULL_LENGTH is enabled, trimming does not occur and retrieved CHAR values are padded to their full length. This mode does not apply to VARCHAR columns, for which trailing spaces are retained on retrieval.

```
mysql> CREATE TABLE t1 (c1 CHAR(10));
Query OK, 0 rows affected (0.37 sec)

mysql> INSERT INTO t1 (c1) VALUES('xy');
Query OK, 1 row affected (0.01 sec)

mysql> SET sql_mode = '';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT c1, CHAR_LENGTH(c1) FROM t1;
+------+-----------------+
| c1   | CHAR_LENGTH(c1) |
+------+-----------------+
| xy   |               2 |
+------+-----------------+
1 row in set (0.00 sec)

mysql> SET sql_mode = 'PAD_CHAR_TO_FULL_LENGTH';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT c1, CHAR_LENGTH(c1) FROM t1;
+------------+-----------------+
| c1         | CHAR_LENGTH(c1) |
+------------+-----------------+
| xy         |              10 |
+------------+-----------------+
1 row in set (0.00 sec)
```

- `PIPES_AS_CONCAT`

  Treat `||` as a string concatenation operator (same as `CONCAT()`) rather than as a synonym for `OR`.

- `REAL_AS_FLOAT`

  Treat `REAL` as a synonym for `FLOAT`. By default, MySQL treats `REAL` as a synonym for `DOUBLE`.

- `STRICT_ALL_TABLES`

  Enable strict mode for all storage engines. Invalid data values are rejected. Additional details follow.

- `STRICT_TRANS_TABLES`

  Enable strict mode for transactional storage engines, and when possible for nontransactional storage engines. Additional details follow.

Strict SQL Mode

Strict mode controls how MySQL handles invalid or missing values in data-change statements such as `INSERT` or `UPDATE`. A value can be invalid for several reasons. For example, it might have the wrong data type for the column, or it might be out of range. A value is missing when a new row to be inserted does not contain a value for a non-`NULL` column that has no explicit `DEFAULT` clause in its definition. (For a `NULL` column, `NULL` is inserted if the value is missing.)

If strict mode is in effect, MySQL inserts adjusted values for invalid or missing values and produces warnings (see Section 13.7.5.39, “`SHOW WARNINGS` Syntax”). In strict mode, you can produce this behavior by using `INSERT IGNORE` or `UPDATE IGNORE`.

For statements such as `SELECT` that do not change data, invalid values generate a warning in strict mode, not an error.

Strict mode does not affect whether foreign key constraints are checked. `foreign_key_checks` can be used for that. (See Section 5.1.4, “Server System Variables”.)

Strict SQL mode is in effect if either `STRICT_ALL_TABLES` or `STRICT_TRANS_TABLES` is enabled, although the effects of these modes differ somewhat:

- For transactional tables, an error occurs for invalid or missing values in a data-change statement when either `STRICT_ALL_TABLES` or `STRICT_TRANS_TABLES` is enabled. The statement is aborted and rolled back.

- For nontransactional tables, the behavior is the same for either mode if the bad value occurs in the first row to be inserted or updated: The statement is aborted and the table remains unchanged. If the statement inserts or modifies multiple rows and the bad value occurs in the second or later row, the result depends on which strict mode is enabled:

- For STRICT_ALL_TABLES, MySQL returns an error and ignores the rest of the rows. However, because the earlier rows have been inserted or updated, the result is a partial update. To avoid this, use single-row statements, which can be aborted without changing the table.
- For STRICT_TRANS_TABLES, MySQL converts an invalid value to the closest valid value for the column and inserts the adjusted value. If a value is missing, MySQL inserts the implicit default value for the column data type. In either case, MySQL generates a warning rather than an error and continues processing the statement. Implicit defaults are described in Section 11.5, “Data Type Default Values”.

As of MySQL 5.7.4, strict mode affects handling of division by zero, zero dates, and zeros in dates, as follows:

- Strict mode affects handling of division by zero, which includes MOD(*N*,0):

  For data-change operations (INSERT, UPDATE):

  - If strict mode is not enabled, division by zero inserts NULL and produces no warning.
  - If strict mode is enabled, division by zero produces an error, unless IGNORE is given as well. For INSERT IGNORE and UPDATE IGNORE, division by zero inserts NULL and produces a warning.

  For SELECT, division by zero returns NULL. Enabling strict mode causes a warning to be produced as well.

- Strict mode affects whether the server permits '0000-00-00' as a valid date:

  - If strict mode is not enabled, '0000-00-00' is permitted and inserts produce no warning.
  - If strict mode is enabled, '0000-00-00' is not permitted and inserts produce an error, unless IGNORE is given as well. For INSERT IGNORE and UPDATE IGNORE, '0000-00-00' is permitted and inserts produce a warning.

- Strict mode affects whether the server permits dates in which the year part is nonzero but the month or day part is 0 (dates such as '2010-00-01' or '2010-01-00'):

  - If strict mode is not enabled, dates with zero parts are permitted and inserts produce no warning.
  - If strict mode is enabled, dates with zero parts are not permitted and inserts produce an error, unless IGNORE is given as well. For INSERT IGNORE and UPDATE IGNORE, dates with zero parts are inserted as '0000-00-00' (which is considered valid with IGNORE) and produce a warning.

Before MySQL 5.7.4, strict mode affects handling of division by zero, zero dates, and zeros in dates, in conjunction with the ERROR_FOR_DIVISION_BY_ZERO, NO_ZERO_DATE, and NO_ZERO_IN_DATE modes. For details, see the descriptions of those modes.

Combination SQL Modes

The following special modes are provided as shorthand for combinations of mode values from the preceding list.

- ANSI

  Equivalent to REAL_AS_FLOAT, PIPES_AS_CONCAT, ANSI_QUOTES, IGNORE_SPACE.

ANSI mode also causes the server to return an error for queries where a set function *S* with an outer reference *S(outer_ref)* cannot be aggregated in the outer query against which the outer reference has been resolved. This is such a query:

```
SELECT * FROM t1 WHERE t1.a IN (SELECT MAX(t1.b) FROM t2 WHERE ...);
```

Here, MAX(t1.b) cannot aggregated in the outer query because it appears in the WHERE clause of that query. Standard SQL requires an error in this situation. If ANSI mode is not enabled, the server treats *S(outer_ref)* in such queries the same way that it would interpret *S(const)*.

See Section 1.8, “MySQL Standards Compliance”.

- DB2

  Equivalent to PIPES_AS_CONCAT, ANSI_QUOTES, IGNORE_SPACE, NO_KEY_OPTIONS, NO_TABLE_OPTIONS, NO_FIELD_OPTIONS.

- MAXDB

  Equivalent to PIPES_AS_CONCAT, ANSI_QUOTES, IGNORE_SPACE, NO_KEY_OPTIONS, NO_TABLE_OPTIONS, NO_FIELD_OPTIONS, NO_AUTO_CREATE_USER.

- MSSQL

  Equivalent to PIPES_AS_CONCAT, ANSI_QUOTES, IGNORE_SPACE, NO_KEY_OPTIONS, NO_TABLE_OPTIONS, NO_FIELD_OPTIONS.

- MYSQL323

  Equivalent to NO_FIELD_OPTIONS, HIGH_NOT_PRECEDENCE.

- MYSQL40

  Equivalent to NO_FIELD_OPTIONS, HIGH_NOT_PRECEDENCE.

- ORACLE

  Equivalent to PIPES_AS_CONCAT, ANSI_QUOTES, IGNORE_SPACE, NO_KEY_OPTIONS, NO_TABLE_OPTIONS, NO_FIELD_OPTIONS, NO_AUTO_CREATE_USER.

- POSTGRESQL

  Equivalent to PIPES_AS_CONCAT, ANSI_QUOTES, IGNORE_SPACE, NO_KEY_OPTIONS, NO_TABLE_OPTIONS, NO_FIELD_OPTIONS.

- TRADITIONAL

  Before MySQL 5.7.4, TRADITIONAL is equivalent to STRICT_TRANS_TABLES, STRICT_ALL_TABLES, NO_ZERO_IN_DATE, NO_ZERO_DATE, ERROR_FOR_DIVISION_BY_ZERO, NO_AUTO_CREATE_USER, and NO_ENGINE_SUBSTITUTION.

  As of MySQL 5.7.4, TRADITIONAL is equivalent to STRICT_TRANS_TABLES, STRICT_ALL_TABLES, NO_AUTO_CREATE_USER, and NO_ENGINE_SUBSTITUTION. The NO_ZERO_IN_DATE, NO_ZERO_DATE, and ERROR_FOR_DIVISION_BY_ZERO modes are not included as of MySQL 5.7.4 because they do nothing. Instead, their pre-5.7.4 effects are included in the effects of strict SQL mode (STRICT_ALL_TABLES or STRICT_TRANS_TABLES).

SQL Mode Changes in MySQL 5.7

As of MySQL 5.7.4, the deprecated ERROR_FOR_DIVISION_BY_ZERO, NO_ZERO_DATE, and NO_ZERO_IN_DATE SQL modes do nothing. Instead, their previous effects are included in the effects of strict SQL mode (STRICT_ALL_TABLES or STRICT_TRANS_TABLES). In other words, strict mode means the same thing as the pre-5.7.4 meaning of strict mode plus the ERROR_FOR_DIVISION_BY_ZERO, NO_ZERO_DATE, and NO_ZERO_IN_DATE modes. This change reduces the number of SQL modes with an effect dependent on strict mode and makes them part of strict mode itself.

This section describes the SQL mode settings to use in MySQL 5.7.4 and up to achieve the same statement execution as before 5.7.4, including the cases for INSERT and UPDATE in which IGNORE is given. It also provides guidelines for determining whether applications need modification to behave the same before and after the SQL mode changes.

The following table shows how to control handling of division by zero before and after the SQL mode changes in MySQL 5.7.4.

Desired Behavior	Old Settings	New Settings
insert NULL, produce no warning	ERROR_FOR_DIVISION_BY_ZERO not enabled	strict mode not enabled
insert NULL, produce warning	ERROR_FOR_DIVISION_BY_ZERO, or ERROR_FOR_DIVISION_BY_ZERO + strict mode + IGNORE	strict mode + IGNORE
error	ERROR_FOR_DIVISION_BY_ZERO + strict mode	strict mode

The following table shows how to control whether the server permits '0000-00-00' as a valid date before and after the SQL mode changes in MySQL 5.7.4.

Desired Behavior	Old Settings	New Settings
insert '0000-00-00', produce no warning	NO_ZERO_DATE not enabled	strict mode not enabled
insert '0000-00-00', produce warning	NO_ZERO_DATE, or NO_ZERO_DATE + strict mode + IGNORE	strict mode + IGNORE
error	NO_ZERO_DATE + strict mode	strict mode

The following table shows how to control whether the server permits dates with zero parts before and after the SQL mode changes in MySQL 5.7.4.

Desired Behavior	Old Settings	New Settings
insert date, produce no warning	NO_ZERO_IN_DATE not enabled	strict mode not enabled
insert '0000-00-00', produce warning	NO_ZERO_IN_DATE, or NO_ZERO_IN_DATE + strict mode + IGNORE	strict mode + IGNORE
error	NO_ZERO_IN_DATE + strict mode	strict mode

The following discussion describes the conditions under which a given statement produces the same or different result as of the SQL mode changes in MySQL 5.7.4. It considers only strict mode (STRICT_ALL_TABLES or STRICT_TRANS_TABLES) and the three deprecated modes (ERROR_FOR_DIVISION_BY_ZERO, NO_ZERO_DATE, and NO_ZERO_IN_DATE). Other SQL modes such as ANSI_QUOTES or ONLY_FULL_GROUP_BY are assumed to be held constant before and after an upgrade.

This discussion also describes how to prepare for an upgrade to 5.7.4 or later from a version older than 5.7.4. *Any modifications should be made before upgrading.*

There is no change in behavior between MySQL 5.6 and 5.7 for the following SQL mode settings. A statement that executes under one of these settings needs no modification to produce the same result in 5.6 and 5.7:

- Strict mode and the three deprecated modes are all not enabled.
- Strict mode and the three deprecated modes are all enabled.

A change from warnings in MySQL 5.6 to no warnings in MySQL 5.7 occurs for the following SQL mode settings. The result of statement execution is the same in 5.6 and 5.7, so statements need no modification unless warnings are considered significant:

- Strict mode is not enabled, but either of the deprecated `ERROR_FOR_DIVISION_BY_ZERO` and `NO_ZERO_DATE` modes are enabled.

A behavior change occurs under the following SQL mode settings. A statement that executes under one of these settings must be modified to produce the same result in 5.6 and 5.7:

- Strict mode is not enabled, `NO_ZERO_IN_DATE` is enabled. For this mode setting, expect these differences in statement execution:
  - In 5.6, the server inserts dates with zero parts as `'0000-00-00'` and produces a warning.
  - In 5.7, the server inserts dates with zero parts as is and produces no warning.
- Strict mode is enabled, with some but not all of the three deprecated modes enabled. For this mode setting, expect these differences in statement execution:

  Statements that would be affected by enabling the not-enabled deprecated modes produce errors in 5.7 but not in 5.6. Suppose that strict mode, `NO_ZERO_DATE`, and `NO_ZERO_IN_DATE` are enabled, and a data-change statement performs division by zero:
  - In 5.6, the statement inserts `NULL` and produces no warning. Enabling `ERROR_FOR_DIVISION_BY_ZERO` would cause an error instead.
  - In 5.7, an error occurs because strict mode implicitly includes the effect of `ERROR_FOR_DIVISION_BY_ZERO`. Enabling `ERROR_FOR_DIVISION_BY_ZERO` explicitly would not change that.

To prepare for an upgrade to MySQL 5.7.4 or later, the main principle is to make sure that your applications will operate the same way in MySQL 5.6 and 5.7. For example, you can adopt either of these approaches to application compatibility:

- Modify the application to set the SQL mode on a version-specific basis. If we assume that an application will not be used with development versions of MySQL 5.7 prior to 5.7.4, it is possible to set the `sql_mode` value for the application based on the current server version as follows:

  ```
  SET sql_mode = IF(LEFT(VERSION(),3)<'5.7',5.6 mode,5.7 mode);
  ```

  The tables shown earlier in this section serve as a guide to the appropriate equivalent modes for MySQL 5.6 and 5.7.
- Modify the application to execute under a SQL mode for which statements produce the same result in MySQL 5.6 and 5.7.

Tip

TRADITIONAL SQL mode in MySQL 5.6 includes strict mode and the three deprecated modes. If you write applications to operate in TRADITIONAL mode in MySQL 5.6, there is no change to make for MySQL 5.7.

When assessing SQL mode compatibility between MySQL 5.6 and 5.7, consider particularly these statement execution contexts:

- Replication. You will encounter replication incompatibility related to the SQL mode changes under the following conditions:

  - MySQL 5.6 master and 5.7 slave
  - Statement-based replication
  - A SQL mode setting for which statements produce different results in MySQL 5.6 and 5.7, as described earlier

  To handle this incompatibility, use one of these workarounds:

  - Use row-based replication
  - Use IGNORE
  - Use a SQL mode for which statements do not produce different results in MySQL 5.6 and 5.7

- Stored programs (stored procedures and functions, triggers, and events). Each stored program executes using the SQL mode in effect at the time it was created. To identify stored programs that may be affected by differences between MySQL 5.6 and 5.7 in SQL mode handling, use these queries:

```
SELECT ROUTINE_SCHEMA, ROUTINE_NAME, ROUTINE_TYPE, SQL_MODE
FROM INFORMATION_SCHEMA.ROUTINES
WHERE SQL_MODE LIKE '%STRICT%'
OR SQL_MODE LIKE '%DIVISION%'
OR SQL_MODE LIKE '%NO_ZERO%';

SELECT TRIGGER_SCHEMA, TRIGGER_NAME, SQL_MODE
FROM INFORMATION_SCHEMA.TRIGGERS
WHERE SQL_MODE LIKE '%STRICT%'
OR SQL_MODE LIKE '%DIVISION%'
OR SQL_MODE LIKE '%NO_ZERO%';

SELECT EVENT_SCHEMA, EVENT_NAME, SQL_MODE
FROM INFORMATION_SCHEMA.EVENTS
WHERE SQL_MODE LIKE '%STRICT%'
OR SQL_MODE LIKE '%DIVISION%'
OR SQL_MODE LIKE '%NO_ZERO%';
```

5.1.8 Server Plugins

MySQL supports a plugin API that enables creation of server components. Plugins can be loaded at server startup, or loaded and unloaded at runtime without restarting the server. The components supported by this interface include, but are not limited to, storage engines, full-text parser plugins, partitioning support, and server extensions.

5.1.8.1 Installing and Uninstalling Plugins

Server plugins must be loaded in to the server before they can be used. MySQL enables you to load a plugin at server startup or at runtime. It is also possible to control the activation of loaded plugins at startup, and to unload them at runtime.

- Installing plugins
- Controlling plugin activation
- Uninstalling plugins

Installing Plugins

Server plugins must be known to the server before they can be used. A plugin can be made known several ways, as described here. In the following descriptions, *`plugin_name`* stands for a plugin name such as `innodb` or `csv`.

Built-in plugins:

A plugin that is built in to the server is known by the server automatically. Normally, the server enables the plugin at startup, although this can be changed with the `--`*`plugin_name`* option.

Plugins registered in the `mysql.plugin` table:

The `mysql.plugin` table serves as a registry of plugins. The server normally enables each plugin listed in the table at startup, although whether a given plugin is enabled can be changed with the `--`*`plugin_name`* option. If the server is started with the `--skip-grant-tables` option, it does not consult this table and does not load the plugins listed there.

Plugins named with command-line options:

A plugin that is located in a plugin library file can be loaded at server startup with the `--plugin-load` option. Normally, the server enables the plugin at startup, although this can be changed with the `--`*`plugin_name`* option.

The option value is a semicolon-separated list of *`name`*`=`*`plugin_library`* pairs. Each *`name`* is the name of the plugin, and *`plugin_library`* is the name of the shared library that contains the plugin code. If a plugin library is named without any preceding plugin name, the server loads all plugins in the library. Each library file must be located in the directory named by the `plugin_dir` system variable.

This option does not register any plugin in the `mysql.plugin` table. For subsequent restarts, the server loads the plugin again only if `--plugin-load` is given again. That is, this option effects a one-time installation that persists only for one server invocation.

`--plugin-load` enables plugins to be loaded even when `--skip-grant-tables` is given (which causes the server to ignore the `mysql.plugin` table). `--plugin-load` also enables plugins to be loaded at startup under configurations when plugins cannot be loaded at runtime.

The `--plugin-load-add` option complements the `--plugin-load` option. `--plugin-load-add` adds a plugin or plugins to the set of plugins to be loaded at startup. The argument format is the same as for `--plugin-load`. `--plugin-load-add` can be used to avoid specifying a large set of plugins as a single long unwieldy `--plugin-load`. argument. `--plugin-load-add` can be given in the absence of `--plugin-load`, but any instance of `--plugin-load-add` that appears before `--plugin-load`. has no effect because `--plugin-load` resets the set of plugins to load. In other words, these options:

```
--plugin-load=x --plugin-load-add=y
```

are equivalent to this option:

```
--plugin-load="x;y"
```

But these options:

```
--plugin-load-add=y --plugin-load=x
```

are equivalent to this option:

```
--plugin-load=x
```

Plugins installed with the `INSTALL PLUGIN` statement:

A plugin that is located in a plugin library file can be loaded at runtime with the `INSTALL PLUGIN` statement. The statement also registers the plugin in the `mysql.plugin` table to cause the server to load it on subsequent restarts. For this reason, `INSTALL PLUGIN` requires the `INSERT` privilege for the `mysql.plugin` table.

If a plugin is named both using a `--plugin-load` option and in the `mysql.plugin` table, the server starts but writes these messages to the error log:

```
2013-09-24T12:35:29.584584Z 0 [ERROR] Function 'plugin_name'
already exists
2013-09-24T12:35:29.584616Z 0 [Warning] Couldn't load plugin named
'plugin_name' with soname 'plugin_object_file'.
```

Example: The `--plugin-load` option installs a plugin at server startup. To install a plugin named `myplugin` in a plugin library file named `somepluglib.so`, use these lines in a `my.cnf` file:

```
[mysqld]
plugin-load=myplugin=somepluglib.so
```

In this case, the plugin is not registered in `mysql.plugin`. Restarting the server without the `--plugin-load` option causes the plugin not to be loaded at startup.

Alternatively, the `INSTALL PLUGIN` statement causes the server to load the plugin code from the library file at runtime:

```
mysql> INSTALL PLUGIN myplugin SONAME 'somepluglib.so';
```

`INSTALL PLUGIN` also causes “permanent” plugin registration: The server lists the plugin in the `mysql.plugin` table to ensure that it is loaded on subsequent server restarts.

Many plugins can be loaded either at server startup or at runtime. However, if a plugin is designed such that it must be loaded and initialized during server startup, use `--plugin-load` rather than `INSTALL PLUGIN`.

While a plugin is loaded, information about it is available at runtime from several sources, such as the `INFORMATION_SCHEMA.PLUGINS` table and the `SHOW PLUGINS` statement. For more information, see Section 5.1.8.2, “Obtaining Server Plugin Information”.

Controlling Plugin Activation

If the server knows about a plugin when it starts (for example, because the plugin is named using a `--plugin-load` option or registered in the `mysql.plugin` table), the server loads and enables the plugin by default. It is possible to control activation for such a plugin using a `--plugin_name[=value]` startup option named after the plugin. In the following descriptions, `plugin_name` stands for a plugin name such as `innodb` or `csv`. As with other options, dashes and underscores are interchangeable in option names. For example, `--my_plugin=ON` and `--my-plugin=ON` are equivalent.

- `--plugin_name=OFF`

  Tells the server to disable the plugin.

- `--plugin_name[=ON]`

  Tells the server to enable the plugin. (Specifying the option as `--plugin_name` without a value has the same effect.) If the plugin fails to initialize, the server runs with the plugin disabled.

- `--plugin_name=FORCE`

  Tells the server to enable the plugin, but if plugin initialization fails, the server does not start. In other words, this option forces the server to run with the plugin enabled or not at all.

- `--plugin_name=FORCE_PLUS_PERMANENT`

  Like `FORCE`, but in addition prevents the plugin from being unloaded at runtime. If a user attempts to do so with `UNINSTALL PLUGIN`, an error occurs.

The values `OFF`, `ON`, `FORCE`, and `FORCE_PLUS_PERMANENT` are not case sensitive.

The activation state for plugins is visible in the `LOAD_OPTION` column of the `INFORMATION_SCHEMA.PLUGINS` table.

Suppose that `CSV`, `BLACKHOLE`, and `ARCHIVE` are built-in pluggable storage engines and that you want the server to load them at startup, subject to these conditions: The server is permitted to run if `CSV` initialization fails, but must require that `BLACKHOLE` initialization succeeds, and `ARCHIVE` should be disabled. To accomplish that, use these lines in an option file:

```
[mysqld]
csv=ON
blackhole=FORCE
archive=OFF
```

The `--enable-plugin_name` option format is supported as a synonym for `--plugin_name=ON`. The `--disable-plugin_name` and `--skip-plugin_name` option formats are supported as synonyms for `--plugin_name=OFF`.

If a plugin is disabled, either explicitly with `OFF` or implicitly because it was enabled with `ON` but failed to initialize, aspects of server operation that require the plugin will change. For example, if the plugin implements a storage engine, existing tables for the storage engine become inaccessible, and attempts to create new tables for the storage engine result in tables that use the default storage engine unless the `NO_ENGINE_SUBSTITUTION` SQL mode has been enabled to cause an error to occur instead.

Disabling a plugin may require adjustment to other options. For example, if you start the server using `--skip-innodb` to disable `InnoDB`, other `innodb_xxx` options likely will need to be omitted from the startup command. In addition, because `InnoDB` is the default storage engine, it will not start unless you specify another available storage engine with `--default_storage_engine`. You must also set `--default_tmp_storage_engine`.

Uninstalling Plugins

A plugin known to the server can be uninstalled to disable it at runtime with the `UNINSTALL PLUGIN` statement. The statement unloads the plugin and removes it from the `mysql.plugin` table if it is registered there. For this reason, UNINSTALL PLUGIN statement requires the DELETE privilege for the mysql.plugin table. With the plugin no longer registered in the table, the server will not load the plugin automatically for subsequent restarts.

UNINSTALL PLUGIN can unload plugins regardless of whether they were loaded with INSTALL PLUGIN or --plugin-load.

UNINSTALL PLUGIN is subject to these exceptions:

- It cannot unload plugins that are built in to the server. These can be identified as those that have a library name of NULL in the output from INFORMATION_SCHEMA.PLUGINS or SHOW PLUGINS.

- It cannot unload plugins for which the server was started with --*plugin_name*=FORCE_PLUS_PERMANENT, which prevents plugin unloading at runtime. These can be identified from the LOAD_OPTION column of the INFORMATION_SCHEMA.PLUGINS table.

5.1.8.2 Obtaining Server Plugin Information

There are several ways to determine which plugins are installed in the server:

- The INFORMATION_SCHEMA.PLUGINS table contains a row for each loaded plugin. Any that have a PLUGIN_LIBRARY value of NULL are built in and cannot be unloaded.

```
mysql> SELECT * FROM information_schema.PLUGINS\G
*************************** 1. row ***************************
           PLUGIN_NAME: binlog
        PLUGIN_VERSION: 1.0
         PLUGIN_STATUS: ACTIVE
           PLUGIN_TYPE: STORAGE ENGINE
   PLUGIN_TYPE_VERSION: 50158.0
        PLUGIN_LIBRARY: NULL
PLUGIN_LIBRARY_VERSION: NULL
         PLUGIN_AUTHOR: MySQL AB
    PLUGIN_DESCRIPTION: This is a pseudo storage engine to represent the binlog in a transaction
        PLUGIN_LICENSE: GPL
           LOAD_OPTION: FORCE
...
*************************** 10. row ***************************
           PLUGIN_NAME: InnoDB
        PLUGIN_VERSION: 1.0
         PLUGIN_STATUS: ACTIVE
           PLUGIN_TYPE: STORAGE ENGINE
   PLUGIN_TYPE_VERSION: 50158.0
        PLUGIN_LIBRARY: ha_innodb_plugin.so
PLUGIN_LIBRARY_VERSION: 1.0
         PLUGIN_AUTHOR: Innobase Oy
    PLUGIN_DESCRIPTION: Supports transactions, row-level locking,
                        and foreign keys
        PLUGIN_LICENSE: GPL
           LOAD_OPTION: ON
...
```

- The SHOW PLUGINS statement displays a row for each loaded plugin. Any that have a Library value of NULL are built in and cannot be unloaded.

```
mysql> SHOW PLUGINS\G
*************************** 1. row ***************************
   Name: binlog
 Status: ACTIVE
   Type: STORAGE ENGINE
Library: NULL
```

```
License: GPL
...
*************************** 10. row ***************************
   Name: InnoDB
 Status: ACTIVE
   Type: STORAGE ENGINE
Library: ha_innodb_plugin.so
License: GPL
...
```

- The `mysql.plugin` table shows which plugins have been registered with `INSTALL PLUGIN`. The table contains only plugin names and library file names, so it does not provide as much information as the `PLUGINS` table or the `SHOW PLUGINS` statement.

5.1.9 IPv6 Support

Support for IPv6 in MySQL includes these capabilities:

- MySQL Server can accept TCP/IP connections from clients connecting over IPv6. For example, this command connects over IPv6 to the MySQL server on the local host:

  ```
  shell> mysql -h ::1
  ```

  To use this capability, two things must be true:

  - Your system must be configured to support IPv6. See Section 5.1.9.1, “Verifying System Support for IPv6”.

  - The default MySQL server configuration permits only IPv4 connections, so the server must be configured for IPv6 connections. To permit IPv6 connections in addition to or instead of IPv4 connections, start the server with an appropriate `--bind-address` option. See Section 5.1.4, “Server System Variables”.

- MySQL account names permit IPv6 addresses to enable DBAs to specify privileges for clients that connect to the server over IPv6. See Section 6.2.3, “Specifying Account Names”. IPv6 addresses can be specified in account names in statements such as `CREATE USER`, `GRANT`, and `REVOKE`. For example:

  ```
  mysql> CREATE USER 'bill'@'::1' IDENTIFIED BY 'secret';
  mysql> GRANT SELECT ON mydb.* TO 'bill'@'::1';
  ```

- IPv6 functions enable conversion between string and internal format IPv6 address formats, and checking whether values represent valid IPv6 addresses. For example, `INET6_ATON()` and `INET6_NTOA()` are similar to `INET_ATON()` and `INET_NTOA()`, but handle IPv6 addresses in addition to IPv4 addresses. See Section 12.16, “Miscellaneous Functions”.

The following sections describe how to set up MySQL so that clients can connect to the server over IPv6.

5.1.9.1 Verifying System Support for IPv6

Before MySQL Server can accept IPv6 connections, the operating system on your server host must support IPv6. As a simple test to determine whether that is true, try this command:

```
shell> ping6 ::1
16 bytes from ::1, icmp_seq=0 hlim=64 time=0.171 ms
16 bytes from ::1, icmp_seq=1 hlim=64 time=0.077 ms
...
```

To produce a description of your system's network interfaces, invoke `ifconfig -a` and look for IPv6 addresses in the output.

If your host does not support IPv6, consult your system documentation for instructions on enabling it. It might be that you need only reconfigure an existing network interface to add an IPv6 address. Or a more extensive change might be needed, such as rebuilding the kernel with IPv6 options enabled.

These links may be helpful in setting up IPv6 on various platforms:

- Windows XP
- Gentoo Linux
- Ubuntu Linux
- Linux (Generic)
- Mac OS X

5.1.9.2 Configuring the MySQL Server to Permit IPv6 Connections

The MySQL server listens on a single network socket for TCP/IP connections. This socket is bound to a single address, but it is possible for an address to map onto multiple network interfaces. To specify an address, use the `--bind-address=addr` option at server startup, where *`addr`* is an IPv4 or IPv6 address or a host name. (IPv6 addresses are not supported before MySQL 5.5.3.) If *`addr`* is a host name, the server resolves the name to an IP address and binds to that address.

The server treats different types of addresses as follows:

- If the address is `*`, the server accepts TCP/IP connections on all server host IPv6 and IPv4 interfaces if the server host supports IPv6, or accepts TCP/IP connections on all IPv4 addresses otherwise. Use this address to permit both IPv4 and IPv6 connections on all server interfaces. This value is the default.
- If the address is `0.0.0.0`, the server accepts TCP/IP connections on all server host IPv4 interfaces.
- If the address is `::`, the server accepts TCP/IP connections on all server host IPv4 and IPv6 interfaces. Use this address to permit both IPv4 and IPv6 connections on all server interfaces.
- If the address is an IPv4-mapped address, the server accepts TCP/IP connections for that address, in either IPv4 or IPv6 format. For example, if the server is bound to `::ffff:127.0.0.1`, clients can connect using `--host=127.0.0.1` or `--host=::ffff:127.0.0.1`.
- If the address is a “regular” IPv4 or IPv6 address (such as `127.0.0.1` or `::1`), the server accepts TCP/IP connections only for that IPv4 or IPv6 address.

If you intend to bind the server to a specific address, be sure that the `mysql.user` grant table contains an account with administrative privileges that you can use to connect to that address. Otherwise, you will not be able to shut down the server. For example, if you bind the server to `*`, you can connect to it using all existing accounts. But if you bind the server to `::1`, it accepts connections only on that address. In that case, first make sure that the `'root'@'::1'` account is present in the `mysql.user` table so you can still connect to the server to shut it down.

5.1.9.3 Connecting Using the IPv6 Local Host Address

The following procedure shows how to configure MySQL to permit IPv6 connections by clients that connect to the local server using the `::1` local host address. The instructions given here assume that your system supports IPv6.

1. Start the MySQL server with an appropriate `--bind-address` option to permit it to accept IPv6 connections. For example, put the following lines in your server option file and restart the server:

   ```
   [mysqld]
   bind-address = *
   ```

   Alternatively, you can bind the server to `::1`, but that makes the server more restrictive for TCP/IP connections. It accepts only IPv6 connections for that single address and rejects IPv4 connections. For more information, see Section 5.1.9.2, “Configuring the MySQL Server to Permit IPv6 Connections”.

2. As an administrator, connect to the server and create an account for a local user who will connect from the `::1` local IPv6 host address:

   ```
   mysql> CREATE USER 'ipv6user'@'::1' IDENTIFIED BY 'ipv6pass';
   ```

   For the permitted syntax of IPv6 addresses in account names, see Section 6.2.3, “Specifying Account Names”. In addition to the `CREATE USER` statement, you can issue `GRANT` statements that give specific privileges to the account, although that is not necessary for the remaining steps in this procedure.

3. Invoke the `mysql` client to connect to the server using the new account:

   ```
   shell> mysql -h ::1 -u ipv6user -pipv6pass
   ```

4. Try some simple statements that show connection information:

   ```
   mysql> STATUS
   ...
   Connection:   ::1 via TCP/IP
   ...

   mysql> SELECT CURRENT_USER(), @@bind_address;
   +----------------+----------------+
   | CURRENT_USER() | @@bind_address |
   +----------------+----------------+
   | ipv6user@::1   | ::             |
   +----------------+----------------+
   ```

5.1.9.4 Connecting Using IPv6 Nonlocal Host Addresses

The following procedure shows how to configure MySQL to permit IPv6 connections by remote clients. It is similar to the preceding procedure for local clients, but the server and client hosts are distinct and each has its own nonlocal IPv6 address. The example uses these addresses:

```
Server host: 2001:db8:0:f101::1
Client host: 2001:db8:0:f101::2
```

These addresses are chosen from the nonroutable address range recommended by IANA for documentation purposes and suffice for testing on your local network. To accept IPv6 connections from clients outside the local network, the server host must have a public address. If your network provider assigns you an IPv6 address, you can use that. Otherwise, another way to obtain an address is to use an IPv6 broker; see Section 5.1.9.5, “Obtaining an IPv6 Address from a Broker”.

1. Start the MySQL server with an appropriate `--bind-address` option to permit it to accept IPv6 connections. For example, put the following lines in your server option file and restart the server:

   ```
   [mysqld]
   ```

```
bind-address = *
```

Alternatively, you can bind the server to 2001:db8:0:f101::1, but that makes the server more restrictive for TCP/IP connections. It accepts only IPv6 connections for that single address and rejects IPv4 connections. For more information, see Section 5.1.9.2, “Configuring the MySQL Server to Permit IPv6 Connections”.

2. On the server host (2001:db8:0:f101::1), create an account for a user who will connect from the client host (2001:db8:0:f101::2):

```
mysql> CREATE USER 'remoteipv6user'@'2001:db8:0:f101::2' IDENTIFIED BY 'remoteipv6pass';
```

3. On the client host (2001:db8:0:f101::2), invoke the mysql client to connect to the server using the new account:

```
shell> mysql -h 2001:db8:0:f101::1 -u remoteipv6user -premoteipv6pass
```

4. Try some simple statements that show connection information:

```
mysql> STATUS
...
Connection:   2001:db8:0:f101::1 via TCP/IP
...

mysql> SELECT CURRENT_USER(), @@bind_address;
+-----------------------------------+----------------+
| CURRENT_USER()                    | @@bind_address |
+-----------------------------------+----------------+
| remoteipv6user@2001:db8:0:f101::2 | ::             |
+-----------------------------------+----------------+
```

5.1.9.5 Obtaining an IPv6 Address from a Broker

If you do not have a public IPv6 address that enables your system to communicate over IPv6 outside your local network, you can obtain one from an IPv6 broker. The Wikipedia IPv6 Tunnel Broker page lists several brokers and their features, such as whether they provide static addresses and the supported routing protocols.

After configuring your server host to use a broker-supplied IPv6 address, start the MySQL server with an appropriate --bind-address option to permit the server to accept IPv6 connections. For example, put the following lines in the server option file and restart the server:

```
[mysqld]
bind-address = *
```

Alternatively, you can bind the server to to the specific IPv6 address provided by the broker, but that makes the server more restrictive for TCP/IP connections. It accepts only IPv6 connections for that single address and rejects IPv4 connections. For more information, see Section 5.1.9.2, “Configuring the MySQL Server to Permit IPv6 Connections”. In addition, if the broker allocates dynamic addresses, the address provided for your system might change the next time you connect to the broker. If so, any accounts you create that name the original address become invalid. To bind to a specific address but avoid this change-of-address problem, you may be able to arrange with the broker for a static IPv6 address.

The following example shows how to use Freenet6 as the broker and the gogoc IPv6 client package on Gentoo Linux.

1. Create a account at Freenet6 by visiting this URL and signing up:

```
http://gogonet.gogo6.com
```

2. After creating the account, go to this URL, sign in, and create a user ID and password for the IPv6 broker:

   ```
   http://gogonet.gogo6.com/page/freenet6-registration
   ```

3. As root, install gogoc:

   ```
   shell> emerge gogoc
   ```

4. Edit /etc/gogoc/gogoc.conf to set the userid and password values. For example:

   ```
   userid=gogouser
   passwd=gogopass
   ```

5. Start gogoc:

   ```
   shell> /etc/init.d/gogoc start
   ```

   To start gogoc each time your system boots, execute this command:

   ```
   shell> rc-update add gogoc default
   ```

6. Use ping6 to try to ping a host:

   ```
   shell> ping6 ipv6.google.com
   ```

7. To see your IPv6 address:

   ```
   shell> ifconfig tun
   ```

5.1.10 Server-Side Help

MySQL Server supports a HELP statement that returns online information from the MySQL Reference manual (see Section 13.8.3, “HELP Syntax”). The proper operation of this statement requires that the help tables in the mysql database be initialized with help topic information, which is done by processing the contents of the fill_help_tables.sql script.

If you install MySQL using a binary or source distribution on Unix, help table content initialization occurs when you run mysql_install_db. For an RPM distribution on Linux or binary distribution on Windows, content initialization occurs as part of the MySQL installation process.

If you upgrade MySQL using a binary distribution, help table content is not upgraded automatically, but you can upgrade it manually. Locate the fill_help_tables.sql file in the share or share/mysql directory. Change location into that directory and process the file with the mysql client as follows:

```
shell> mysql -u root mysql < fill_help_tables.sql
```

You can also obtain the latest fill_help_tables.sql at any time to upgrade your help tables. Download the proper file for your version of MySQL from http://dev.mysql.com/doc/index-other.html. After downloading and uncompressing the file, process it with mysql as described previously.

If you are working with Bazaar and a MySQL development source tree, you must use a downloaded copy of the `fill_help_tables.sql` file because the source tree contains only a “stub” version.

> **Note**
>
> For a server that participates in replication, the help table content upgrade process involves multiple servers. For details, see Section 16.4.1.27, “Replication of Server-Side Help Tables”.

5.1.11 Server Response to Signals

On Unix, signals can be sent to processes. `mysqld` responds to signals sent to it as follows:

- `SIGTERM` causes the server to shut down.
- `SIGHUP` causes the server to reload the grant tables and to flush tables, logs, the thread cache, and the host cache. These actions are like various forms of the `FLUSH` statement. The server also writes a status report to the error log that has this format:

```
Status information:

Current dir: /var/mysql/data/
Running threads: 0  Stack size: 196608
Current locks:

Key caches:
default
Buffer_size:       8388600
Block_size:           1024
Division_limit:        100
Age_limit:             300
blocks used:             0
not flushed:             0
w_requests:              0
writes:                  0
r_requests:              0
reads:                   0

handler status:
read_key:            0
read_next:           0
read_rnd             0
read_first:          1
write:               0
delete               0
update:              0

Table status:
Opened tables:          5
Open tables:            0
Open files:             7
Open streams:           0

Alarm status:
Active alarms:   1
Max used alarms: 2
Next alarm time: 67
```

On some Mac OS X 10.3 versions, `mysqld` ignores `SIGHUP` and `SIGQUIT`.

5.1.12 The Shutdown Process

The server shutdown process takes place as follows:

1. The shutdown process is initiated.

   This can occur initiated several ways. For example, a user with the `SHUTDOWN` privilege can execute a `mysqladmin shutdown` command. `mysqladmin` can be used on any platform supported by MySQL. Other operating system-specific shutdown initiation methods are possible as well: The server shuts down on Unix when it receives a `SIGTERM` signal. A server running as a service on Windows shuts down when the services manager tells it to.

2. The server creates a shutdown thread if necessary.

   Depending on how shutdown was initiated, the server might create a thread to handle the shutdown process. If shutdown was requested by a client, a shutdown thread is created. If shutdown is the result of receiving a `SIGTERM` signal, the signal thread might handle shutdown itself, or it might create a separate thread to do so. If the server tries to create a shutdown thread and cannot (for example, if memory is exhausted), it issues a diagnostic message that appears in the error log:

   ```
   Error: Can't create thread to kill server
   ```

3. The server stops accepting new connections.

   To prevent new activity from being initiated during shutdown, the server stops accepting new client connections by closing the handlers for the network interfaces to which it normally listens for connections: the TCP/IP port, the Unix socket file, the Windows named pipe, and shared memory on Windows.

4. The server terminates current activity.

   For each thread associated with a client connection, the server breaks the connection to the client and marks the thread as killed. Threads die when they notice that they are so marked. Threads for idle connections die quickly. Threads that currently are processing statements check their state periodically and take longer to die. For additional information about thread termination, see Section 13.7.6.4, “`KILL` Syntax”, in particular for the instructions about killed `REPAIR TABLE` or `OPTIMIZE TABLE` operations on `MyISAM` tables.

   For threads that have an open transaction, the transaction is rolled back. Note that if a thread is updating a nontransactional table, an operation such as a multiple-row `UPDATE` or `INSERT` may leave the table partially updated because the operation can terminate before completion.

   If the server is a master replication server, it treats threads associated with currently connected slaves like other client threads. That is, each one is marked as killed and exits when it next checks its state.

   If the server is a slave replication server, it stops the I/O and SQL threads, if they are active, before marking client threads as killed. The SQL thread is permitted to finish its current statement (to avoid causing replication problems), and then stops. If the SQL thread is in the middle of a transaction at this point, the server waits until the current replication event group (if any) has finished executing, or until the user issues a `KILL QUERY` or `KILL CONNECTION` statement. See also Section 13.4.2.7, “`STOP SLAVE` Syntax”. Since nontransactional statements cannot be rolled back, in order to guarantee crash-safe replication, only transactional tables should be used.

   Note

   In order to guarantee crash safety on the slave, you must also run the slave with `--relay-log-recovery` enabled.

See also Section 16.2.2, “Replication Relay and Status Logs”).

5. The server shuts down or closes storage engines.

   At this stage, the server flushes the table cache and closes all open tables.

   Each storage engine performs any actions necessary for tables that it manages. InnoDB flushes its buffer pool to disk (unless innodb_fast_shutdown is 2), writes the current LSN to the tablespace, and terminates its own internal threads. MyISAM flushes any pending index writes for a table.

6. The server exits.

5.2 MySQL Server Logs

MySQL Server has several logs that can help you find out what activity is taking place.

Log Type	Information Written to Log
Error log	Problems encountered starting, running, or stopping mysqld
General query log	Established client connections and statements received from clients
Binary log	Statements that change data (also used for replication)
Relay log	Data changes received from a replication master server
Slow query log	Queries that took more than long_query_time seconds to execute

By default, no logs are enabled (except the error log on Windows). The following log-specific sections provide information about the server options that enable logging.

By default, the server writes files for all enabled logs in the data directory. You can force the server to close and reopen the log files (or in some cases switch to a new log file) by flushing the logs. Log flushing occurs when you issue a FLUSH LOGS statement; execute mysqladmin with a flush-logs or refresh argument; or execute mysqldump with a --flush-logs or --master-data option. See Section 13.7.6.3, “FLUSH Syntax”, Section 4.5.2, “mysqladmin — Client for Administering a MySQL Server”, and Section 4.5.4, “mysqldump — A Database Backup Program”. In addition, the binary log is flushed when its size reaches the value of the max_binlog_size system variable.

You can control the general query and slow query logs during runtime. You can enable or disable logging, or change the log file name. You can tell the server to write general query and slow query entries to log tables, log files, or both. For details, see Section 5.2.1, “Selecting General Query and Slow Query Log Output Destinations”, Section 5.2.3, “The General Query Log”, and Section 5.2.5, “The Slow Query Log”.

The relay log is used only on slave replication servers, to hold data changes from the master server that must also be made on the slave. For discussion of relay log contents and configuration, see Section 16.2.2.1, “The Slave Relay Log”.

For information about log maintenance operations such as expiration of old log files, see Section 5.2.6, “Server Log Maintenance”.

For information about keeping logs secure, see Section 6.1.2.3, “Passwords and Logging”.

5.2.1 Selecting General Query and Slow Query Log Output Destinations

MySQL Server provides flexible control over the destination of output to the general query log and the slow query log, if those logs are enabled. Possible destinations for log entries are log files or the general_log and slow_log tables in the mysql database. Either or both destinations can be selected.

Log control at server startup. The `--log-output` option specifies the destination for log output. This option does not in itself enable the logs. Its syntax is `--log-output[=value,...]`:

- If `--log-output` is given with a value, the value should be a comma-separated list of one or more of the words `TABLE` (log to tables), `FILE` (log to files), or `NONE` (do not log to tables or files). `NONE`, if present, takes precedence over any other specifiers.
- If `--log-output` is omitted, the default logging destination is `FILE`.

The `general_log` system variable controls logging to the general query log for the selected log destinations. If specified at server startup, `general_log` takes an optional argument of 1 or 0 to enable or disable the log. To specify a file name other than the default for file logging, set the `general_log_file` variable. Similarly, the `slow_query_log` variable controls logging to the slow query log for the selected destinations and setting `slow_query_log_file` specifies a file name for file logging. If either log is enabled, the server opens the corresponding log file and writes startup messages to it. However, further logging of queries to the file does not occur unless the `FILE` log destination is selected.

Examples:

- To write general query log entries to the log table and the log file, use `--log-output=TABLE,FILE` to select both log destinations and `--general_log` to enable the general query log.
- To write general and slow query log entries only to the log tables, use `--log-output=TABLE` to select tables as the log destination and `--general_log` and `--slow_query_log` to enable both logs.
- To write slow query log entries only to the log file, use `--log-output=FILE` to select files as the log destination and `--slow_query_log` to enable the slow query log. (In this case, because the default log destination is `FILE`, you could omit the `--log-output` option.)

Log control at runtime. The system variables associated with log tables and files enable runtime control over logging:

- The global `log_output` system variable indicates the current logging destination. It can be modified at runtime to change the destination.
- The global `general_log` and `slow_query_log` variables indicate whether the general query log and slow query log are enabled (`ON`) or disabled (`OFF`). You can set these variables at runtime to control whether the logs are enabled.
- The global `general_log_file` and `slow_query_log_file` variables indicate the names of the general query log and slow query log files. You can set these variables at server startup or at runtime to change the names of the log files.
- To disable or enable general query logging for the current connection, set the session `sql_log_off` variable to `ON` or `OFF`.

The use of tables for log output offers the following benefits:

- Log entries have a standard format. To display the current structure of the log tables, use these statements:

  ```
  SHOW CREATE TABLE mysql.general_log;
  SHOW CREATE TABLE mysql.slow_log;
  ```

- Log contents are accessible through SQL statements. This enables the use of queries that select only those log entries that satisfy specific criteria. For example, to select log contents associated with a particular client (which can be useful for identifying problematic queries from that client), it is easier to do this using a log table than a log file.

- Logs are accessible remotely through any client that can connect to the server and issue queries (if the client has the appropriate log table privileges). It is not necessary to log in to the server host and directly access the file system.

The log table implementation has the following characteristics:

- In general, the primary purpose of log tables is to provide an interface for users to observe the runtime execution of the server, not to interfere with its runtime execution.
- `CREATE TABLE`, `ALTER TABLE`, and `DROP TABLE` are valid operations on a log table. For `ALTER TABLE` and `DROP TABLE`, the log table cannot be in use and must be disabled, as described later.
- By default, the log tables use the `CSV` storage engine that writes data in comma-separated values format. For users who have access to the `.CSV` files that contain log table data, the files are easy to import into other programs such as spreadsheets that can process CSV input.

  The log tables can be altered to use the `MyISAM` storage engine. You cannot use `ALTER TABLE` to alter a log table that is in use. The log must be disabled first. No engines other than `CSV` or `MyISAM` are legal for the log tables.
- To disable logging so that you can alter (or drop) a log table, you can use the following strategy. The example uses the general query log; the procedure for the slow query log is similar but uses the `slow_log` table and `slow_query_log` system variable.

```
SET @old_log_state = @@global.general_log;
SET GLOBAL general_log = 'OFF';
ALTER TABLE mysql.general_log ENGINE = MyISAM;
SET GLOBAL general_log = @old_log_state;
```

- `TRUNCATE TABLE` is a valid operation on a log table. It can be used to expire log entries.
- `RENAME TABLE` is a valid operation on a log table. You can atomically rename a log table (to perform log rotation, for example) using the following strategy:

```
USE mysql;
DROP TABLE IF EXISTS general_log2;
CREATE TABLE general_log2 LIKE general_log;
RENAME TABLE general_log TO general_log_backup, general_log2 TO general_log;
```

- `CHECK TABLE` is a valid operation on a log table.
- `LOCK TABLES` cannot be used on a log table.
- `INSERT`, `DELETE`, and `UPDATE` cannot be used on a log table. These operations are permitted only internally to the server itself.
- `FLUSH TABLES WITH READ LOCK` and the state of the global `read_only` system variable have no effect on log tables. The server can always write to the log tables.
- Entries written to the log tables are not written to the binary log and thus are not replicated to slave servers.
- To flush the log tables or log files, use `FLUSH TABLES` or `FLUSH LOGS`, respectively.
- Partitioning of log tables is not permitted.

- A mysqldump dump includes statements to recreate those tables so that they are not missing after reloading the dump file. Log table contents are not dumped.

5.2.2 The Error Log

The error log contains information indicating when mysqld was started and stopped and also any critical errors that occur while the server is running. If mysqld notices a table that needs to be automatically checked or repaired, it writes a message to the error log.

On some operating systems, the error log contains a stack trace if mysqld dies. The trace can be used to determine where mysqld died. See Section 22.4, “Debugging and Porting MySQL”.

In the following discussion, “console” means stderr, the standard error output; this is your terminal or console window unless the standard error output has been redirected. (For example, if invoked with the --syslog option, mysqld_safe arranges for the server's stderr to be sent to the syslog facility, as described later.)

On Windows, the --log-error and --console options both affect error logging:

- Without --log-error, mysqld writes error messages to *host_name*.err in the data directory.

- With --log-error[=*file_name*], mysqld writes error messages to an error log file. The server uses the named file if present, creating it in in the data directory unless an absolute path name is given to specify a different directory. If no file is named, the default name is *host_name*.err in the data directory.

- If --console is given, mysqld writes error messages to the console. --log-error, if given, is ignored and has no effect. If both options are present, their order does not matter: --console takes precedence and error messages go to the console. (In MySQL 5.5 and 5.6, the precedence is reversed: --log-error causes --console to be ignored.)

In addition, on Windows, events and error messages are written to the Windows Event Log within the Application log. Entries marked as Warning and Note are written to the Event Log, but not informational messages such as information statements from individual storage engines. These log entries have a source of MySQL. You cannot disable writing information to the Windows Event Log.

On Unix and Unix-like systems, mysqld writes error log messages as follows:

- Without --log-error, mysqld writes error messages to the console.

- With --log-error[=file_name], mysqld writes error messages to an error log file. The server uses the named file if present, creating it in the data directory unless an absolute path name is given to specify a different directory. If no file is named, the default name is *host_name*.err in the data directory.

At runtime, if the server writes error messages to the console, it sets the log_error system variable to stderr. Otherwise, log_error indicates the error log file name. In particular, on Windows, --console overrides use of an error log file and sends error messages to the console, so log_error is set to stderr. This occurs even if --log-error is also given.

If you flush the logs using FLUSH LOGS or mysqladmin flush-logs and mysqld is writing the error log to a file (for example, if it was started with the --log-error option), the server closes and reopens the log file. To rename the file, do so manually before flushing. Then flushing the logs reopens a new file with the original file name. For example, you can rename the file and create a new one using the following commands:

```
shell> mv host_name.err host_name.err-old
shell> mysqladmin flush-logs
shell> mv host_name.err-old backup-directory
```

On Windows, use rename rather than mv.

No error log renaming occurs when the logs are flushed if the server is not writing to a named file.

If you use mysqld_safe to start mysqld, mysqld_safe arranges for mysqld to write error messages to a log file or to syslog. mysqld_safe has three error-logging options, --syslog, --skip-syslog, and --log-error. The default with no logging options or with --skip-syslog is to use the default log file. To explicitly specify use of an error log file, specify --log-error=*file_name* to mysqld_safe, and mysqld_safe will arrange for mysqld to write messages to a log file. To use syslog instead, specify the --syslog option.

If you specify --log-error in an option file in a [mysqld], [server], or [mysqld_safe] section, mysqld_safe will find and use the option.

If mysqld_safe is used to start mysqld and mysqld dies unexpectedly, mysqld_safe notices that it needs to restart mysqld and writes a restarted mysqld message to the error log.

As of MySQL 5.7.2, the log_error_verbosity system variable controls verbosity of the server in writing error, warning, and note messages to the error log. Permitted values are 1 (errors only), 2 (errors and warnings), 3 (errors, warnings, and notes), with a default of 3. If the value is greater than 2, aborted connections are written to the error log, and access-denied errors for new connection attempts are written. See Section C.5.2.11, “Communication Errors and Aborted Connections”.

Before MySQL 5.7.2, the log_warnings system variable can be used to control warning logging to the error log. The default value is enabled (1). Warning logging can be disabled using a value of 0.

As of MySQL 5.7.2, the log_timestamps system variable controls the timestamp time zone of messages written to the error log (as well as to general query log and slow query log files). Permitted values are UTC (the default) and SYSTEM (local system time zone). Before MySQL 5.7.2, messages use the local system time zone.

As of MySQL 5.7.2, the ID included in error log messages is that of the thread within mysqld responsible for writing the message. This indicates which part of the server produced the message, and is consistent with general query log and slow query log messages, which include the connection thread ID. Before MySQL 5.7.2, the ID in error log messages is that of the mysqld process ID.

5.2.3 The General Query Log

The general query log is a general record of what mysqld is doing. The server writes information to this log when clients connect or disconnect, and it logs each SQL statement received from clients. The general query log can be very useful when you suspect an error in a client and want to know exactly what the client sent to mysqld.

mysqld writes statements to the query log in the order that it receives them, which might differ from the order in which they are executed. This logging order is in contrast with that of the binary log, for which statements are written after they are executed but before any locks are released. In addition, the query log may contain statements that only select data while such statements are never written to the binary log.

When using statement-based logging all statements are written to the query log, but when using row-based logging, updates are sent as row changes rather than SQL statements, and thus these statements are never written to the query log when binlog_format is ROW. A given update also might not be written to the query log when this variable is set to MIXED, depending on the statement used. See Section 16.1.2.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”, for more information.

By default, the general query log is disabled. To specify the initial general query log state explicitly, use --general_log[={0|1}]. With no argument or an argument of 1, --general_log enables the log. With an argument of 0, this option disables the log. To specify a log file name, use --general_log_file=*file_name*. To specify the log destination, use --log-output (as described in Section 5.2.1, “Selecting General Query and Slow Query Log Output Destinations”).

If you specify no name for the general query log file, the default name is *host_name*.log. The server creates the file in the data directory unless an absolute path name is given to specify a different directory.

To disable or enable the general query log or change the log file name at runtime, use the global general_log and general_log_file system variables. Set general_log to 0 (or OFF) to disable the log or to 1 (or ON) to enable it. Set general_log_file to specify the name of the log file. If a log file already is open, it is closed and the new file is opened.

When the general query log is enabled, the server writes output to any destinations specified by the --log-output option or log_output system variable. If you enable the log, the server opens the log file and writes startup messages to it. However, further logging of queries to the file does not occur unless the FILE log destination is selected. If the destination is NONE, the server writes no queries even if the general log is enabled. Setting the log file name has no effect on logging if the log destination value does not contain FILE.

Server restarts and log flushing do not cause a new general query log file to be generated (although flushing closes and reopens it). To rename the file and create a new one, use the following commands:

```
shell> mv host_name.log host_name-old.log
shell> mysqladmin flush-logs
shell> mv host_name-old.log backup-directory
```

On Windows, use rename rather than mv.

You can also rename the general query log file at runtime by disabling the log:

```
SET GLOBAL general_log = 'OFF';
```

With the log disabled, rename the log file externally; for example, from the command line. Then enable the log again:

```
SET GLOBAL general_log = 'ON';
```

This method works on any platform and does not require a server restart.

The session sql_log_off variable can be set to ON or OFF to disable or enable general query logging for the current connection.

Passwords in statements written to the general query log are rewritten by the server not to occur literally in plain text. Password rewriting can be suppressed for the general query log by starting the server with the --log-raw option. This option may be useful for diagnostic purposes, to see the exact text of statements as received by the server, but for security reasons is not recommended for production use. See also Section 6.1.2.3, “Passwords and Logging”.

An implication of password rewriting is that statements that cannot be parsed (due, for example, to syntax errors) are not written to the general query log because they cannot be known to be password free. Use cases that require logging of all statements including those with errors should use the `--log-raw` option, bearing in mind that this also bypasses password writing.

As of MySQL 5.7.2, the `log_timestamps` system variable controls the timestamp time zone of messages written to the general query log file (as well as to the slow query log file and the error log). It does not affect the time zone of general query log and slow query log messages written to log tables, but rows retrieved from those tables can be converted from the local system time zone to any desired time zone with `CONVERT_TZ()` or by setting the session `time_zone` system variable. Before MySQL 5.7.2, messages use the local system time zone.

5.2.4 The Binary Log

The binary log contains “events” that describe database changes such as table creation operations or changes to table data. It also contains events for statements that potentially could have made changes (for example, a `DELETE` which matched no rows), unless row-based logging is used. The binary log also contains information about how long each statement took that updated data. The binary log has two important purposes:

- For replication, the binary log on a master replication server provides a record of the data changes to be sent to slave servers. The master server sends the events contained in its binary log to its slaves, which execute those events to make the same data changes that were made on the master. See Section 16.2, “Replication Implementation”.

- Certain data recovery operations require use of the binary log. After a backup has been restored, the events in the binary log that were recorded after the backup was made are re-executed. These events bring databases up to date from the point of the backup. See Section 7.5, “Point-in-Time (Incremental) Recovery Using the Binary Log”.

The binary log is not used for statements such as `SELECT` or `SHOW` that do not modify data. To log all statements (for example, to identify a problem query), use the general query log. See Section 5.2.3, “The General Query Log”.

Running a server with binary logging enabled makes performance slightly slower. However, the benefits of the binary log in enabling you to set up replication and for restore operations generally outweigh this minor performance decrement.

The binary log is crash-safe. Only complete events or transactions are logged or read back.

Passwords in statements written to the binary log are rewritten by the server not to occur literally in plain text. See also Section 6.1.2.3, “Passwords and Logging”.

The following discussion describes some of the server options and variables that affect the operation of binary logging. For a complete list, see Section 16.1.4.4, “Binary Log Options and Variables”.

To enable the binary log, start the server with the `--log-bin[=`*`base_name`*`]` option. If no *`base_name`* value is given, the default name is the value of the `pid-file` option (which by default is the name of host machine) followed by `-bin`. If the basename is given, the server writes the file in the data directory unless the basename is given with a leading absolute path name to specify a different directory. It is recommended that you specify a basename explicitly rather than using the default of the host name; see Section C.5.8, “Known Issues in MySQL”, for the reason.

If you supply an extension in the log name (for example, `--log-bin=`*`base_name.extension`*), the extension is silently removed and ignored.

`mysqld` appends a numeric extension to the binary log basename to generate binary log file names. The number increases each time the server creates a new log file, thus creating an ordered series of files. The server creates a new file in the series each time it starts or flushes the logs. The server also creates a new binary log file automatically after the current log's size reaches max_binlog_size. A binary log file may become larger than max_binlog_size if you are using large transactions because a transaction is written to the file in one piece, never split between files.

To keep track of which binary log files have been used, mysqld also creates a binary log index file that contains the names of all used binary log files. By default, this has the same basename as the binary log file, with the extension '.index'. You can change the name of the binary log index file with the --log-bin-index[=*file_name*] option. You should not manually edit this file while mysqld is running; doing so would confuse mysqld.

The term “binary log file” generally denotes an individual numbered file containing database events. The term “binary log” collectively denotes the set of numbered binary log files plus the index file.

A client that has the SUPER privilege can disable binary logging of its own statements by using a SET sql_log_bin=0 statement. See Section 5.1.4, “Server System Variables”.

By default, the server logs the length of the event as well as the event itself and uses this to verify that the event was written correctly. You can also cause the server to write checksums for the events by setting the binlog_checksum system variable. When reading back from the binary log, the master uses the event length by default, but can be made to use checksums if available by enabling the master_verify_checksum system variable. The slave I/O thread also verifies events received from the master. You can cause the slave SQL thread to use checksums if available when reading from the relay log by enabling the slave_sql_verify_checksum system variable.

The format of the events recorded in the binary log is dependent on the binary logging format. Three format types are supported, row-based logging, statement-based logging and mixed-base logging. The binary logging format used depends on the MySQL version. For general descriptions of the logging formats, see Section 5.2.4.1, “Binary Logging Formats”. For detailed information about the format of the binary log, see MySQL Internals: The Binary Log.

The server evaluates the --binlog-do-db and --binlog-ignore-db options in the same way as it does the --replicate-do-db and --replicate-ignore-db options. For information about how this is done, see Section 16.2.3.1, “Evaluation of Database-Level Replication and Binary Logging Options”.

A replication slave server by default does not write to its own binary log any data modifications that are received from the replication master. To log these modifications, start the slave with the --log-slave-updates option in addition to the --log-bin option (see Section 16.1.4.3, “Replication Slave Options and Variables”). This is done when a slave is also to act as a master to other slaves in chained replication.

You can delete all binary log files with the RESET MASTER statement, or a subset of them with PURGE BINARY LOGS. See Section 13.7.6.6, “RESET Syntax”, and Section 13.4.1.1, “PURGE BINARY LOGS Syntax”.

If you are using replication, you should not delete old binary log files on the master until you are sure that no slave still needs to use them. For example, if your slaves never run more than three days behind, once a day you can execute mysqladmin flush-logs on the master and then remove any logs that are more than three days old. You can remove the files manually, but it is preferable to use PURGE BINARY LOGS, which also safely updates the binary log index file for you (and which can take a date argument). See Section 13.4.1.1, “PURGE BINARY LOGS Syntax”.

You can display the contents of binary log files with the mysqlbinlog utility. This can be useful when you want to reprocess statements in the log for a recovery operation. For example, you can update a MySQL server from the binary log as follows:

```
shell> mysqlbinlog log_file | mysql -h server_name
```

mysqlbinlog also can be used to display replication slave relay log file contents because they are written using the same format as binary log files. For more information on the mysqlbinlog utility and how to use it, see Section 4.6.7, “mysqlbinlog — Utility for Processing Binary Log Files”. For more information about the binary log and recovery operations, see Section 7.5, “Point-in-Time (Incremental) Recovery Using the Binary Log”.

Binary logging is done immediately after a statement or transaction completes but before any locks are released or any commit is done. This ensures that the log is logged in commit order.

Updates to nontransactional tables are stored in the binary log immediately after execution.

Within an uncommitted transaction, all updates (UPDATE, DELETE, or INSERT) that change transactional tables such as InnoDB tables are cached until a COMMIT statement is received by the server. At that point, mysqld writes the entire transaction to the binary log before the COMMIT is executed.

Modifications to nontransactional tables cannot be rolled back. If a transaction that is rolled back includes modifications to nontransactional tables, the entire transaction is logged with a ROLLBACK statement at the end to ensure that the modifications to those tables are replicated.

When a thread that handles the transaction starts, it allocates a buffer of binlog_cache_size to buffer statements. If a statement is bigger than this, the thread opens a temporary file to store the transaction. The temporary file is deleted when the thread ends.

The Binlog_cache_use status variable shows the number of transactions that used this buffer (and possibly a temporary file) for storing statements. The Binlog_cache_disk_use status variable shows how many of those transactions actually had to use a temporary file. These two variables can be used for tuning binlog_cache_size to a large enough value that avoids the use of temporary files.

The max_binlog_cache_size system variable (default 4GB, which is also the maximum) can be used to restrict the total size used to cache a multiple-statement transaction. If a transaction is larger than this many bytes, it fails and rolls back. The minimum value is 4096.

If you are using the binary log and row based logging, concurrent inserts are converted to normal inserts for CREATE ... SELECT or INSERT ... SELECT statements. This is done to ensure that you can re-create an exact copy of your tables by applying the log during a backup operation. If you are using statement-based logging, the original statement is written to the log.

The binary log format has some known limitations that can affect recovery from backups. See Section 16.4.1, “Replication Features and Issues”.

Binary logging for stored programs is done as described in Section 18.7, “Binary Logging of Stored Programs”.

Note that the binary log format differs in MySQL 5.7 from previous versions of MySQL, due to enhancements in replication. See Section 16.4.2, “Replication Compatibility Between MySQL Versions”.

Writes to the binary log file and binary log index file are handled in the same way as writes to MyISAM tables. See Section C.5.4.3, “How MySQL Handles a Full Disk”.

By default, the binary log is not synchronized to disk at each write. So if the operating system or machine (not only the MySQL server) crashes, there is a chance that the last statements of the binary log are lost. To prevent this, you can make the binary log be synchronized to disk after every *N* writes to the binary log, with the sync_binlog system variable. See Section 5.1.4, “Server System Variables”. 1 is the safest value for sync_binlog, but also the slowest. Even with sync_binlog set to 1, there is still the chance of an inconsistency between the table content and binary log content in case of a crash. For example, if you are using InnoDB tables and the MySQL server processes a COMMIT statement, it writes the whole transaction to the binary log and then commits this transaction into InnoDB. If the server crashes between those two operations, the transaction is rolled back by InnoDB at restart but still exists in the binary log. To resolve this, you should set --innodb_support_xa to 1. Although this option is related to the support of XA transactions in InnoDB, it also ensures that the binary log and InnoDB data files are synchronized.

For this option to provide a greater degree of safety, the MySQL server should also be configured to synchronize the binary log and the InnoDB logs to disk before committing the transaction. The InnoDB logs are synchronized by default, and sync_binlog=1 can be used to synchronize the binary log. The effect of this option is that at restart after a crash, after doing a rollback of transactions, the MySQL server cuts rolled back InnoDB transactions from the binary log. This ensures that the binary log reflects the exact data of InnoDB tables, and so, that the slave remains in synchrony with the master (not receiving a statement which has been rolled back).

If the MySQL server discovers at crash recovery that the binary log is shorter than it should have been, it lacks at least one successfully committed InnoDB transaction. This should not happen if sync_binlog=1 and the disk/file system do an actual sync when they are requested to (some do not), so the server prints an error message The binary log *file_name* is shorter than its expected size. In this case, this binary log is not correct and replication should be restarted from a fresh snapshot of the master's data.

The session values of the following system variables are written to the binary log and honored by the replication slave when parsing the binary log:

- sql_mode (except that the NO_DIR_IN_CREATE mode is not replicated; see Section 16.4.1.34, “Replication and Variables”)
- foreign_key_checks
- unique_checks
- character_set_client
- collation_connection
- collation_database
- collation_server
- sql_auto_is_null

5.2.4.1 Binary Logging Formats

The server uses several logging formats to record information in the binary log. The exact format employed depends on the version of MySQL being used. There are three logging formats:

- Replication capabilities in MySQL originally were based on propagation of SQL statements from master to slave. This is called *statement-based logging*. You can cause this format to be used by starting the server with --binlog-format=STATEMENT.
- In *row-based logging*, the master writes events to the binary log that indicate how individual table rows are affected. You can cause the server to use row-based logging by starting it with --binlog-format=ROW.
- A third option is also available: *mixed logging*. With mixed logging, statement-based logging is used by default, but the logging mode switches automatically to row-based in certain cases as described below. You can cause MySQL to use mixed logging explicitly by starting mysqld with the option --binlog-format=MIXED.

In MySQL 5.7, the default binary logging format is STATEMENT.

The logging format can also be set or limited by the storage engine being used. This helps to eliminate issues when replicating certain statements between a master and slave which are using different storage engines.

With statement-based replication, there may be issues with replicating nondeterministic statements. In deciding whether or not a given statement is safe for statement-based replication, MySQL determines whether it can guarantee that the statement can be replicated using statement-based logging. If MySQL cannot make this guarantee, it marks the statement as potentially unreliable and issues the warning, `Statement may not be safe to log in statement format`.

You can avoid these issues by using MySQL's row-based replication instead.

5.2.4.2 Setting The Binary Log Format

You can select the binary logging format explicitly by starting the MySQL server with `--binlog-format=type`. The supported values for *`type`* are:

- `STATEMENT` causes logging to be statement based.
- `ROW` causes logging to be row based.
- `MIXED` causes logging to use mixed format.

In MySQL 5.7, the default binary logging format is `STATEMENT`.

The logging format also can be switched at runtime. To specify the format globally for all clients, set the global value of the `binlog_format` system variable:

```
mysql> SET GLOBAL binlog_format = 'STATEMENT';
mysql> SET GLOBAL binlog_format = 'ROW';
mysql> SET GLOBAL binlog_format = 'MIXED';
```

An individual client can control the logging format for its own statements by setting the session value of `binlog_format`:

```
mysql> SET SESSION binlog_format = 'STATEMENT';
mysql> SET SESSION binlog_format = 'ROW';
mysql> SET SESSION binlog_format = 'MIXED';
```

Note

Each MySQL Server can set its own and only its own binary logging format (true whether `binlog_format` is set with global or session scope). This means that changing the logging format on a replication master does not cause a slave to change its logging format to match. (When using `STATEMENT` mode, the `binlog_format` system variable is not replicated; when using `MIXED` or `ROW` logging mode, it is replicated but is ignored by the slave.) Changing the binary logging format on the master while replication is ongoing, or without also changing it on the slave can thus cause unexpected results, or even cause replication to fail altogether.

To change the global or session `binlog_format` value, you must have the `SUPER` privilege.

In addition to switching the logging format manually, a slave server may switch the format *automatically*. This happens when the server is running in either `STATEMENT` or `MIXED` format and encounters an event in the binary log that is written in `ROW` logging format. In that case, the slave switches to row-based replication temporarily for that event, and switches back to the previous format afterward.

There are several reasons why a client might want to set binary logging on a per-session basis:

- A session that makes many small changes to the database might want to use row-based logging.
- A session that performs updates that match many rows in the `WHERE` clause might want to use statement-based logging because it will be more efficient to log a few statements than many rows.
- Some statements require a lot of execution time on the master, but result in just a few rows being modified. It might therefore be beneficial to replicate them using row-based logging.

There are exceptions when you cannot switch the replication format at runtime:

- From within a stored function or a trigger
- If the session is currently in row-based replication mode and has open temporary tables

Trying to switch the format in any of these cases results in an error.

If you are using `InnoDB` tables and the transaction isolation level is `READ COMMITTED` or `READ UNCOMMITTED`, only row-based logging can be used. It is *possible* to change the logging format to `STATEMENT`, but doing so at runtime leads very rapidly to errors because `InnoDB` can no longer perform inserts.

Switching the replication format at runtime is not recommended when any temporary tables exist, because temporary tables are logged only when using statement-based replication, whereas with row-based replication they are not logged. With mixed replication, temporary tables are usually logged; exceptions happen with user-defined functions (UDFs) and with the `UUID()` function.

With the binary log format set to `ROW`, many changes are written to the binary log using the row-based format. Some changes, however, still use the statement-based format. Examples include all DDL (data definition language) statements such as `CREATE TABLE`, `ALTER TABLE`, or `DROP TABLE`.

The `--binlog-row-event-max-size` option is available for servers that are capable of row-based replication. Rows are stored into the binary log in chunks having a size in bytes not exceeding the value of this option. The value must be a multiple of 256. The default value is 1024.

> **Warning**
>
> When using *statement-based logging* for replication, it is possible for the data on the master and slave to become different if a statement is designed in such a way that the data modification is *nondeterministic*; that is, it is left to the will of the query optimizer. In general, this is not a good practice even outside of replication. For a detailed explanation of this issue, see Section C.5.8, “Known Issues in MySQL”.

For information about logs kept by replication slaves, see Section 16.2.2, “Replication Relay and Status Logs”.

5.2.4.3 Mixed Binary Logging Format

When running in `MIXED` logging format, the server automatically switches from statement-based to row-based logging under the following conditions:

- When a function contains `UUID()`.
- When one or more tables with `AUTO_INCREMENT` columns are updated and a trigger or stored function is invoked. Like all other unsafe statements, this generates a warning if `binlog_format = STATEMENT`.

For more information, see Section 16.4.1.1, “Replication and AUTO_INCREMENT”.

- When the body of a view requires row-based replication, the statement creating the view also uses it. For example, this occurs when the statement creating a view uses the UUID() function.
- When a call to a UDF is involved.
- If a statement is logged by row and the session that executed the statement has any temporary tables, logging by row is used for all subsequent statements (except for those accessing temporary tables) until all temporary tables in use by that session are dropped.

  This is true whether or not any temporary tables are actually logged.

  Temporary tables cannot be logged using row-based format; thus, once row-based logging is used, all subsequent statements using that table are unsafe. The server approximates this condition by treating all statements executed during the session as unsafe until the session no longer holds any temporary tables.
- When FOUND_ROWS() or ROW_COUNT() is used. (Bug #12092, Bug #30244)
- When USER(), CURRENT_USER(), or CURRENT_USER is used. (Bug #28086)
- When a statement refers to one or more system variables. (Bug #31168)

  Exception. The following system variables, when used with session scope (only), do not cause the logging format to switch:

  - auto_increment_increment
  - auto_increment_offset
  - character_set_client
  - character_set_connection
  - character_set_database
  - character_set_server
  - collation_connection
  - collation_database
  - collation_server
  - foreign_key_checks
  - identity
  - last_insert_id
  - lc_time_names
  - pseudo_thread_id
  - sql_auto_is_null
  - time_zone

  - `timestamp`
  - `unique_checks`

  For information about determining system variable scope, see Section 5.1.5, “Using System Variables”.

  For information about how replication treats `sql_mode`, see Section 16.4.1.34, “Replication and Variables”.

- When one of the tables involved is a log table in the `mysql` database.
- When the `LOAD_FILE()` function is used. (Bug #39701)

Note

A warning is generated if you try to execute a statement using statement-based logging that should be written using row-based logging. The warning is shown both in the client (in the output of `SHOW WARNINGS`) and through the `mysqld` error log. A warning is added to the `SHOW WARNINGS` table each time such a statement is executed. However, only the first statement that generated the warning for each client session is written to the error log to prevent flooding the log.

In addition to the decisions above, individual engines can also determine the logging format used when information in a table is updated. The logging capabilities of an individual engine can be defined as follows:

- If an engine supports row-based logging, the engine is said to be *row-logging capable*.
- If an engine supports statement-based logging, the engine is said to be *statement-logging capable*.

A given storage engine can support either or both logging formats. The following table lists the formats supported by each engine.

Storage Engine	Row Logging Supported	Statement Logging Supported
`ARCHIVE`	Yes	Yes
`BLACKHOLE`	Yes	Yes
`CSV`	Yes	Yes
`EXAMPLE`	Yes	No
`FEDERATED`	Yes	Yes
`HEAP`	Yes	Yes
`InnoDB`	Yes	Yes when the transaction isolation level is `REPEATABLE READ` or `SERIALIZABLE`; No otherwise.
`MyISAM`	Yes	Yes
`MERGE`	Yes	Yes
`NDBCLUSTER`	Yes	No

In MySQL 5.7, whether a statement is to be logged and the logging mode to be used is determined according to the type of statement (safe, unsafe, or binary injected), the binary logging format (`STATEMENT`, `ROW`, or `MIXED`), and the logging capabilities of the storage engine (statement capable, row capable, both, or neither). (Binary injection refers to logging a change that must be logged using ROW format.)

Statements may be logged with or without a warning; failed statements are not logged, but generate errors in the log. This is shown in the following decision table, where **SLC** stands for “statement-logging capable” and **RLC** stands for “row-logging capable”.

Condition				**Action**	
Type	**binlog_format**	**SLC**	**RLC**	**Error / Warning**	**Logged as**
*	*	No	No	Error: Cannot execute statement: Binary logging is impossible since at least one engine is involved that is both row-incapable and statement-incapable.	-
Safe	STATEMENT	Yes	No	-	STATEMENT
Safe	MIXED	Yes	No	-	STATEMENT
Safe	ROW	Yes	No	Error: Cannot execute statement: Binary logging is impossible since BINLOG_FORMAT = ROW and at least one table uses a storage engine that is not capable of row-based logging.	-
Unsafe	STATEMENT	Yes	No	Warning: Unsafe statement binlogged in statement format, since BINLOG_FORMAT = STATEMENT	STATEMENT
Unsafe	MIXED	Yes	No	Error: Cannot execute statement: Binary logging of an unsafe statement is impossible when the storage engine is limited to statement-based logging, even if BINLOG_FORMAT = MIXED.	-

Condition				**Action**	
Type	**binlog_format**	**SLC**	**RLC**	**Error / Warning**	**Logged as**
Unsafe	ROW	Yes	No	Error: Cannot execute statement: Binary logging is impossible since BINLOG_FORMAT = ROW and at least one table uses a storage engine that is not capable of row-based logging.	-
Row Injection	STATEMENT	Yes	No	Error: Cannot execute row injection: Binary logging is not possible since at least one table uses a storage engine that is not capable of row-based logging.	-
Row Injection	MIXED	Yes	No	Error: Cannot execute row injection: Binary logging is not possible since at least one table uses a storage engine that is not capable of row-based logging.	-
Row Injection	ROW	Yes	No	Error: Cannot execute row injection: Binary logging is not possible since at least one table uses a storage engine that is not capable of row-based logging.	-
Safe	STATEMENT	No	Yes	Error: Cannot execute statement: Binary logging is impossible since BINLOG_FORMAT = STATEMENT and at least one table uses	-

Condition				Action	
Type	**binlog_format**	**SLC**	**RLC**	**Error / Warning**	**Logged as**
				a storage engine that is not capable of statement-based logging.	
Safe	MIXED	No	Yes	-	ROW
Safe	ROW	No	Yes	-	ROW
Unsafe	STATEMENT	No	Yes	Error: Cannot execute statement: Binary logging is impossible since BINLOG_FORMAT = STATEMENT and at least one table uses a storage engine that is not capable of statement-based logging.	-
Unsafe	MIXED	No	Yes	-	ROW
Unsafe	ROW	No	Yes	-	ROW
Row Injection	STATEMENT	No	Yes	Error: Cannot execute row injection: Binary logging is not possible since BINLOG_FORMAT = STATEMENT.	-
Row Injection	MIXED	No	Yes	-	ROW
Row Injection	ROW	No	Yes	-	ROW
Safe	STATEMENT	Yes	Yes	-	STATEMENT
Safe	MIXED	Yes	Yes	-	STATEMENT
Safe	ROW	Yes	Yes	-	ROW
Unsafe	STATEMENT	Yes	Yes	Warning: Unsafe statement binlogged in statement format since BINLOG_FORMAT = STATEMENT.	STATEMENT
Unsafe	MIXED	Yes	Yes	-	ROW
Unsafe	ROW	Yes	Yes	-	ROW
Row Injection	STATEMENT	Yes	Yes	Error: Cannot execute row	-

Condition				Action	
Type	**binlog_format**	**SLC**	**RLC**	**Error / Warning**	**Logged as**
				injection: Binary logging is not possible because BINLOG_FORMAT = STATEMENT.	
Row Injection	MIXED	Yes	Yes	-	ROW
Row Injection	ROW	Yes	Yes	-	ROW

When a warning is produced by the determination, a standard MySQL warning is produced (and is available using SHOW WARNINGS). The information is also written to the mysqld error log. Only one error for each error instance per client connection is logged to prevent flooding the log. The log message includes the SQL statement that was attempted.

If a slave server was started with log_error_verbosity set to display warnings, the slave prints messages to the error log to provide information about its status, such as the binary log and relay log coordinates where it starts its job, when it is switching to another relay log, when it reconnects after a disconnect, statements that are unsafe for statement-based logging, and so forth.

5.2.4.4 Logging Format for Changes to mysql Database Tables

The contents of the grant tables in the mysql database can be modified directly (for example, with INSERT or DELETE) or indirectly (for example, with GRANT or CREATE USER). Statements that affect mysql database tables are written to the binary log using the following rules:

- Data manipulation statements that change data in mysql database tables directly are logged according to the setting of the binlog_format system variable. This pertains to statements such as INSERT, UPDATE, DELETE, REPLACE, DO, LOAD DATA INFILE, SELECT, and TRUNCATE TABLE.
- Statements that change the mysql database indirectly are logged as statements regardless of the value of binlog_format. This pertains to statements such as GRANT, REVOKE, SET PASSWORD, RENAME USER, CREATE (all forms except CREATE TABLE ... SELECT), ALTER (all forms), and DROP (all forms).

CREATE TABLE ... SELECT is a combination of data definition and data manipulation. The CREATE TABLE part is logged using statement format and the SELECT part is logged according to the value of binlog_format.

5.2.5 The Slow Query Log

The slow query log consists of SQL statements that took more than long_query_time seconds to execute and required at least min_examined_row_limit rows to be examined. The minimum and default values of long_query_time are 0 and 10, respectively. The value can be specified to a resolution of microseconds. For logging to a file, times are written including the microseconds part. For logging to tables, only integer times are written; the microseconds part is ignored.

By default, administrative statements are not logged, nor are queries that do not use indexes for lookups. This behavior can be changed using log_slow_admin_statements and log_queries_not_using_indexes, as described later.

The time to acquire the initial locks is not counted as execution time. mysqld writes a statement to the slow query log after it has been executed and after all locks have been released, so log order might differ from execution order.

By default, the slow query log is disabled. To specify the initial slow query log state explicitly, use --slow_query_log[={0|1}]. With no argument or an argument of 1, --slow_query_log enables the log. With an argument of 0, this option disables the log. To specify a log file name, use --slow_query_log_file=*file_name*. To specify the log destination, use --log-output (as described in Section 5.2.1, "Selecting General Query and Slow Query Log Output Destinations").

If you specify no name for the slow query log file, the default name is *host_name*-slow.log. The server creates the file in the data directory unless an absolute path name is given to specify a different directory.

To disable or enable the slow query log or change the log file name at runtime, use the global slow_query_log and slow_query_log_file system variables. Set slow_query_log to 0 (or OFF) to disable the log or to 1 (or ON) to enable it. Set slow_query_log_file to specify the name of the log file. If a log file already is open, it is closed and the new file is opened.

When the slow query log is enabled, the server writes output to any destinations specified by the --log-output option or log_output system variable. If you enable the log, the server opens the log file and writes startup messages to it. However, further logging of queries to the file does not occur unless the FILE log destination is selected. If the destination is NONE, the server writes no queries even if the slow query log is enabled. Setting the log file name has no effect on logging if the log destination value does not contain FILE.

The server writes less information to the slow query log (and binary log) if you use the --log-short-format option.

To include slow administrative statements in the statements written to the slow query log, use the log_slow_admin_statements system variable. Administrative statements include ALTER TABLE, ANALYZE TABLE, CHECK TABLE, CREATE INDEX, DROP INDEX, OPTIMIZE TABLE, and REPAIR TABLE.

To include queries that do not use indexes for row lookups in the statements written to the slow query log, enable the log_queries_not_using_indexes system variable. When such queries are logged, the slow query log may grow quickly. It is possible to put a rate limit on these queries by setting the log_throttle_queries_not_using_indexes system variable. By default, this variable is 0, which means there is no limit. Positive values impose a per-minute limit on logging of queries that do not use indexes. The first such query opens a 60-second window within which the server logs queries up to the given limit, then suppresses additional queries. If there are suppressed queries when the window ends, the server logs a summary that indicates how many there were and the aggregate time spent in them. The next 60-second window begins when the server logs the next query that does not use indexes.

The server uses the controlling parameters in the following order to determine whether to write a query to the slow query log:

1. The query must either not be an administrative statement, or log_slow_admin_statements must be enabled.

2. The query must have taken at least long_query_time seconds, or log_queries_not_using_indexes must be enabled and the query used no indexes for row lookups.

3. The query must have examined at least min_examined_row_limit rows.

4. The query must not be suppressed according to the log_throttle_queries_not_using_indexes setting.

As of MySQL 5.7.2, the log_timestamps system variable controls the timestamp time zone of messages written to the slow query log file (as well as to the general query log file and the error log). It does not affect the time zone of general query log and slow query log messages written to log tables, but rows retrieved from those tables can be converted from the local system time zone to any desired time zone with CONVERT_TZ() or by setting the session time_zone system variable. Before MySQL 5.7.2, messages use the local system time zone.

The server does not write queries handled by the query cache to the slow query log, nor queries that would not benefit from the presence of an index because the table has zero rows or one row.

By default, a replication slave does not write replicated queries to the slow query log. To change this, use the log_slow_slave_statements system variable.

Passwords in statements written to the slow query log are rewritten by the server not to occur literally in plain text. See also Section 6.1.2.3, "Passwords and Logging".

The slow query log can be used to find queries that take a long time to execute and are therefore candidates for optimization. However, examining a long slow query log can become a difficult task. To make this easier, you can process a slow query log file using the mysqldumpslow command to summarize the queries that appear in the log. See Section 4.6.8, "mysqldumpslow — Summarize Slow Query Log Files".

5.2.6 Server Log Maintenance

As described in Section 5.2, "MySQL Server Logs", MySQL Server can create several different log files to help you see what activity is taking place. However, you must clean up these files regularly to ensure that the logs do not take up too much disk space.

When using MySQL with logging enabled, you may want to back up and remove old log files from time to time and tell MySQL to start logging to new files. See Section 7.2, "Database Backup Methods".

On a Linux (Red Hat) installation, you can use the mysql-log-rotate script for this. If you installed MySQL from an RPM distribution, this script should have been installed automatically. Be careful with this script if you are using the binary log for replication. You should not remove binary logs until you are certain that their contents have been processed by all slaves.

On other systems, you must install a short script yourself that you start from cron (or its equivalent) for handling log files.

For the binary log, you can set the expire_logs_days system variable to expire binary log files automatically after a given number of days (see Section 5.1.4, "Server System Variables"). If you are using replication, you should set the variable no lower than the maximum number of days your slaves might lag behind the master. To remove binary logs on demand, use the PURGE BINARY LOGS statement (see Section 13.4.1.1, "PURGE BINARY LOGS Syntax").

You can force MySQL to start using new log files by flushing the logs. Log flushing occurs when you issue a FLUSH LOGS statement or execute a mysqladmin flush-logs, mysqladmin refresh, mysqldump --flush-logs, or mysqldump --master-data command. See Section 13.7.6.3, "FLUSH Syntax", Section 4.5.2, "mysqladmin — Client for Administering a MySQL Server", and Section 4.5.4, "mysqldump — A Database Backup Program". In addition, the binary log is flushed when its size reaches the value of the max_binlog_size system variable.

FLUSH LOGS supports optional modifiers to enable selective flushing of individual logs (for example, FLUSH BINARY LOGS).

A log-flushing operation does the following:

- If general query logging or slow query logging to a log file is enabled, the server closes and reopens the general query log file or slow query log file.
- If binary logging is enabled, the server closes the current binary log file and opens a new log file with the next sequence number.
- If the server was started with the `--log-error` option to cause the error log to be written to a file, the server closes and reopens the log file.

The server creates a new binary log file when you flush the logs. However, it just closes and reopens the general and slow query log files. To cause new files to be created on Unix, rename the current log files before flushing them. At flush time, the server opens new log files with the original names. For example, if the general and slow query log files are named `mysql.log` and `mysql-slow.log`, you can use a series of commands like this:

```
shell> cd mysql-data-directory
shell> mv mysql.log mysql.old
shell> mv mysql-slow.log mysql-slow.old
shell> mysqladmin flush-logs
```

On Windows, use `rename` rather than `mv`.

At this point, you can make a backup of `mysql.old` and `mysql-slow.old` and then remove them from disk.

A similar strategy can be used to back up the error log file, if there is one.

You can rename the general query log or slow query log at runtime by disabling the log:

```
SET GLOBAL general_log = 'OFF';
SET GLOBAL slow_query_log = 'OFF';
```

With the logs disabled, rename the log files externally; for example, from the command line. Then enable the logs again:

```
SET GLOBAL general_log = 'ON';
SET GLOBAL slow_query_log = 'ON';
```

This method works on any platform and does not require a server restart.

5.3 Running Multiple MySQL Instances on One Machine

In some cases, you might want to run multiple instances of MySQL on a single machine. You might want to test a new MySQL release while leaving an existing production setup undisturbed. Or you might want to give different users access to different `mysqld` servers that they manage themselves. (For example, you might be an Internet Service Provider that wants to provide independent MySQL installations for different customers.)

It is possible to use a different MySQL server binary per instance, or use the same binary for multiple instances, or any combination of the two approaches. For example, you might run a server from MySQL 5.6 and one from MySQL 5.7, to see how different versions handle a given workload. Or you might run multiple instances of the current production version, each managing a different set of databases.

Whether or not you use distinct server binaries, each instance that you run must be configured with unique values for several operating parameters. This eliminates the potential for conflict between instances. Parameters can be set on the command line, in option files, or by setting environment variables. See

Section 4.2.3, “Specifying Program Options”. To see the values used by a given instance, connect to it and execute a `SHOW VARIABLES` statement.

The primary resource managed by a MySQL instance is the data directory. Each instance should use a different data directory, the location of which is specified using the `--datadir=path` option. For methods of configuring each instance with its own data directory, and warnings about the dangers of failing to do so, see Section 5.3.1, “Setting Up Multiple Data Directories”.

In addition to using different data directories, several other options must have different values for each server instance:

- `--port=port_num`

  `--port` controls the port number for TCP/IP connections. Alternatively, if the host has multiple network addresses, you can use `--bind-address` to cause each server to listen to a different address.

- `--socket=path`

  `--socket` controls the Unix socket file path on Unix or the named pipe name on Windows. On Windows, it is necessary to specify distinct pipe names only for those servers configured to permit named-pipe connections.

- `--shared-memory-base-name=name`

  This option is used only on Windows. It designates the shared-memory name used by a Windows server to permit clients to connect using shared memory. It is necessary to specify distinct shared-memory names only for those servers configured to permit shared-memory connections.

- `--pid-file=file_name`

  This option indicates the path name of the file in which the server writes its process ID.

If you use the following log file options, their values must differ for each server:

- `--general_log_file=file_name`
- `--log-bin[=file_name]`
- `--slow_query_log_file=file_name`
- `--log-error[=file_name]`

For further discussion of log file options, see Section 5.2, “MySQL Server Logs”.

To achieve better performance, you can specify the following option differently for each server, to spread the load between several physical disks:

- `--tmpdir=path`

Having different temporary directories also makes it easier to determine which MySQL server created any given temporary file.

If you have multiple MySQL installations in different locations, you can specify the base directory for each installation with the `--basedir=path` option. This causes each instance to automatically use a different data directory, log files, and PID file because the default for each of those parameters is relative to the base directory. In that case, the only other options you need to specify are the `--socket` and `--port` options. Suppose that you install different versions of MySQL using `tar` file binary distributions. These install in different locations, so you can start the server for each installation using the command `bin/mysqld_safe` under its corresponding base directory. `mysqld_safe` determines the proper `--`

basedir option to pass to mysqld, and you need specify only the --socket and --port options to mysqld_safe.

As discussed in the following sections, it is possible to start additional servers by specifying appropriate command options or by setting environment variables. However, if you need to run multiple servers on a more permanent basis, it is more convenient to use option files to specify for each server those option values that must be unique to it. The --defaults-file option is useful for this purpose.

5.3.1 Setting Up Multiple Data Directories

Each MySQL Instance on a machine should have its own data directory. The location is specified using the --datadir=*path* option.

There are different methods of setting up a data directory for a new instance:

- Create a new data directory.
- Copy an existing data directory.

The following discussion provides more detail about each method.

> **Warning**
>
> Normally, you should never have two servers that update data in the same databases. This may lead to unpleasant surprises if your operating system does not support fault-free system locking. If (despite this warning) you run multiple servers using the same data directory and they have logging enabled, you must use the appropriate options to specify log file names that are unique to each server. Otherwise, the servers try to log to the same files.
>
> Even when the preceding precautions are observed, this kind of setup works only with MyISAM and MERGE tables, and not with any of the other storage engines. Also, this warning against sharing a data directory among servers always applies in an NFS environment. Permitting multiple MySQL servers to access a common data directory over NFS is a *very bad idea*. The primary problem is that NFS is the speed bottleneck. It is not meant for such use. Another risk with NFS is that you must devise a way to ensure that two or more servers do not interfere with each other. Usually NFS file locking is handled by the lockd daemon, but at the moment there is no platform that performs locking 100% reliably in every situation.

Create a New Data Directory

With this method, the data directory will be in the same state as when you first install MySQL. It will have the default set of MySQL accounts and no user data.

On Unix, initialize the data directory by running mysql_install_db. See Section 2.9.1, “Postinstallation Procedures for Unix-like Systems”.

On Windows, the data directory is included in the MySQL distribution:

- MySQL Zip archive distributions for Windows contain an unmodified data directory. You can unpack such a distribution into a temporary location, then copy it data directory to where you are setting up the new instance.
- Windows MSI package installers create and set up the data directory that the installed server will use, but also create a pristine “template” data directory named data under the installation directory. After an installation has been performed using an MSI package, the template data directory can be copied to set up additional MySQL instances.

Copy an Existing Data Directory

With this method, any MySQL accounts or user data present in the data directory are carried over to the new data directory.

1. Stop the existing MySQL instance using the data directory. This must be a clean shutdown so that the instance flushes any pending changes to disk.

2. Copy the data directory to the location where the new data directory should be.

3. Copy the `my.cnf` or `my.ini` option file used by the existing instance. This serves as a basis for the new instance.

4. Modify the new option file so that any pathnames referring to the original data directory refer to the new data directory. Also, modify any other options that must be unique per instance, such as the TCP/IP port number and the log files. For a list of parameters that must be unique per instance, see Section 5.3, “Running Multiple MySQL Instances on One Machine”.

5. Start the new instance, telling it to use the new option file.

5.3.2 Running Multiple MySQL Instances on Windows

You can run multiple servers on Windows by starting them manually from the command line, each with appropriate operating parameters, or by installing several servers as Windows services and running them that way. General instructions for running MySQL from the command line or as a service are given in Section 2.3, “Installing MySQL on Microsoft Windows”. The following sections describe how to start each server with different values for those options that must be unique per server, such as the data directory. These options are listed in Section 5.3, “Running Multiple MySQL Instances on One Machine”.

5.3.2.1 Starting Multiple MySQL Instances at the Windows Command Line

The procedure for starting a single MySQL server manually from the command line is described in Section 2.3.5.5, “Starting MySQL from the Windows Command Line”. To start multiple servers this way, you can specify the appropriate options on the command line or in an option file. It is more convenient to place the options in an option file, but it is necessary to make sure that each server gets its own set of options. To do this, create an option file for each server and tell the server the file name with a `--defaults-file` option when you run it.

Suppose that you want to run `mysqld` on port 3307 with a data directory of `C:\mydata1`, and `mysqld-debug` on port 3308 with a data directory of `C:\mydata2`. Use this procedure:

1. Make sure that each data directory exists, including its own copy of the `mysql` database that contains the grant tables.

2. Create two option files. For example, create one file named `C:\my-opts1.cnf` that looks like this:

   ```
   [mysqld]
   datadir = C:/mydata1
   port = 3307
   ```

   Create a second file named `C:\my-opts2.cnf` that looks like this:

   ```
   [mysqld]
   datadir = C:/mydata2
   port = 3308
   ```

3. Use the `--defaults-file` option to start each server with its own option file:

```
C:\> C:\mysql\bin\mysqld --defaults-file=C:\my-opts1.cnf
C:\> C:\mysql\bin\mysqld-debug --defaults-file=C:\my-opts2.cnf
```

Each server starts in the foreground (no new prompt appears until the server exits later), so you will need to issue those two commands in separate console windows.

To shut down the servers, connect to each using the appropriate port number:

```
C:\> C:\mysql\bin\mysqladmin --port=3307 shutdown
C:\> C:\mysql\bin\mysqladmin --port=3308 shutdown
```

Servers configured as just described permit clients to connect over TCP/IP. If your version of Windows supports named pipes and you also want to permit named-pipe connections, use the `mysqld` or `mysqld-debug` server and specify options that enable the named pipe and specify its name. Each server that supports named-pipe connections must use a unique pipe name. For example, the `C:\my-opts1.cnf` file might be written like this:

```
[mysqld]
datadir = C:/mydata1
port = 3307
enable-named-pipe
socket = mypipe1
```

Modify `C:\my-opts2.cnf` similarly for use by the second server. Then start the servers as described previously.

A similar procedure applies for servers that you want to permit shared-memory connections. Enable such connections with the `--shared-memory` option and specify a unique shared-memory name for each server with the `--shared-memory-base-name` option.

5.3.2.2 Starting Multiple MySQL Instances as Windows Services

On Windows, a MySQL server can run as a Windows service. The procedures for installing, controlling, and removing a single MySQL service are described in Section 2.3.5.7, “Starting MySQL as a Windows Service”.

To set up multiple MySQL services, you must make sure that each instance uses a different service name in addition to the other parameters that must be unique per instance.

For the following instructions, suppose that you want to run the `mysqld` server from two different versions of MySQL that are installed at `C:\mysql-5.5.9` and `C:\mysql-5.7.5`, respectively. (This might be the case if you are running 5.5.9 as your production server, but also want to conduct tests using 5.7.5.)

To install MySQL as a Windows service, use the `--install` or `--install-manual` option. For information about these options, see Section 2.3.5.7, “Starting MySQL as a Windows Service”.

Based on the preceding information, you have several ways to set up multiple services. The following instructions describe some examples. Before trying any of them, shut down and remove any existing MySQL services.

- **Approach 1:** Specify the options for all services in one of the standard option files. To do this, use a different service name for each server. Suppose that you want to run the 5.5.9 `mysqld` using the service name of `mysqld1` and the 5.7.5 `mysqld` using the service name `mysqld2`. In this case, you can use the `[mysqld1]` group for 5.5.9 and the `[mysqld2]` group for 5.7.5. For example, you can set up `C:\my.cnf` like this:

```
# options for mysqld1 service
[mysqld1]
basedir = C:/mysql-5.5.9
port = 3307
enable-named-pipe
socket = mypipe1

# options for mysqld2 service
[mysqld2]
basedir = C:/mysql-5.7.5
port = 3308
enable-named-pipe
socket = mypipe2
```

Install the services as follows, using the full server path names to ensure that Windows registers the correct executable program for each service:

```
C:\> C:\mysql-5.5.9\bin\mysqld --install mysqld1
C:\> C:\mysql-5.7.5\bin\mysqld --install mysqld2
```

To start the services, use the services manager, or use `NET START` with the appropriate service names:

```
C:\> NET START mysqld1
C:\> NET START mysqld2
```

To stop the services, use the services manager, or use `NET STOP` with the appropriate service names:

```
C:\> NET STOP mysqld1
C:\> NET STOP mysqld2
```

- **Approach 2:** Specify options for each server in separate files and use `--defaults-file` when you install the services to tell each server what file to use. In this case, each file should list options using a `[mysqld]` group.

  With this approach, to specify options for the 5.5.9 `mysqld`, create a file `C:\my-opts1.cnf` that looks like this:

  ```
  [mysqld]
  basedir = C:/mysql-5.5.9
  port = 3307
  enable-named-pipe
  socket = mypipe1
  ```

  For the 5.7.5 `mysqld`, create a file `C:\my-opts2.cnf` that looks like this:

  ```
  [mysqld]
  basedir = C:/mysql-5.7.5
  port = 3308
  enable-named-pipe
  socket = mypipe2
  ```

  Install the services as follows (enter each command on a single line):

  ```
  C:\> C:\mysql-5.5.9\bin\mysqld --install mysqld1
             --defaults-file=C:\my-opts1.cnf
  C:\> C:\mysql-5.7.5\bin\mysqld --install mysqld2
             --defaults-file=C:\my-opts2.cnf
  ```

When you install a MySQL server as a service and use a --defaults-file option, the service name must precede the option.

After installing the services, start and stop them the same way as in the preceding example.

To remove multiple services, use mysqld --remove for each one, specifying a service name following the --remove option. If the service name is the default (MySQL), you can omit it.

5.3.3 Running Multiple MySQL Instances on Unix

One way is to run multiple MySQL instances on Unix is to compile different servers with different default TCP/IP ports and Unix socket files so that each one listens on different network interfaces. Compiling in different base directories for each installation also results automatically in a separate, compiled-in data directory, log file, and PID file location for each server.

Assume that an existing 5.6 server is configured for the default TCP/IP port number (3306) and Unix socket file (/tmp/mysql.sock). To configure a new 5.7.5 server to have different operating parameters, use a CMake command something like this:

```
shell> cmake . -DMYSQL_TCP_PORT=port_number \
             -DMYSQL_UNIX_ADDR=file_name \
             -DCMAKE_INSTALL_PREFIX=/usr/local/mysql-5.7.5
```

Here, *port_number* and *file_name* must be different from the default TCP/IP port number and Unix socket file path name, and the CMAKE_INSTALL_PREFIX value should specify an installation directory different from the one under which the existing MySQL installation is located.

If you have a MySQL server listening on a given port number, you can use the following command to find out what operating parameters it is using for several important configurable variables, including the base directory and Unix socket file name:

```
shell> mysqladmin --host=host_name --port=port_number variables
```

With the information displayed by that command, you can tell what option values *not* to use when configuring an additional server.

If you specify localhost as the host name, mysqladmin defaults to using a Unix socket file connection rather than TCP/IP. To explicitly specify the connection protocol, use the --protocol={TCP|SOCKET|PIPE|MEMORY} option.

You need not compile a new MySQL server just to start with a different Unix socket file and TCP/IP port number. It is also possible to use the same server binary and start each invocation of it with different parameter values at runtime. One way to do so is by using command-line options:

```
shell> mysqld_safe --socket=file_name --port=port_number
```

To start a second server, provide different --socket and --port option values, and pass a --datadir=*path* option to mysqld_safe so that the server uses a different data directory.

Alternatively, put the options for each server in a different option file, then start each server using a --defaults-file option that specifies the path to the appropriate option file. For example, if the option files for two server instances are named /usr/local/mysql/my.cnf and /usr/local/mysql/my.cnf2, start the servers like this: command:

```
shell> mysqld_safe --defaults-file=/usr/local/mysql/my.cnf
shell> mysqld_safe --defaults-file=/usr/local/mysql/my.cnf2
```

Another way to achieve a similar effect is to use environment variables to set the Unix socket file name and TCP/IP port number:

```
shell> MYSQL_UNIX_PORT=/tmp/mysqld-new.sock
shell> MYSQL_TCP_PORT=3307
shell> export MYSQL_UNIX_PORT MYSQL_TCP_PORT
shell> mysql_install_db --user=mysql
shell> mysqld_safe --datadir=/path/to/datadir &
```

This is a quick way of starting a second server to use for testing. The nice thing about this method is that the environment variable settings apply to any client programs that you invoke from the same shell. Thus, connections for those clients are automatically directed to the second server.

Section 2.11, “Environment Variables”, includes a list of other environment variables you can use to affect MySQL programs.

On Unix, the `mysqld_multi` script provides another way to start multiple servers. See Section 4.3.4, “`mysqld_multi` — Manage Multiple MySQL Servers”.

5.3.4 Using Client Programs in a Multiple-Server Environment

To connect with a client program to a MySQL server that is listening to different network interfaces from those compiled into your client, you can use one of the following methods:

- Start the client with `--host=host_name --port=port_number` to connect using TCP/IP to a remote server, with `--host=127.0.0.1 --port=port_number` to connect using TCP/IP to a local server, or with `--host=localhost --socket=file_name` to connect to a local server using a Unix socket file or a Windows named pipe.

- Start the client with `--protocol=TCP` to connect using TCP/IP, `--protocol=SOCKET` to connect using a Unix socket file, `--protocol=PIPE` to connect using a named pipe, or `--protocol=MEMORY` to connect using shared memory. For TCP/IP connections, you may also need to specify `--host` and `--port` options. For the other types of connections, you may need to specify a `--socket` option to specify a Unix socket file or Windows named-pipe name, or a `--shared-memory-base-name` option to specify the shared-memory name. Shared-memory connections are supported only on Windows.

- On Unix, set the `MYSQL_UNIX_PORT` and `MYSQL_TCP_PORT` environment variables to point to the Unix socket file and TCP/IP port number before you start your clients. If you normally use a specific socket file or port number, you can place commands to set these environment variables in your `.login` file so that they apply each time you log in. See Section 2.11, “Environment Variables”.

- Specify the default Unix socket file and TCP/IP port number in the `[client]` group of an option file. For example, you can use `C:\my.cnf` on Windows, or the `.my.cnf` file in your home directory on Unix. See Section 4.2.3.3, “Using Option Files”.

- In a C program, you can specify the socket file or port number arguments in the `mysql_real_connect()` call. You can also have the program read option files by calling `mysql_options()`. See Section 21.8.7, “C API Function Descriptions”.

- If you are using the Perl `DBD::mysql` module, you can read options from MySQL option files. For example:

  ```
  $dsn = "DBI:mysql:test;mysql_read_default_group=client;"
  ```

```
        . "mysql_read_default_file=/usr/local/mysql/data/my.cnf";
$dbh = DBI->connect($dsn, $user, $password);
```

See Section 21.10, “MySQL Perl API”.

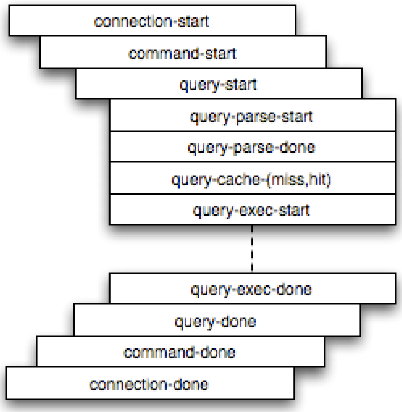
Other programming interfaces may provide similar capabilities for reading option files.

5.4 Tracing mysqld Using DTrace

The DTrace probes in the MySQL server are designed to provide information about the execution of queries within MySQL and the different areas of the system being utilized during that process. The organization and triggering of the probes means that the execution of an entire query can be monitored with one level of probes (query-start and query-done) but by monitoring other probes you can get successively more detailed information about the execution of the query in terms of the locks used, sort methods and even row-by-row and storage-engine level execution information.

The DTrace probes are organized so that you can follow the entire query process, from the point of connection from a client, through the query execution, row-level operations, and back out again. You can think of the probes as being fired within a specific sequence during a typical client connect/execute/disconnect sequence, as shown in the following figure.

Figure 5.1 The MySQL Architecture Using Pluggable Storage Engines



Global information is provided in the arguments to the DTrace probes at various levels. Global information, that is, the connection ID and user/host and where relevant the query string, is provided at key levels (connection-start, command-start, query-start, and query-exec-start). As you go deeper into the probes, it is assumed either you are only interested in the individual executions (row-level probes provide information on the database and table name only), or that you will combine the row-level probes with the notional parent probes to provide the information about a specific query. Examples of this will be given as the format and arguments of each probe are provided.

For more information on DTrace and writing DTrace scripts, read the DTrace User Guide.

MySQL 5.7 includes support for DTrace probes on Solaris 10 Update 5 (Solaris 5/08) on SPARC, x86 and x86_64 platforms. Probes are also supported on Mac OS X 10.4 and higher. Enabling the probes should be automatic on these platforms. To explicitly enable or disable the probes during building, use the -DENABLE_DTRACE=1 or -DENABLE_DTRACE=0 option to CMake.

If a non-Solaris platform includes DTrace support, building mysqld on that platform will include DTrace support.

5.4.1 mysqld DTrace Probe Reference

MySQL supports the following static probes, organized into groups of functionality.

Table 5.5 MySQL DTrace Probes

Group	Probes
Connection	connection-start, connection-done
Command	command-start, command-done
Query	query-start, query-done
Query Parsing	query-parse-start, query-parse-done
Query Cache	query-cache-hit, query-cache-miss
Query Execution	query-exec-start, query-exec-done
Row Level	insert-row-start, insert-row-done
	update-row-start, update-row-done
	delete-row-start, delete-row-done
Row Reads	read-row-start, read-row-done
Index Reads	index-read-row-start, index-read-row-done
Lock	handler-rdlock-start, handler-rdlock-done
	handler-wrlock-start, handler-wrlock-done
	handler-unlock-start, handler-unlock-done
Filesort	filesort-start, filesort-done
Statement	select-start, select-done
	insert-start, insert-done
	insert-select-start, insert-select-done
	update-start, update-done
	multi-update-start, multi-update-done
	delete-start, delete-done
	multi-delete-start, multi-delete-done
Network	net-read-start, net-read-done, net-write-start, net-write-done
Keycache	keycache-read-start, keycache-read-block, keycache-read-done, keycache-read-hit, keycache-read-miss, keycache-write-start, keycache-write-block, keycache-write-done

Note

When extracting the argument data from the probes, each argument is available as arg*N*, starting with arg0. To identify each argument within the definitions they are provided with a descriptive name, but you must access the information using the corresponding arg*N* parameter.

5.4.1.1 Connection Probes

The connection-start and connection-done probes enclose a connection from a client, regardless of whether the connection is through a socket or network connection.

```
connection-start(connectionid, user, host)
connection-done(status, connectionid)
```

- connection-start: Triggered after a connection and successful login/authentication have been completed by a client. The arguments contain the connection information:
    - connectionid: An unsigned long containing the connection ID. This is the same as the process ID shown as the Id value in the output from SHOW PROCESSLIST.
    - user: The username used when authenticating. The value will be blank for the anonymous user.
    - host: The host of the client connection. For a connection made using UNIX sockets, the value will be blank.
- connection-done: Triggered just as the connection to the client has been closed. The arguments are:
    - status: The status of the connection when it was closed. A logout operation will have a value of 0; any other termination of the connection has a nonzero value.
    - connectionid: The connection ID of the connection that was closed.

The following D script will quantify and summarize the average duration of individual connections, and provide a count, dumping the information every 60 seconds:

```
#!/usr/sbin/dtrace -s


mysql*:::connection-start
{
  self->start = timestamp;
}

mysql*:::connection-done
/self->start/
{
  @ = quantize(((timestamp - self->start)/1000000));
  self->start = 0;
}

tick-60s
{
  printa(@);
}
```

When executed on a server with a large number of clients you might see output similar to this:

```
  1  57413                        :tick-60s

           value  ------------- Distribution ------------- count
              -1 |                                         0
               0 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ 30011
               1 |                                         59
               2 |                                         5
               4 |                                         20
               8 |                                         29
              16 |                                         18
              32 |                                         27
              64 |                                         30
             128 |                                         11
             256 |                                         10
             512 |                                         1
            1024 |                                         6
            2048 |                                         8
            4096 |                                         9
            8192 |                                         8
           16384 |                                         2
           32768 |                                         1
           65536 |                                         1
          131072 |                                         0
          262144 |                                         1
          524288 |                                         0
```

5.4.1.2 Command Probes

The command probes are executed before and after a client command is executed, including any SQL statement that might be executed during that period. Commands include operations such as the initialization of the DB, use of the COM_CHANGE_USER operation (supported by the MySQL protocol), and manipulation of prepared statements. Many of these commands are used only by the MySQL client API from various connectors such as PHP and Java.

```
command-start(connectionid, command, user, host)
command-done(status)
```

- command-start: Triggered when a command is submitted to the server.
  - connectionid: The connection ID of the client executing the command.
  - command: An integer representing the command that was executed. Possible values are shown in the following table.

Value	Name	Description
00	COM_SLEEP	Internal thread state
01	COM_QUIT	Close connection
02	COM_INIT_DB	Select database (USE ...)
03	COM_QUERY	Execute a query
04	COM_FIELD_LIST	Get a list of fields
05	COM_CREATE_DB	Create a database (deprecated)
06	COM_DROP_DB	Drop a database (deprecated)
07	COM_REFRESH	Refresh connection
08	COM_SHUTDOWN	Shutdown server
09	COM_STATISTICS	Get statistics
10	COM_PROCESS_INFO	Get processes (SHOW PROCESSLIST)
11	COM_CONNECT	Initialize connection
12	COM_PROCESS_KILL	Kill process
13	COM_DEBUG	Get debug information
14	COM_PING	Ping
15	COM_TIME	Internal thread state
16	COM_DELAYED_INSERT	Internal thread state
17	COM_CHANGE_USER	Change user
18	COM_BINLOG_DUMP	Used by a replication slave or mysqlbinlog to initiate a binary log read
19	COM_TABLE_DUMP	Used by a replication slave to get the master table information

Value	Name	Description
20	COM_CONNECT_OUT	Used by a replication slave to log a connection to the server
21	COM_REGISTER_SLAVE	Used by a replication slave during registration
22	COM_STMT_PREPARE	Prepare a statement
23	COM_STMT_EXECUTE	Execute a statement
24	COM_STMT_SEND_LONG_DATA	Used by a client when requesting extended data
25	COM_STMT_CLOSE	Close a prepared statement
26	COM_STMT_RESET	Reset a prepared statement
27	COM_SET_OPTION	Set a server option
28	COM_STMT_FETCH	Fetch a prepared statement

  - `user`: The user executing the command.

  - `host`: The client host.

- `command-done`: Triggered when the command execution completes. The `status` argument contains 0 if the command executed successfully, or 1 if the statement was terminated before normal completion.

The `command-start` and `command-done` probes are best used when combined with the statement probes to get an idea of overall execution time.

5.4.1.3 Query Probes

The `query-start` and `query-done` probes are triggered when a specific query is received by the server and when the query has been completed and the information has been successfully sent to the client.

```
query-start(query, connectionid, database, user, host)
query-done(status)
```

- `query-start`: Triggered after the query string has been received from the client. The arguments are:

  - `query`: The full text of the submitted query.

  - `connectionid`: The connection ID of the client that submitted the query. The connection ID equals the connection ID returned when the client first connects and the `Id` value in the output from `SHOW PROCESSLIST`.

  - `database`: The database name on which the query is being executed.

  - `user`: The username used to connect to the server.

  - `host`: The hostname of the client.

- `query-done`: Triggered once the query has been executed and the information has been returned to the client. The probe includes a single argument, `status`, which returns 0 when the query is successfully executed and 1 if there was an error.

You can get a simple report of the execution time for each query using the following D script:

```
#!/usr/sbin/dtrace -s

#pragma D option quiet

dtrace:::BEGIN
{
   printf("%-20s %-20s %-40s %-9s\n", "Who", "Database", "Query", "Time(ms)");
}

mysql*:::query-start
{
   self->query = copyinstr(arg0);
   self->connid = arg1;
   self->db    = copyinstr(arg2);
   self->who   = strjoin(copyinstr(arg3),strjoin("@",copyinstr(arg4)));
   self->querystart = timestamp;
}

mysql*:::query-done
{
   printf("%-20s %-20s %-40s %-9d\n",self->who,self->db,self->query,
          (timestamp - self->querystart) / 1000000);
}
```

When executing the above script you should get a basic idea of the execution time of your queries:

```
shell> ./query.d
Who                  Database             Query                                    Time(ms)
root@localhost       test                 select * from t1 order by i limit 10     0
root@localhost       test                 set global query_cache_size=0            0
root@localhost       test                 select * from t1 order by i limit 10     776
root@localhost       test                 select * from t1 order by i limit 10     773
root@localhost       test                 select * from t1 order by i desc limit 10 795
```

5.4.1.4 Query Parsing Probes

The query parsing probes are triggered before the original SQL statement is parsed and when the parsing of the statement and determination of the execution model required to process the statement has been completed:

```
query-parse-start(query)
query-parse-done(status)
```

- query-parse-start: Triggered just before the statement is parsed by the MySQL query parser. The single argument, query, is a string containing the full text of the original query.

- query-parse-done: Triggered when the parsing of the original statement has been completed. The status is an integer describing the status of the operation. A 0 indicates that the query was successfully parsed. A 1 indicates that the parsing of the query failed.

For example, you could monitor the execution time for parsing a given query using the following D script:

```
#!/usr/sbin/dtrace -s

#pragma D option quiet

mysql*:::query-parse-start
{
   self->parsestart = timestamp;
   self->parsequery = copyinstr(arg0);
}

mysql*:::query-parse-done
/arg0 == 0/
{
   printf("Parsing %s: %d microseconds\n", self->parsequery,((timestamp - self->parsestart)/1000));
}

mysql*:::query-parse-done
/arg0 != 0/
```

```
{
   printf("Error parsing %s: %d microseconds\n", self->parsequery,((timestamp - self->parsestart)/1000));
}
```

In the above script a predicate is used on `query-parse-done` so that different output is generated based on the status value of the probe.

When running the script and monitoring the execution:

```
shell> ./query-parsing.d
Error parsing select from t1 join (t2) on (t1.i = t2.i) order by t1.s,t1.i limit 10: 36 ms
Parsing select * from t1 join (t2) on (t1.i = t2.i) order by t1.s,t1.i limit 10: 176 ms
```

5.4.1.5 Query Cache Probes

The query cache probes are fired when executing any query. The `query-cache-hit` query is triggered when a query exists in the query cache and can be used to return the query cache information. The arguments contain the original query text and the number of rows returned from the query cache for the query. If the query is not within the query cache, or the query cache is not enabled, then the `query-cache-miss` probe is triggered instead.

```
query-cache-hit(query, rows)
query-cache-miss(query)
```

- `query-cache-hit`: Triggered when the query has been found within the query cache. The first argument, `query`, contains the original text of the query. The second argument, `rows`, is an integer containing the number of rows in the cached query.

- `query-cache-miss`: Triggered when the query is not found within the query cache. The first argument, `query`, contains the original text of the query.

The query cache probes are best combined with a probe on the main query so that you can determine the differences in times between using or not using the query cache for specified queries. For example, in the following D script, the query and query cache information are combined into the information output during monitoring:

```
#!/usr/sbin/dtrace -s

#pragma D option quiet

dtrace:::BEGIN
{
   printf("%-20s %-20s %-40s %2s %-9s\n", "Who", "Database", "Query", "QC", "Time(ms)");
}

mysql*:::query-start
{
   self->query = copyinstr(arg0);
   self->connid = arg1;
   self->db    = copyinstr(arg2);
   self->who   = strjoin(copyinstr(arg3),strjoin("@",copyinstr(arg4)));
   self->querystart = timestamp;
   self->qc = 0;
}

mysql*:::query-cache-hit
{
   self->qc = 1;
}

mysql*:::query-cache-miss
{
   self->qc = 0;
}

mysql*:::query-done
{
   printf("%-20s %-20s %-40s %-2s %-9d\n",self->who,self->db,self->query,(self->qc ? "Y" : "N"),
          (timestamp - self->querystart) / 1000000);
}
```

When executing the script you can see the effects of the query cache. Initially the query cache is disabled. If you set the query cache size and then execute the query multiple times you should see that the query cache is being used to return the query data:

```
shell> ./query-cache.d
root@localhost       test                 select * from t1 order by i limit 10     N  1072
root@localhost                            set global query_cache_size=262144       N  0
root@localhost       test                 select * from t1 order by i limit 10     N  781
root@localhost       test                 select * from t1 order by i limit 10     Y  0
```

5.4.1.6 Query Execution Probes

The query execution probe is triggered when the actual execution of the query starts, after the parsing and checking the query cache but before any privilege checks or optimization. By comparing the difference between the start and done probes you can monitor the time actually spent servicing the query (instead of just handling the parsing and other elements of the query).

```

query-exec-start(query, connectionid, database, user, host, exec_type)
query-exec-done(status)
```

Note

The information provided in the arguments for query-start and query-exec-start are almost identical and designed so that you can choose to monitor either the entire query process (using query-start) or only the execution (using query-exec-start) while exposing the core information about the user, client, and query being executed.

- query-exec-start: Triggered when the execution of a individual query is started. The arguments are:
  - query: The full text of the submitted query.
  - connectionid: The connection ID of the client that submitted the query. The connection ID equals the connection ID returned when the client first connects and the Id value in the output from SHOW PROCESSLIST.
  - database: The database name on which the query is being executed.
  - user: The username used to connect to the server.
  - host: The hostname of the client.
  - exec_type: The type of execution. Execution types are determined based on the contents of the query and where it was submitted. The values for each type are shown in the following table.

Value	Description
0	Executed query from sql_parse, top-level query.
1	Executed prepared statement
2	Executed cursor statement

Value	Description
3	Executed query in stored procedure

- query-exec-done: Triggered when the execution of the query has completed. The probe includes a single argument, status, which returns 0 when the query is successfully executed and 1 if there was an error.

5.4.1.7 Row-Level Probes

The *row-{start,done} probes are triggered each time a row operation is pushed down to a storage engine. For example, if you execute an INSERT statement with 100 rows of data, then the insert-row-start and insert-row-done probes will be triggered 100 times each, for each row insert.

```
insert-row-start(database, table)
insert-row-done(status)

update-row-start(database, table)
update-row-done(status)

delete-row-start(database, table)
delete-row-done(status)
```

- insert-row-start: Triggered before a row is inserted into a table.
- insert-row-done: Triggered after a row is inserted into a table.
- update-row-start: Triggered before a row is updated in a table.
- update-row-done: Triggered before a row is updated in a table.
- delete-row-start: Triggered before a row is deleted from a table.
- delete-row-done: Triggered before a row is deleted from a table.

The arguments supported by the probes are consistent for the corresponding start and done probes in each case:

- database: The database name.
- table: The table name.
- status: The status; 0 for success or 1 for failure.

Because the row-level probes are triggered for each individual row access, these probes can be triggered many thousands of times each second, which may have a detrimental effect on both the monitoring script and MySQL. The DTrace environment should limit the triggering on these probes to prevent the performance being adversely affected. Either use the probes sparingly, or use counter or aggregation functions to report on these probes and then provide a summary when the script terminates or as part of a query-done or query-exec-done probes.

The following example script summarizes the duration of each row operation within a larger query:

```
#!/usr/sbin/dtrace -s

#pragma D option quiet

dtrace:::BEGIN
{
   printf("%-2s %-10s %-10s %9s %9s %-s \n",
          "St", "Who", "DB", "ConnID", "Dur ms", "Query");
}
```

```
mysql*:::query-start
{
   self->query = copyinstr(arg0);
   self->who   = strjoin(copyinstr(arg3),strjoin("@",copyinstr(arg4)));
   self->db    = copyinstr(arg2);
   self->connid = arg1;
   self->querystart = timestamp;
   self->rowdur = 0;
}

mysql*:::query-done
{
   this->elapsed = (timestamp - self->querystart) /1000000;
   printf("%2d %-10s %-10s %9d %9d %s\n",
          arg0, self->who, self->db,
          self->connid, this->elapsed, self->query);
}

mysql*:::query-done
/ self->rowdur /
{
   printf("%34s %9d %s\n", "", (self->rowdur/1000000), "-> Row ops");
}

mysql*:::insert-row-start
{
   self->rowstart = timestamp;
}

mysql*:::delete-row-start
{
   self->rowstart = timestamp;
}

mysql*:::update-row-start
{
   self->rowstart = timestamp;
}

mysql*:::insert-row-done
{
   self->rowdur += (timestamp-self->rowstart);
}

mysql*:::delete-row-done
{
   self->rowdur += (timestamp-self->rowstart);
}

mysql*:::update-row-done
{
   self->rowdur += (timestamp-self->rowstart);
}
```

Running the above script with a query that inserts data into a table, you can monitor the exact time spent performing the raw row insertion:

```
St Who        DB            ConnID    Dur ms Query
 0 @localhost test              13     20767 insert into t1(select * from t2)
                                        4827 -> Row ops
```

5.4.1.8 Read Row Probes

The read row probes are triggered at a storage engine level each time a row read operation occurs. These probes are specified within each storage engine (as opposed to the *row-start probes which are in the storage engine interface). These probes can therefore be used to monitor individual storage engine row-level operations and performance. Because these probes are triggered around the storage engine row read interface, they may be hit a significant number of times during a basic query.

```
read-row-start(database, table, scan_flag)
read-row-done(status)
```

- `read-row-start`: Triggered when a row is read by the storage engine from the specified `database` and `table`. The `scan_flag` is set to 1 (true) when the read is part of a table scan (that is, a sequential read), or 0 (false) when the read is of a specific record.
- `read-row-done`: Triggered when a row read operation within a storage engine completes. The `status` returns 0 on success, or a positive value on failure.

5.4.1.9 Index Probes

The index probes are triggered each time a a row is read using one of the indexes for the specified table. The probe is triggered within the corresponding storage engine for the table.

```
index-read-row-start(database, table)
index-read-row-done(status)
```

- `index-read-row-start`: Triggered when a row is read by the storage engine from the specified `database` and `table`.
- `index-read-row-done`: Triggered when an indexed row read operation within a storage engine completes. The `status` returns 0 on success, or a positive value on failure.

5.4.1.10 Lock Probes

The lock probes are called whenever an external lock is requested by MySQL for a table using the corresponding lock mechanism on the table as defined by the table's engine type. There are three different types of lock, the read lock, write lock, and unlock operations. Using the probes you can determine the duration of the external locking routine (that is, the time taken by the storage engine to implement the lock, including any time waiting for another lock to become free) and the total duration of the lock/unlock process.

```
handler-rdlock-start(database, table)
handler-rdlock-done(status)

handler-wrlock-start(database, table)
handler-wrlock-done(status)

handler-unlock-start(database, table)
handler-unlock-done(status)
```

- `handler-rdlock-start`: Triggered when a read lock is requested on the specified `database` and `table`.
- `handler-wrlock-start`: Triggered when a write lock is requested on the specified `database` and `table`.
- `handler-unlock-start`: Triggered when an unlock request is made on the specified `database` and `table`.
- `handler-rdlock-done`: Triggered when a read lock request completes. The `status` is 0 if the lock operation succeeded, or `>0` on failure.
- `handler-wrlock-done`: Triggered when a write lock request completes. The `status` is 0 if the lock operation succeeded, or `>0` on failure.

- handler-unlock-done: Triggered when an unlock request completes. The status is 0 if the unlock operation succeeded, or >0 on failure.

You can use arrays to monitor the locking and unlocking of individual tables and then calculate the duration of the entire table lock using the following script:

```
#!/usr/sbin/dtrace -s

#pragma D option quiet

mysql*:::handler-rdlock-start
{
   self->rdlockstart = timestamp;
   this->lockref = strjoin(copyinstr(arg0),strjoin("@",copyinstr(arg1)));
   self->lockmap[this->lockref] = self->rdlockstart;
   printf("Start: Lock->Read   %s.%s\n",copyinstr(arg0),copyinstr(arg1));
}

mysql*:::handler-wrlock-start
{
   self->wrlockstart = timestamp;
   this->lockref = strjoin(copyinstr(arg0),strjoin("@",copyinstr(arg1)));
   self->lockmap[this->lockref] = self->rdlockstart;
   printf("Start: Lock->Write  %s.%s\n",copyinstr(arg0),copyinstr(arg1));
}

mysql*:::handler-unlock-start
{
   self->unlockstart = timestamp;
   this->lockref = strjoin(copyinstr(arg0),strjoin("@",copyinstr(arg1)));
   printf("Start: Lock->Unlock %s.%s (%d ms lock duration)\n",
          copyinstr(arg0),copyinstr(arg1),
          (timestamp - self->lockmap[this->lockref])/1000000);
}

mysql*:::handler-rdlock-done
{
   printf("End:   Lock->Read   %d ms\n",
          (timestamp - self->rdlockstart)/1000000);
}

mysql*:::handler-wrlock-done
{
   printf("End:   Lock->Write  %d ms\n",
          (timestamp - self->wrlockstart)/1000000);
}

mysql*:::handler-unlock-done
{
   printf("End:   Lock->Unlock %d ms\n",
          (timestamp - self->unlockstart)/1000000);
}
```

When executed, you should get information both about the duration of the locking process itself, and of the locks on a specific table:

```
Start: Lock->Read   test.t2
End:   Lock->Read   0 ms
Start: Lock->Unlock test.t2 (25743 ms lock duration)
End:   Lock->Unlock 0 ms
Start: Lock->Read   test.t2
End:   Lock->Read   0 ms
Start: Lock->Unlock test.t2 (1 ms lock duration)
End:   Lock->Unlock 0 ms
Start: Lock->Read   test.t2
End:   Lock->Read   0 ms
```

```
Start: Lock->Unlock test.t2 (1 ms lock duration)
End:   Lock->Unlock 0 ms
Start: Lock->Read   test.t2
End:   Lock->Read   0 ms
```

5.4.1.11 Filesort Probes

The filesort probes are triggered whenever a filesort operation is applied to a table. For more information on filesort and the conditions under which it occurs, see Section 8.2.1.15, “ORDER BY Optimization”.

```
filesort-start(database, table)
filesort-done(status, rows)
```

- filesort-start: Triggered when the filesort operation starts on a table. The two arguments to the probe, database and table, will identify the table being sorted.
- filesort-done: Triggered when the filesort operation completes. Two arguments are supplied, the status (0 for success, 1 for failure), and the number of rows sorted during the filesort process.

An example of this is in the following script, which tracks the duration of the filesort process in addition to the duration of the main query:

```
#!/usr/sbin/dtrace -s

#pragma D option quiet

dtrace:::BEGIN
{
   printf("%-2s %-10s %-10s %9s %18s %-s \n",
          "St", "Who", "DB", "ConnID", "Dur microsec", "Query");
}

mysql*:::query-start
{
   self->query = copyinstr(arg0);
   self->who   = strjoin(copyinstr(arg3),strjoin("@",copyinstr(arg4)));
   self->db    = copyinstr(arg2);
   self->connid = arg1;
   self->querystart = timestamp;
   self->filesort = 0;
   self->fsdb = "";
   self->fstable = "";
}

mysql*:::filesort-start
{
  self->filesort = timestamp;
  self->fsdb = copyinstr(arg0);
  self->fstable = copyinstr(arg1);
}

mysql*:::filesort-done
{
   this->elapsed = (timestamp - self->filesort) /1000;
   printf("%2d %-10s %-10s %9d %18d Filesort on %s\n",
          arg0, self->who, self->fsdb,
          self->connid, this->elapsed, self->fstable);
}

mysql*:::query-done
{
   this->elapsed = (timestamp - self->querystart) /1000;
   printf("%2d %-10s %-10s %9d %18d %s\n",
          arg0, self->who, self->db,
          self->connid, this->elapsed, self->query);
}
```

Executing a query on a large table with an ORDER BY clause that triggers a filesort, and then creating an index on the table and then repeating the same query, you can see the difference in execution speed:

```
St Who        DB             ConnID       Dur microsec Query
 0 @localhost test               14           11335469 Filesort on t1
 0 @localhost test               14           11335787 select * from t1 order by i limit 100
 0 @localhost test               14          466734378 create index t1a on t1 (i)
 0 @localhost test               14              26472 select * from t1 order by i limit 100
```

5.4.1.12 Statement Probes

The individual statement probes are provided to give specific information about different statement types. For the start probes the string of the query is provided as a the only argument. Depending on the statement type, the information provided by the corresponding done probe will differ. For all done probes the status of the operation (0 for success, >0 for failure) is provided. For SELECT, INSERT, INSERT ... (SELECT FROM ...), DELETE, and DELETE FROM t1,t2 operations the number of rows affected is returned.

For UPDATE and UPDATE t1,t2 ... statements the number of rows matched and the number of rows actually changed is provided. This is because the number of rows actually matched by the corresponding WHERE clause, and the number of rows changed can differ. MySQL does not update the value of a row if the value already matches the new setting.

```
select-start(query)
select-done(status,rows)

insert-start(query)
insert-done(status,rows)

insert-select-start(query)
insert-select-done(status,rows)

update-start(query)
update-done(status,rowsmatched,rowschanged)

multi-update-start(query)
multi-update-done(status,rowsmatched,rowschanged)

delete-start(query)
delete-done(status,rows)

multi-delete-start(query)
multi-delete-done(status,rows)
```

- select-start: Triggered before a SELECT statement.
- select-done: Triggered at the end of a SELECT statement.
- insert-start: Triggered before a INSERT statement.
- insert-done: Triggered at the end of an INSERT statement.
- insert-select-start: Triggered before an INSERT ... SELECT statement.
- insert-select-done: Triggered at the end of an INSERT ... SELECT statement.
- update-start: Triggered before an UPDATE statement.
- update-done: Triggered at the end of an UPDATE statement.
- multi-update-start: Triggered before an UPDATE statement involving multiple tables.

- multi-update-done: Triggered at the end of an UPDATE statement involving multiple tables.
- delete-start: Triggered before a DELETE statement.
- delete-done: Triggered at the end of a DELETE statement.
- multi-delete-start: Triggered before a DELETE statement involving multiple tables.
- multi-delete-done: Triggered at the end of a DELETE statement involving multiple tables.

The arguments for the statement probes are:

- query: The query string.
- status: The status of the query. 0 for success, and >0 for failure.
- rows: The number of rows affected by the statement. This returns the number rows found for SELECT, the number of rows deleted for DELETE, and the number of rows successfully inserted for INSERT.
- rowsmatched: The number of rows matched by the WHERE clause of an UPDATE operation.
- rowschanged: The number of rows actually changed during an UPDATE operation.

You use these probes to monitor the execution of these statement types without having to monitor the user or client executing the statements. A simple example of this is to track the execution times:

```
#!/usr/sbin/dtrace -s

#pragma D option quiet

dtrace:::BEGIN
{
   printf("%-60s %-8s %-8s %-8s\n", "Query", "RowsU", "RowsM", "Dur (ms)");
}

mysql*:::update-start, mysql*:::insert-start,
mysql*:::delete-start, mysql*:::multi-delete-start,
mysql*:::multi-delete-done, mysql*:::select-start,
mysql*:::insert-select-start, mysql*:::multi-update-start
{
    self->query = copyinstr(arg0);
    self->querystart = timestamp;
}

mysql*:::insert-done, mysql*:::select-done,
mysql*:::delete-done, mysql*:::multi-delete-done, mysql*:::insert-select-done
/ self->querystart /
{
    this->elapsed = ((timestamp - self->querystart)/1000000);
    printf("%-60s %-8d %-8d %d\n",
           self->query,
           0,
           arg1,
           this->elapsed);
    self->querystart = 0;
}

mysql*:::update-done, mysql*:::multi-update-done
/ self->querystart /
{
    this->elapsed = ((timestamp - self->querystart)/1000000);
    printf("%-60s %-8d %-8d %d\n",
           self->query,
           arg1,
           arg2,
           this->elapsed);
    self->querystart = 0;
}
```

When executed you can see the basic execution times and rows matches:

```
Query                                                        RowsU    RowsM    Dur (ms)
select * from t2                                             0        275      0
insert into t2 (select * from t2)                            0        275      9
update t2 set i=5 where i > 75                               110      110      8
update t2 set i=5 where i < 25                               254      134      12
delete from t2 where i < 5                                   0        0        0
```

Another alternative is to use the aggregation functions in DTrace to aggregate the execution time of individual statements together:

```
#!/usr/sbin/dtrace -s

#pragma D option quiet


mysql*:::update-start, mysql*:::insert-start,
mysql*:::delete-start, mysql*:::multi-delete-start,
mysql*:::multi-delete-done, mysql*:::select-start,
mysql*:::insert-select-start, mysql*:::multi-update-start
{
    self->querystart = timestamp;
}

mysql*:::select-done
{
        @statements["select"] = sum(((timestamp - self->querystart)/1000000));
}

mysql*:::insert-done, mysql*:::insert-select-done
{
        @statements["insert"] = sum(((timestamp - self->querystart)/1000000));
}

mysql*:::update-done, mysql*:::multi-update-done
{
        @statements["update"] = sum(((timestamp - self->querystart)/1000000));
}

mysql*:::delete-done, mysql*:::multi-delete-done
{
        @statements["delete"] = sum(((timestamp - self->querystart)/1000000));
}

tick-30s
{
        printa(@statements);
}
```

The script just shown aggregates the times spent doing each operation, which could be used to help benchmark a standard suite of tests.

```
 delete                                                            0
  update                                                           0
  insert                                                          23
  select                                                        2484

  delete                                                           0
  update                                                           0
  insert                                                          39
  select                                                          10744

  delete                                                              0
  update                                                             26
  insert                                                             56
  select                                                          10944

  delete                                                              0
  update                                                             26
  insert                                                           2287
  select                                                          15985
```

5.4.1.13 Network Probes

The network probes monitor the transfer of information from the MySQL server and clients of all types over the network. The probes are defined as follows:

```
net-read-start()
net-read-done(status, bytes)
net-write-start(bytes)
net-write-done(status)
```

- `net-read-start`: Triggered when a network read operation is started.
- `net-read-done`: Triggered when the network read operation completes. The `status` is an `integer` representing the return status for the operation, `0` for success and `1` for failure. The `bytes` argument is an integer specifying the number of bytes read during the process.
- `net-start-bytes`: Triggered when data is written to a network socket. The single argument, `bytes`, specifies the number of bytes written to the network socket.
- `net-write-done`: Triggered when the network write operation has completed. The single argument, `status`, is an integer representing the return status for the operation, `0` for success and `1` for failure.

You can use the network probes to monitor the time spent reading from and writing to network clients during execution. The following D script provides an example of this. Both the cumulative time for the read or write is calculated, and the number of bytes. Note that the dynamic variable size has been increased (using the `dynvarsize` option) to cope with the rapid firing of the individual probes for the network reads/writes.

```
#!/usr/sbin/dtrace -s

#pragma D option quiet
#pragma D option dynvarsize=4m

dtrace:::BEGIN
{
   printf("%-2s %-30s %-10s %9s %18s %-s \n",
          "St", "Who", "DB", "ConnID", "Dur microsec", "Query");
}

mysql*:::query-start
{
   self->query = copyinstr(arg0);
   self->who   = strjoin(copyinstr(arg3),strjoin("@",copyinstr(arg4)));
   self->db    = copyinstr(arg2);
   self->connid = arg1;
   self->querystart = timestamp;
   self->netwrite = 0;
   self->netwritecum = 0;
   self->netwritebase = 0;
   self->netread = 0;
   self->netreadcum = 0;
   self->netreadbase = 0;
}

mysql*:::net-write-start
{
   self->netwrite += arg0;
   self->netwritebase = timestamp;
}

mysql*:::net-write-done
{
   self->netwritecum += (timestamp - self->netwritebase);
   self->netwritebase = 0;
}

mysql*:::net-read-start
{
   self->netreadbase = timestamp;
}

mysql*:::net-read-done
{
   self->netread += arg1;
   self->netreadcum += (timestamp - self->netreadbase);
   self->netreadbase = 0;
}

mysql*:::query-done
{
   this->elapsed = (timestamp - self->querystart) /1000000;
   printf("%2d %-30s %-10s %9d %18d %s\n",
          arg0, self->who, self->db,
          self->connid, this->elapsed, self->query);
   printf("Net read: %d bytes (%d ms) write: %d bytes (%d ms)\n",
               self->netread, (self->netreadcum/1000000),
               self->netwrite, (self->netwritecum/1000000));
}
```

When executing the above script on a machine with a remote client, you can see that approximately a third of the time spent executing the query is related to writing the query results back to the client.

```
St Who                            DB            ConnID       Dur microsec Query
 0 root@::ffff:192.168.0.108      test              31              3495 select * from t1 limit 1000000
Net read: 0 bytes (0 ms) write: 10000075 bytes (1220 ms)
```

5.4.1.14 Keycache Probes

The keycache probes are triggered when using the index key cache used with the MyISAM storage engine. Probes exist to monitor when data is read into the keycache, cached key data is written from the cache into a cached file, or when accessing the keycache.

Keycache usage indicates when data is read or written from the index files into the cache, and can be used to monitor how efficient the memory allocated to the keycache is being used. A high number of keycache reads across a range of queries may indicate that the keycache is too small for size of data being accessed.

```
keycache-read-start(filepath, bytes, mem_used, mem_free)
keycache-read-block(bytes)
keycache-read-hit()
keycache-read-miss()
keycache-read-done(mem_used, mem_free)
keycache-write-start(filepath, bytes, mem_used, mem_free)
keycache-write-block(bytes)
keycache-write-done(mem_used, mem_free)
```

When reading data from the index files into the keycache, the process first initializes the read operation (indicated by keycache-read-start), then loads blocks of data (keycache-read-block), and then the read block is either matches the data being identified (keycache-read-hit) or more data needs to be read (keycache-read-miss). Once the read operation has completed, reading stops with the keycache-read-done.

Data will be read from the index file into the keycache only when the specified key is not already within the keycache.

- keycache-read-start: Triggered when the keycache read operation is started. Data is read from the specified filepath, reading the specified number of bytes. The mem_used and mem_avail indicate memory currently used by the keycache and the amount of memory available within the keycache.
- keycache-read-block: Triggered when the keycache reads a block of data, of the specified number of bytes, from the index file into the keycache.
- keycache-read-hit: Triggered when the block of data read from the index file matches the key data requested.
- keycache-read-miss: Triggered when the block of data read from the index file does not match the key data needed.
- keycache-read-done: Triggered when the keycache read operation has completed. The mem_used and mem_avail indicate memory currently used by the keycache and the amount of memory available within the keycache.

Keycache writes occur when the index information is updated during an INSERT, UPDATE, or DELETE operation, and the cached key information is flushed back to the index file.

- keycache-write-start: Triggered when the keycache write operation is started. Data is written to the specified filepath, reading the specified number of bytes. The mem_used and mem_avail indicate memory currently used by the keycache and the amount of memory available within the keycache.
- keycache-write-block: Triggered when the keycache writes a block of data, of the specified number of bytes, to the index file from the keycache.
- keycache-write-done: Triggered when the keycache write operation has completed. The mem_used and mem_avail indicate memory currently used by the keycache and the amount of memory available within the keycache.

Chapter 6 Security

Table of Contents

When thinking about security within a MySQL installation, you should consider a wide range of possible topics and how they affect the security of your MySQL server and related applications:

- General factors that affect security. These include choosing good passwords, not granting unnecessary privileges to users, ensuring application security by preventing SQL injections and data corruption, and others. See Section 6.1, "General Security Issues".

- Security of the installation itself. The data files, log files, and the all the application files of your installation should be protected to ensure that they are not readable or writable by unauthorized parties. For more information, see Section 2.9, "Postinstallation Setup and Testing".

- Access control and security within the database system itself, including the users and databases granted with access to the databases, views and stored programs in use within the database. For more information, see Section 6.2, "The MySQL Access Privilege System", and Section 6.3, "MySQL User Account Management".

- Network security of MySQL and your system. The security is related to the grants for individual users, but you may also wish to restrict MySQL so that it is available only locally on the MySQL server host, or to a limited set of other hosts.

- Ensure that you have adequate and appropriate backups of your database files, configuration and log files. Also be sure that you have a recovery solution in place and test that you are able to successfully recover the information from your backups. See Chapter 7, *Backup and Recovery*.

6.1 General Security Issues

This section describes general security issues to be aware of and what you can do to make your MySQL installation more secure against attack or misuse. For information specifically about the access control system that MySQL uses for setting up user accounts and checking database access, see Section 2.9, “Postinstallation Setup and Testing”.

For answers to some questions that are often asked about MySQL Server security issues, see Section B.9, “MySQL 5.7 FAQ: Security”.

6.1.1 Security Guidelines

Anyone using MySQL on a computer connected to the Internet should read this section to avoid the most common security mistakes.

In discussing security, it is necessary to consider fully protecting the entire server host (not just the MySQL server) against all types of applicable attacks: eavesdropping, altering, playback, and denial of service. We do not cover all aspects of availability and fault tolerance here.

MySQL uses security based on Access Control Lists (ACLs) for all connections, queries, and other operations that users can attempt to perform. There is also support for SSL-encrypted connections between MySQL clients and servers. Many of the concepts discussed here are not specific to MySQL at all; the same general ideas apply to almost all applications.

When running MySQL, follow these guidelines:

- **Do not ever give anyone (except MySQL `root` accounts) access to the `user` table in the `mysql` database!** This is critical.

- Learn how the MySQL access privilege system works (see Section 6.2, “The MySQL Access Privilege System”). Use the `GRANT` and `REVOKE` statements to control access to MySQL. Do not grant more privileges than necessary. Never grant privileges to all hosts.

  Checklist:

  - Try `mysql -u root`. If you are able to connect successfully to the server without being asked for a password, anyone can connect to your MySQL server as the MySQL `root` user with full privileges! Review the MySQL installation instructions, paying particular attention to the information about setting a `root` password. See Section 2.9.2, “Securing the Initial MySQL Accounts”.

  - Use the `SHOW GRANTS` statement to check which accounts have access to what. Then use the `REVOKE` statement to remove those privileges that are not necessary.

- Do not store cleartext passwords in your database. If your computer becomes compromised, the intruder can take the full list of passwords and use them. Instead, use `SHA2()`, `SHA1()`, `MD5()`, or some other one-way hashing function and store the hash value.

  To prevent password recovery using rainbow tables, do not use these functions on a plain password; instead, choose some string to be used as a salt, and use hash(hash(password)+salt) values.

- Do not choose passwords from dictionaries. Special programs exist to break passwords. Even passwords like “xfish98” are very bad. Much better is “duag98” which contains the same word “fish” but typed one key to the left on a standard QWERTY keyboard. Another method is to use a password that is taken from the first characters of each word in a sentence (for example, “Four score and seven years ago” results in a password of “Fsasya”). The password is easy to remember and type, but difficult to guess for someone who does not know the sentence. In this case, you can additionally substitute digits for the number words to obtain the phrase “4 score and 7 years ago”, yielding the password “4sa7ya” which is even more difficult to guess.

- Invest in a firewall. This protects you from at least 50% of all types of exploits in any software. Put MySQL behind the firewall or in a demilitarized zone (DMZ).

  Checklist:

  - Try to scan your ports from the Internet using a tool such as nmap. MySQL uses port 3306 by default. This port should not be accessible from untrusted hosts. As a simple way to check whether your MySQL port is open, try the following command from some remote machine, where *server_host* is the host name or IP address of the host on which your MySQL server runs:

    ```
    shell> telnet server_host 3306
    ```

    If telnet hangs or the connection is refused, the port is blocked, which is how you want it to be. If you get a connection and some garbage characters, the port is open, and should be closed on your firewall or router, unless you really have a good reason to keep it open.

- Applications that access MySQL should not trust any data entered by users, and should be written using proper defensive programming techniques. See Section 6.1.7, “Client Programming Security Guidelines”.

- Do not transmit plain (unencrypted) data over the Internet. This information is accessible to everyone who has the time and ability to intercept it and use it for their own purposes. Instead, use an encrypted protocol such as SSL or SSH. MySQL supports internal SSL connections. Another technique is to use SSH port-forwarding to create an encrypted (and compressed) tunnel for the communication.

- Learn to use the tcpdump and strings utilities. In most cases, you can check whether MySQL data streams are unencrypted by issuing a command like the following:

  ```
  shell> tcpdump -l -i eth0 -w - src or dst port 3306 | strings
  ```

  This works under Linux and should work with small modifications under other systems.

  > **Warning**
  >
  > If you do not see cleartext data, this does not always mean that the information actually is encrypted. If you need high security, consult with a security expert.

6.1.2 Keeping Passwords Secure

Passwords occur in several contexts within MySQL. The following sections provide guidelines that enable end users and administrators to keep these passwords secure and avoid exposing them. There is also a discussion of how MySQL uses password hashing internally and of a plugin that you can use to enforce stricter passwords.

6.1.2.1 End-User Guidelines for Password Security

MySQL users should use the following guidelines to keep passwords secure.

When you run a client program to connect to the MySQL server, it is inadvisable to specify your password in a way that exposes it to discovery by other users. The methods you can use to specify your password when you run client programs are listed here, along with an assessment of the risks of each method. In short, the safest methods are to have the client program prompt for the password or to specify the password in a properly protected option file.

- Use the `mysql_config_editor` utility, which enables you to store authentication credentials in an encrypted login file named `.mylogin.cnf`. The file can be read later by MySQL client programs to obtain authentication credentials for connecting to MySQL Server. See Section 4.6.6, “`mysql_config_editor` — MySQL Configuration Utility”.

- Use a `-pyour_pass` or `--password=your_pass` option on the command line. For example:

  ```
  shell> mysql -u francis -pfrank db_name
  ```

  This is convenient *but insecure*. On some systems, your password becomes visible to system status programs such as `ps` that may be invoked by other users to display command lines. MySQL clients typically overwrite the command-line password argument with zeros during their initialization sequence. However, there is still a brief interval during which the value is visible. Also, on some systems this overwriting strategy is ineffective and the password remains visible to `ps`. (SystemV Unix systems and perhaps others are subject to this problem.)

  If your operating environment is set up to display your current command in the title bar of your terminal window, the password remains visible as long as the command is running, even if the command has scrolled out of view in the window content area.

- Use the `-p` or `--password` option on the command line with no password value specified. In this case, the client program solicits the password interactively:

  ```
  shell> mysql -u francis -p db_name
  Enter password: ********
  ```

  The “`*`” characters indicate where you enter your password. The password is not displayed as you enter it.

  It is more secure to enter your password this way than to specify it on the command line because it is not visible to other users. However, this method of entering a password is suitable only for programs that you run interactively. If you want to invoke a client from a script that runs noninteractively, there is no opportunity to enter the password from the keyboard. On some systems, you may even find that the first line of your script is read and interpreted (incorrectly) as your password.

- Store your password in an option file. For example, on Unix, you can list your password in the `[client]` section of the `.my.cnf` file in your home directory:

  ```
  [client]
  password=your_pass
  ```

  To keep the password safe, the file should not be accessible to anyone but yourself. To ensure this, set the file access mode to `400` or `600`. For example:

  ```
  shell> chmod 600 .my.cnf
  ```

  To name from the command line a specific option file containing the password, use the `--defaults-file=file_name` option, where `file_name` is the full path name to the file. For example:

```
shell> mysql --defaults-file=/home/francis/mysql-opts
```

Section 4.2.3.3, “Using Option Files”, discusses option files in more detail.

- Store your password in the MYSQL_PWD environment variable. See Section 2.11, “Environment Variables”.

  This method of specifying your MySQL password must be considered *extremely insecure* and should not be used. Some versions of ps include an option to display the environment of running processes. On some systems, if you set MYSQL_PWD, your password is exposed to any other user who runs ps. Even on systems without such a version of ps, it is unwise to assume that there are no other methods by which users can examine process environments.

On Unix, the mysql client writes a record of executed statements to a history file (see Section 4.5.1.3, “mysql Logging”). By default, this file is named .mysql_history and is created in your home directory. Passwords can be written as plain text in SQL statements such as CREATE USER, GRANT, and SET PASSWORD, so if you use these statements, they are logged in the history file. To keep this file safe, use a restrictive access mode, the same way as described earlier for the .my.cnf file.

If your command interpreter is configured to maintain a history, any file in which the commands are saved will contain MySQL passwords entered on the command line. For example, bash uses ~/.bash_history. Any such file should have a restrictive access mode.

6.1.2.2 Administrator Guidelines for Password Security

Database administrators should use the following guidelines to keep passwords secure.

MySQL stores passwords for user accounts in the mysql.user table. Access to this table should never be granted to any nonadministrative accounts.

Account passwords can be expired so that users must reset them. See Section 6.3.6, “Password Expiration Policy”, and Section 6.3.7, “Password Expiration and Sandbox Mode”.

The validate_password plugin can be used to enforce a policy on acceptable password. See Section 6.1.2.6, “The Password Validation Plugin”.

A user who has access to modify the plugin directory (the value of the plugin_dir system variable) or the my.cnf file that specifies the location of the plugin directory can replace plugins and modify the capabilities provided by plugins, including authentication plugins.

Files such as log files to which passwords might be written should be protected. See Section 6.1.2.3, “Passwords and Logging”.

6.1.2.3 Passwords and Logging

Passwords can be written as plain text in SQL statements such as CREATE USER, GRANT, and SET PASSWORD, or statements that invoke the PASSWORD() function. If these statements are logged by the MySQL server as written, such passwords become available to anyone with access to the logs.

Passwords in statements written to the general query log, slow query log, and binary log are rewritten by the server not to occur literally in plain text. Password rewriting can be suppressed for the general query log by starting the server with the --log-raw option. This option may be useful for diagnostic purposes, to see the exact text of statements as received by the server, but for security reasons is not recommended for production use.

In MySQL 5.7, statement logging avoids writing passwords in plain text for the following statements:

```
CREATE USER ... IDENTIFIED BY ...
GRANT ... IDENTIFIED BY ...
SET PASSWORD ...
SLAVE START ... PASSWORD = ...
CREATE SERVER ... OPTIONS(... PASSWORD ...)
ALTER SERVER ... OPTIONS(... PASSWORD ...)
```

Passwords in those statements are rewritten not to appear literally in statement text, for the general query log, slow query log, and binary log. Rewriting does not apply to other statements.

For the general query log, password rewriting can be suppressed by starting the server with the `--log-raw` option. This option may be useful for diagnostic purposes, to see the exact text of statements as received by the server, but for security reasons is not recommended for production use.

Contents of the audit log file produced by the audit log plugin are not encrypted. For security reasons, this file should be written to a directory accessible only to the MySQL server and users with a legitimate reason to view the log. See Section 6.3.13.2, “Audit Log Plugin Security Considerations”.

To guard log files against unwarranted exposure, they should be located in a directory that restricts access to only the server and the database administrator. If you log to tables in the `mysql` database, access to those tables should never be granted to any nonadministrative accounts.

Replication slaves store the password for the replication master in the master info repository, which can be either a file or a table (see Section 16.2.2, “Replication Relay and Status Logs”). Ensure that the repository can be accessed only by the database administrator. An alternative to storing the password in a file is to use the `START SLAVE` statement to specify credentials for connecting to the master.

Database backups that include tables or log files containing passwords should be protected using a restricted access mode.

6.1.2.4 Password Hashing in MySQL

MySQL lists user accounts in the `user` table of the `mysql` database. Each MySQL account can be assigned a password, although the `user` table does not store the cleartext version of the password, but a hash value computed from it.

MySQL uses passwords in two phases of client/server communication:

- When a client attempts to connect to the server, there is an initial authentication step in which the client must present a password that has a hash value matching the hash value stored in the `user` table for the account the client wants to use.

- After the client connects, it can (if it has sufficient privileges) set or change the password hash for accounts listed in the `user` table. The client can do this by using the `PASSWORD()` function to generate a password hash, or by using a password-generating statement (`CREATE USER`, `GRANT`, or `SET PASSWORD`).

In other words, the server *checks* hash values during authentication when a client first attempts to connect. The server *generates* hash values if a connected client invokes the `PASSWORD()` function or uses a password-generating statement to set or change a password.

Password hashing methods in MySQL have the history described following. These changes are illustrated by changes in the result from the `PASSWORD()` function that computes password hash values and in the structure of the `user` table where passwords are stored.

The Original (Pre-4.1) Hashing Method

The original hashing method produced a 16-byte string. Such hashes look like this:

```
mysql> SELECT PASSWORD('mypass');
+--------------------+
| PASSWORD('mypass') |
+--------------------+
| 6f8c114b58f2ce9e   |
+--------------------+
```

To store account passwords, the Password column of the user table was at this point 16 bytes long.

The 4.1 Hashing Method

MySQL 4.1 introduced password hashing that provides better security and reduces the risk of passwords being intercepted. There were several aspects to this change:

- Different PASSWORD() function result format
- Widening of the Password column
- Control over the default hashing method
- Control over the permitted hashing methods for clients attempting to connect to the server

The changes in MySQL 4.1 took place in two stages:

- MySQL 4.1.0 used a preliminary version of the 4.1 hashing method. Because this method was so short lived, the following discussion says no more about it.
- In MySQL 4.1.1, the hashing method was modified to produce a longer 41-byte hash value:

  ```
  mysql> SELECT PASSWORD('mypass');
  +-------------------------------------------+
  | PASSWORD('mypass')                        |
  +-------------------------------------------+
  | *6C8989366EAF75BB670AD8EA7A7FC1176A95CEF4 |
  +-------------------------------------------+
  ```

  The longer password hash format has better cryptographic properties, and client authentication based on long hashes is more secure than that based on the older short hashes.

  To accommodate longer password hashes, the Password column in the user table was changed at this point to be 41 bytes, its current length.

  A widened Password column can store password hashes in both the pre-4.1 and 4.1 formats. The format of any given hash value can be determined two ways:

  - The length: 4.1 and pre-4.1 hashes are 41 and 16 bytes, respectively.
  - Password hashes in the 4.1 format always begin with a “*” character, whereas passwords in the pre-4.1 format never do.

  To permit explicit generation of pre-4.1 password hashes, two additional changes were made:

  - The OLD_PASSWORD() function was added, which returns hash values in the 16-byte format.
  - For compatibility purposes, the old_passwords system variable was added, to enable DBAs and applications control over the hashing method. The default old_passwords value of 0 causes hashing to use the 4.1 method (41-byte hash values), but setting old_passwords=1 causes hashing to use the pre-4.1 method. In this case, PASSWORD() produces 16-byte values and is equivalent to OLD_PASSWORD()

To permit DBAs control over how clients are permitted to connect, the secure_auth system variable was added. Starting the server with this variable disabled or enabled permits or prohibits clients to connect using the older pre-4.1 password hashing method. Before MySQL 5.6.5, secure_auth is disabled by default. As of 5.6.5, secure_auth is enabled by default to promote a more secure default configuration DBAs can disable it at their discretion, but this is not recommended), and pre-4.1 password hashes are deprecated and should be avoided. (For account upgrade instructions, see Section 6.3.9.3, “Migrating Away from Pre-4.1 Password Hashing and the mysql_old_password Plugin”.)

In addition, the mysql client supports a --secure-auth option that is analogous to secure_auth, but from the client side. It can be used to prevent connections to less secure accounts that use pre-4.1 password hashing. This option is disabled by default before MySQL 5.6.7, enabled thereafter.

Compatibility Issues Related to Hashing Methods

The widening of the Password column in MySQL 4.1 from 16 bytes to 41 bytes affects installation or upgrade operations as follows:

- If you perform a new installation of MySQL, the Password column is made 41 bytes long automatically.
- Upgrades from MySQL 4.1 or later to current versions of MySQL should not give rise to any issues in regard to the Password column because both versions use the same column length and password hashing method.
- For upgrades from a pre-4.1 release to 4.1 or later, you must upgrade the system tables after upgrading. (See Section 4.4.7, “mysql_upgrade — Check and Upgrade MySQL Tables”.)

The 4.1 hashing method is understood only by MySQL 4.1 (and newer) servers and clients, which can result in some compatibility problems. A 4.1 or newer client can connect to a pre-4.1 server, because the client understands both the pre-4.1 and 4.1 password hashing methods. However, a pre-4.1 client that attempts to connect to a 4.1 or newer server may run into difficulties. For example, a 4.0 mysql client may fail with the following error message:

```
shell> mysql -h localhost -u root
Client does not support authentication protocol requested
by server; consider upgrading MySQL client
```

This phenomenon also occurs for attempts to use the older PHP mysql extension after upgrading to MySQL 4.1 or newer. (See Common Problems with MySQL and PHP.)

The following discussion describes the differences between the pre-4.1 and 4.1 hashing methods, and what you should do if you upgrade your server but need to maintain backward compatibility with pre-4.1 clients. (However, permitting connections by old clients is not recommended and should be avoided if possible.) Additional information can be found in Section C.5.2.4, “Client does not support authentication protocol”. This information is of particular importance to PHP programmers migrating MySQL databases from versions older than 4.1 to 4.1 or higher.

The differences between short and long password hashes are relevant both for how the server uses passwords during authentication and for how it generates password hashes for connected clients that perform password-changing operations.

The way in which the server uses password hashes during authentication is affected by the width of the Password column:

- If the column is short, only short-hash authentication is used.
- If the column is long, it can hold either short or long hashes, and the server can use either format:

- Pre-4.1 clients can connect, but because they know only about the pre-4.1 hashing method, they can authenticate only using accounts that have short hashes.
- 4.1 and later clients can authenticate using accounts that have short or long hashes.

Even for short-hash accounts, the authentication process is actually a bit more secure for 4.1 and later clients than for older clients. In terms of security, the gradient from least to most secure is:

- Pre-4.1 client authenticating with short password hash
- 4.1 or later client authenticating with short password hash
- 4.1 or later client authenticating with long password hash

The way in which the server generates password hashes for connected clients is affected by the width of the `Password` column and by the `old_passwords` system variable. A 4.1 or later server generates long hashes only if certain conditions are met: The `Password` column must be wide enough to hold long values and `old_passwords` must not be set to 1.

Those conditions apply as follows:

- The `Password` column must be wide enough to hold long hashes (41 bytes). If the column has not been updated and still has the pre-4.1 width of 16 bytes, the server notices that long hashes cannot fit into it and generates only short hashes when a client performs password-changing operations using the `PASSWORD()` function or a password-generating statement. This is the behavior that occurs if you have upgraded from a version of MySQL older than 4.1 to 4.1 or later but have not yet run the `mysql_upgrade` program to widen the `Password` column.
- If the `Password` column is wide, it can store either short or long password hashes. In this case, the `PASSWORD()` function and password-generating statements generate long hashes unless the server was started with the `old_passwords` system variable set to 1 to force the server to generate short password hashes instead.

The purpose of the `old_passwords` system variable is to permit backward compatibility with pre-4.1 clients under circumstances where the server would otherwise generate long password hashes. The option does not affect authentication (4.1 and later clients can still use accounts that have long password hashes), but it does prevent creation of a long password hash in the `user` table as the result of a password-changing operation. Were that permitted to occur, the account could no longer be used by pre-4.1 clients. With `old_passwords` disabled, the following undesirable scenario is possible:

- An old pre-4.1 client connects to an account that has a short password hash.
- The client changes its own password. With `old_passwords` disabled, this results in the account having a long password hash.
- The next time the old client attempts to connect to the account, it cannot, because the account has a long password hash that requires the 4.1 hashing method during authentication. (Once an account has a long password hash in the user table, only 4.1 and later clients can authenticate for it because pre-4.1 clients do not understand long hashes.)

This scenario illustrates that, if you must support older pre-4.1 clients, it is problematic to run a 4.1 or newer server without `old_passwords` set to 1. By running the server with `old_passwords=1`, password-changing operations do not generate long password hashes and thus do not cause accounts to become inaccessible to older clients. (Those clients cannot inadvertently lock themselves out by changing their password and ending up with a long password hash.)

The downside of old_passwords=1 is that any passwords created or changed use short hashes, even for 4.1 or later clients. Thus, you lose the additional security provided by long password hashes. To create an account that has a long hash (for example, for use by 4.1 clients) or to change an existing account to use a long password hash, an administrator can set the session value of old_passwords set to 0 while leaving the global value set to 1:

```
mysql> SET @@session.old_passwords = 0;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @@session.old_passwords, @@global.old_passwords;
+-------------------------+------------------------+
| @@session.old_passwords | @@global.old_passwords |
+-------------------------+------------------------+
|                       0 |                      1 |
+-------------------------+------------------------+
1 row in set (0.00 sec)

mysql> CREATE USER 'newuser'@'localhost' IDENTIFIED BY 'newpass';
Query OK, 0 rows affected (0.03 sec)

mysql> SET PASSWORD FOR 'existinguser'@'localhost' = PASSWORD('existingpass');
Query OK, 0 rows affected (0.00 sec)
```

The following scenarios are possible in MySQL 4.1 or later. The factors are whether the Password column is short or long, and, if long, whether the server is started with old_passwords enabled or disabled.

Scenario 1: Short Password column in user table:

- Only short hashes can be stored in the Password column.
- The server uses only short hashes during client authentication.
- For connected clients, password hash-generating operations involving the PASSWORD() function or password-generating statements use short hashes exclusively. Any change to an account's password results in that account having a short password hash.
- The value of old_passwords is irrelevant because with a short Password column, the server generates only short password hashes anyway.

This scenario occurs when a pre-4.1 MySQL installation has been upgraded to 4.1 or later but mysql_upgrade has not been run to upgrade the system tables in the mysql database. (This is not a recommended configuration because it does not permit use of more secure 4.1 password hashing.)

Scenario 2: Long Password column; server started with old_passwords=1:

- Short or long hashes can be stored in the Password column.
- 4.1 and later clients can authenticate for accounts that have short or long hashes.
- Pre-4.1 clients can authenticate only for accounts that have short hashes.
- For connected clients, password hash-generating operations involving the PASSWORD() function or password-generating statements use short hashes exclusively. Any change to an account's password results in that account having a short password hash.

In this scenario, newly created accounts have short password hashes because old_passwords=1 prevents generation of long hashes. Also, if you create an account with a long hash before setting old_passwords to 1, changing the account's password while old_passwords=1 results in the account being given a short password, causing it to lose the security benefits of a longer hash.

To create a new account that has a long password hash, or to change the password of any existing account to use a long hash, first set the session value of `old_passwords` set to 0 while leaving the global value set to 1, as described previously.

In this scenario, the server has an up to date `Password` column, but is running with the default password hashing method set to generate pre-4.1 hash values. This is not a recommended configuration but may be useful during a transitional period in which pre-4.1 clients and passwords are upgraded to 4.1 or later. When that has been done, it is preferable to run the server with `old_passwords=0` and `secure_auth=1`.

Scenario 3: Long `Password` column; server started with `old_passwords=0`:

- Short or long hashes can be stored in the `Password` column.
- 4.1 and later clients can authenticate using accounts that have short or long hashes.
- Pre-4.1 clients can authenticate only using accounts that have short hashes.
- For connected clients, password hash-generating operations involving the `PASSWORD()` function or password-generating statements use long hashes exclusively. A change to an account's password results in that account having a long password hash.

As indicated earlier, a danger in this scenario is that it is possible for accounts that have a short password hash to become inaccessible to pre-4.1 clients. A change to such an account's password made using the `PASSWORD()` function or a password-generating statement results in the account being given a long password hash. From that point on, no pre-4.1 client can connect to the server using that account. The client must upgrade to 4.1 or later.

If this is a problem, you can change a password in a special way. For example, normally you use `SET PASSWORD` as follows to change an account password:

```
SET PASSWORD FOR 'some_user'@'some_host' = PASSWORD('mypass');
```

To change the password but create a short hash, use the `OLD_PASSWORD()` function instead:

```
SET PASSWORD FOR 'some_user'@'some_host' = OLD_PASSWORD('mypass');
```

`OLD_PASSWORD()` is useful for situations in which you explicitly want to generate a short hash.

The disadvantages for each of the preceding scenarios may be summarized as follows:

In scenario 1, you cannot take advantage of longer hashes that provide more secure authentication.

In scenario 2, `old_passwords=1` prevents accounts with short hashes from becoming inaccessible, but password-changing operations cause accounts with long hashes to revert to short hashes unless you take care to change the session value of `old_passwords` to 0 first.

In scenario 3, accounts with short hashes become inaccessible to pre-4.1 clients if you change their passwords without explicitly using `OLD_PASSWORD()`.

The best way to avoid compatibility problems related to short password hashes is to not use them:

- Upgrade all client programs to MySQL 4.1 or later.
- Run the server with `old_passwords=0`.
- Reset the password for any account with a short password hash to use a long password hash.
- For additional security, run the server with `secure_auth=1`.

6.1.2.5 Implications of Password Hashing Changes in MySQL 4.1 for Application Programs

An upgrade to MySQL version 4.1 or later can cause compatibility issues for applications that use PASSWORD() to generate passwords for their own purposes. Applications really should not do this, because PASSWORD() should be used only to manage passwords for MySQL accounts. But some applications use PASSWORD() for their own purposes anyway.

If you upgrade to 4.1 or later from a pre-4.1 version of MySQL and run the server under conditions where it generates long password hashes, an application using PASSWORD() for its own passwords breaks. The recommended course of action in such cases is to modify the application to use another function, such as SHA2(), SHA1(), or MD5(), to produce hashed values. If that is not possible, you can use the OLD_PASSWORD() function, which is provided for generate short hashes in the old format. However, you should note that OLD_PASSWORD() may one day no longer be supported.

If the server is running with old_passwords=1, it generates short hashes and OLD_PASSWORD() is equivalent to PASSWORD().

PHP programmers migrating their MySQL databases from version 4.0 or lower to version 4.1 or higher should see MySQL and PHP.

6.1.2.6 The Password Validation Plugin

The validate_password plugin can be used to test passwords and improve security. This plugin implements two capabilities:

- In statements that assign a password supplied as a cleartext value, the value is checked against the current password policy and rejected if it is weak (the statement returns an ER_NOT_VALID_PASSWORD error). This affects the CREATE USER, GRANT, and SET PASSWORD statements. Passwords given as arguments to the PASSWORD() and OLD_PASSWORD() functions are checked as well.
- The strength of potential passwords can be assessed using the VALIDATE_PASSWORD_STRENGTH() SQL function, which takes a password argument and returns an integer from 0 (weak) to 100 (strong).

For example, the cleartext password in the following statement is checked. Under the default password policy, which requires passwords to be at least 8 characters long, the password is weak and the statement produces an error:

```
mysql> SET PASSWORD = PASSWORD('abc');
ERROR 1819 (HY000): Your password does not satisfy the current policy
requirements
```

Passwords specified as already hashed values are not checked because the original password value is not available:

```
mysql> SET PASSWORD = '*0D3CED9BEC10A777AEC23CCC353A8C08A633045E';
Query OK, 0 rows affected (0.01 sec)
```

The parameters that control password checking are available as the values of the system variables having names of the form validate_password_*xxx*. These variables can be modified to configure password checking; see Password Validation Plugin Options and Variables.

The three levels of password checking are LOW, MEDIUM, and STRONG. The default is MEDIUM; to change this, modify the value of validate_password_policy. The policies implement increasingly strict password tests. The following descriptions refer to default parameter values; these can be modified by changing the appropriate system variables.

- LOW policy tests password length only. Passwords must be at least 8 characters long.

- MEDIUM policy adds the conditions that passwords must contain at least 1 numeric character, 1 lowercase and uppercase character, and 1 special (nonalphanumeric) character.
- STRONG policy adds the condition that password substrings of length 4 or longer must not match words in the dictionary file, if one has been specified.

If the validate_password plugin is not installed, the validate_password_*xxx* system variables are not available, passwords in statements are not checked, and VALIDATE_PASSWORD_STRENGTH() always returns 0. For example, accounts can be assigned passwords shorter than 8 characters.

Password Validation Plugin Installation

The password-validation plugin is named validate_password. To be usable by the server, the plugin library object file must be located in the MySQL plugin directory (the directory named by the plugin_dir system variable). If necessary, set the value of plugin_dir at server startup to tell the server the location of the plugin directory.

To load the plugin at server startup, use the --plugin-load option to name the object file that contains the plugin. With this plugin-loading method, the option must be given each time the server starts. For example, put these lines in your my.cnf file:

```
[mysqld]
plugin-load=validate_password.so
```

If object files have a suffix different from .so on your system, substitute the correct suffix (for example, .dll on Windows).

Alternatively, to register the plugin at runtime, use this statement (changing the extension as necessary):

```
mysql> INSTALL PLUGIN validate_password SONAME 'validate_password.so';
```

INSTALL PLUGIN loads the plugin, and also registers it in the mysql.plugins table to cause the plugin to be loaded for each subsequent normal server startup.

If the plugin has been previously registered with INSTALL PLUGIN or is loaded with --plugin-load, you can use the --validate-password option at server startup to control plugin activation. For example, to load the plugin and prevent it from being removed at runtime, use these options:

```
[mysqld]
plugin-load=validate_password.so
validate-password=FORCE_PLUS_PERMANENT
```

If it is desired to prevent the server from running without the password-validation plugin, use --validate-password with a value of FORCE or FORCE_PLUS_PERMANENT to force server startup to fail if the plugin does not initialize successfully.

For general information about installing plugins, see Section 5.1.8, “Server Plugins”. To verify plugin installation, examine the INFORMATION_SCHEMA.PLUGINS table or use the SHOW PLUGINS statement. See Section 5.1.8.2, “Obtaining Server Plugin Information”.

Password Validation Plugin Options and Variables

To control the activation of the validate_password plugin, use this option:

- --validate-password[=*value*]

Command-Line Format	--validate-password[=value]

Option-File Format	validate-password	
	Permitted Values	
	Type	enumeration
	Default	ON
	Valid Values	ON
		OFF
		FORCE
		FORCE_PLUS_PERMANENT

This option controls how the server loads the validate_password plugin at startup. The value should be one of those available for plugin-loading options, as described in Section 5.1.8.1, “Installing and Uninstalling Plugins”. For example, --validate-password=FORCE_PLUS_PERMANENT tells the server to load the plugin and prevent it from being removed while the server is running.

This option is available only if the validate_password plugin has been previously registered with INSTALL PLUGIN or is loaded with --plugin-load. See Password Validation Plugin Installation.

If the validate_password plugin is installed, it exposes several system variables that indicate the parameters that control password checking:

```
mysql> SHOW VARIABLES LIKE 'validate_password%';
+--------------------------------------+--------+
| Variable_name                        | Value  |
+--------------------------------------+--------+
| validate_password_dictionary_file    |        |
| validate_password_length             | 8      |
| validate_password_mixed_case_count   | 1      |
| validate_password_number_count       | 1      |
| validate_password_policy             | MEDIUM |
| validate_password_special_char_count | 1      |
+--------------------------------------+--------+
```

To change how passwords are checked, you can set any of these variables at server startup, and most of them at runtime. The following list describes the meaning of each variable.

- validate_password_dictionary_file

System Variable Name	validate_password_dictionary_file	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	file name

The path name of the dictionary file used by the validate_password plugin for checking passwords. This variable is unavailable unless that plugin is installed.

By default, this variable has an empty value and dictionary checks are not performed. To enable dictionary checks, you must set this variable to a nonempty value. If the file is named as a relative path, it is interpreted relative to the server data directory. Its contents should be lowercase, one word per line. Contents are treated as having a character set of utf8. The maximum permitted file size is 1MB.

For the dictionary file to be used during password checking, the password policy must be set to 2 (STRONG); see the description of the validate_password_policy system variable. Assuming that is true, each substring of the password of length 4 up to 100 is compared to the words in the dictionary file. Any match causes the password to be rejected. Comparisons are not case sensitive.

For VALIDATE_PASSWORD_STRENGTH() the password is checked against all policies, including STRONG, so the strength assessment includes the dictionary check regardless of the validate_password_policy value.

Changes to the dictionary file while the server is running require a restart for the server to recognize the changes.

- validate_password_length

System Variable Name	validate_password_length	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	numeric
	Default	8
	Min Value	0

The minimum number of characters that passwords checked by the validate_password plugin must have. This variable is unavailable unless that plugin is installed.

The validate_password_length minimum value is a function of several other related system variables. The server will not set the value less than the value of this expression:

```
validate_password_number_count
+ validate_password_special_char_count
+ (2 * validate_password_mixed_case_count)
```

If the validate_password plugin adjusts the value of validate_password_length due to the preceding constraint, it writes a message to the error log.

- validate_password_mixed_case_count

System Variable Name	validate_password_mixed_case_count	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	numeric
	Default	1
	Min Value	0

The minimum number of lowercase and uppercase characters that passwords checked by the validate_password plugin must have if the password policy is MEDIUM or stronger. This variable is unavailable unless that plugin is installed.

- validate_password_number_count

System Variable Name	validate_password_number_count	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	numeric
	Default	1
	Min Value	0

The minimum number of numeric (digit) characters that passwords checked by the validate_password plugin must have if the password policy is MEDIUM or stronger. This variable is unavailable unless that plugin is installed.

- validate_password_policy

System Variable Name	validate_password_policy	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	enumeration
	Default	1
	Valid Values	0
		1
		2

The password policy enforced by the validate_password plugin. This variable is unavailable unless that plugin is installed.

The validate_password_policy value can be specified using numeric values 0, 1, 2, or the corresponding symbolic values LOW, MEDIUM, STRONG. The following table describes the tests performed for each policy. For the length test, the required length is the value of the validate_password_length system variable. Similarly, the required values for the other tests are given by other validate_password_*xxx* variables.

Policy	Tests Performed
0 or LOW	Length
1 or MEDIUM	Length; numeric, lowercase/uppercase, and special characters
2 or STRONG	Length; numeric, lowercase/uppercase, and special characters; dictionary file

- validate_password_special_char_count

<table>
<tr><td><b>System Variable Name</b></td><td colspan="2"><code>validate_password_special_char_count</code></td></tr>
<tr><td><b>Variable Scope</b></td><td colspan="2">Global</td></tr>
<tr><td><b>Dynamic Variable</b></td><td colspan="2">Yes</td></tr>
<tr><td rowspan="4"></td><td colspan="2"><b>Permitted Values</b></td></tr>
<tr><td><b>Type</b></td><td><code>numeric</code></td></tr>
<tr><td><b>Default</b></td><td><code>1</code></td></tr>
<tr><td><b>Min Value</b></td><td><code>0</code></td></tr>
</table>

The minimum number of nonalphanumeric characters that passwords checked by the `validate_password` plugin must have if the password policy is `MEDIUM` or stronger. This variable is unavailable unless that plugin is installed.

6.1.3 Making MySQL Secure Against Attackers

When you connect to a MySQL server, you should use a password. The password is not transmitted in clear text over the connection. Password handling during the client connection sequence was upgraded in MySQL 4.1.1 to be very secure. If you are still using pre-4.1.1-style passwords, the encryption algorithm is not as strong as the newer algorithm. With some effort, a clever attacker who can sniff the traffic between the client and the server can crack the password. (See Section 6.1.2.4, “Password Hashing in MySQL”, for a discussion of the different password handling methods.)

All other information is transferred as text, and can be read by anyone who is able to watch the connection. If the connection between the client and the server goes through an untrusted network, and you are concerned about this, you can use the compressed protocol to make traffic much more difficult to decipher. You can also use MySQL's internal SSL support to make the connection even more secure. See Section 6.3.11, “Using SSL for Secure Connections”. Alternatively, use SSH to get an encrypted TCP/IP connection between a MySQL server and a MySQL client. You can find an Open Source SSH client at http://www.openssh.org/, and a commercial SSH client at http://www.ssh.com/.

To make a MySQL system secure, you should strongly consider the following suggestions:

- Require all MySQL accounts to have a password. A client program does not necessarily know the identity of the person running it. It is common for client/server applications that the user can specify any user name to the client program. For example, anyone can use the `mysql` program to connect as any other person simply by invoking it as `mysql -u` *`other_user db_name`* if *`other_user`* has no password. If all accounts have a password, connecting using another user's account becomes much more difficult.

  For a discussion of methods for setting passwords, see Section 6.3.5, “Assigning Account Passwords”.

- Make sure that the only Unix user account with read or write privileges in the database directories is the account that is used for running `mysqld`.

- Never run the MySQL server as the Unix `root` user. This is extremely dangerous, because any user with the `FILE` privilege is able to cause the server to create files as `root` (for example, `~root/.bashrc`). To prevent this, `mysqld` refuses to run as `root` unless that is specified explicitly using the `--user=root` option.

  `mysqld` can (and should) be run as an ordinary, unprivileged user instead. You can create a separate Unix account named `mysql` to make everything even more secure. Use this account only for administering MySQL. To start `mysqld` as a different Unix user, add a `user` option that specifies the user name in the [mysqld] group of the my.cnf option file where you specify server options. For example:

```
[mysqld]
user=mysql
```

This causes the server to start as the designated user whether you start it manually or by using mysqld_safe or mysql.server. For more details, see Section 6.1.5, “How to Run MySQL as a Normal User”.

Running mysqld as a Unix user other than root does not mean that you need to change the root user name in the user table. *User names for MySQL accounts have nothing to do with user names for Unix accounts*.

- Do not grant the FILE privilege to nonadministrative users. Any user that has this privilege can write a file anywhere in the file system with the privileges of the mysqld daemon. This includes the server's data directory containing the files that implement the privilege tables. To make FILE-privilege operations a bit safer, files generated with SELECT ... INTO OUTFILE do not overwrite existing files and are writable by everyone.

  The FILE privilege may also be used to read any file that is world-readable or accessible to the Unix user that the server runs as. With this privilege, you can read any file into a database table. This could be abused, for example, by using LOAD DATA to load /etc/passwd into a table, which then can be displayed with SELECT.

  To limit the location in which files can be read and written, set the secure_file_priv system to a specific directory. See Section 5.1.4, “Server System Variables”.

- Do not grant the PROCESS or SUPER privilege to nonadministrative users. The output of mysqladmin processlist and SHOW PROCESSLIST shows the text of any statements currently being executed, so any user who is permitted to see the server process list might be able to see statements issued by other users such as UPDATE user SET password=PASSWORD('not_secure').

  mysqld reserves an extra connection for users who have the SUPER privilege, so that a MySQL root user can log in and check server activity even if all normal connections are in use.

  The SUPER privilege can be used to terminate client connections, change server operation by changing the value of system variables, and control replication servers.

- Do not permit the use of symlinks to tables. (This capability can be disabled with the --skip-symbolic-links option.) This is especially important if you run mysqld as root, because anyone that has write access to the server's data directory then could delete any file in the system! See Using Symbolic Links for MyISAM Tables on Unix.

- Stored programs and views should be written using the security guidelines discussed in Section 18.6, “Access Control for Stored Programs and Views”.

- If you do not trust your DNS, you should use IP addresses rather than host names in the grant tables. In any case, you should be very careful about creating grant table entries using host name values that contain wildcards.

- If you want to restrict the number of connections permitted to a single account, you can do so by setting the max_user_connections variable in mysqld. The GRANT statement also supports resource control options for limiting the extent of server use permitted to an account. See Section 13.7.1.4, “GRANT Syntax”.

- If the plugin directory is writable by the server, it may be possible for a user to write executable code to a file in the directory using `SELECT ... INTO DUMPFILE`. This can be prevented by making `plugin_dir` read only to the server or by setting `--secure-file-priv` to a directory where `SELECT` writes can be made safely.

6.1.4 Security-Related `mysqld` Options and Variables

The following table shows `mysqld` options and system variables that affect security. For descriptions of each of these, see Section 5.1.3, “Server Command Options”, and Section 5.1.4, “Server System Variables”.

Table 6.1 Security Option/Variable Summary

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
allow-suspicious-udfs	Yes	Yes				
automatic_sp_privileges			Yes		Global	Yes
chroot	Yes	Yes				
des-key-file	Yes	Yes				
local_infile			Yes		Global	Yes
old_passwords			Yes		Both	Yes
safe-user-create	Yes	Yes				
secure-auth	Yes	Yes			Global	Yes
- *Variable*: secure_auth			Yes		Global	Yes
secure-file-priv	Yes	Yes			Global	No
- *Variable*: secure_file_priv			Yes		Global	No
skip-grant-tables	Yes	Yes				
skip-name-resolve	Yes	Yes			Global	No
- *Variable*: skip_name_resolve			Yes		Global	No
skip-networking	Yes	Yes			Global	No
- *Variable*: skip_networking			Yes		Global	No
skip-show-database	Yes	Yes			Global	No
- *Variable*: skip_show_database			Yes		Global	No

6.1.5 How to Run MySQL as a Normal User

On Windows, you can run the server as a Windows service using a normal user account.

On Unix, the MySQL server `mysqld` can be started and run by any user. However, you should avoid running the server as the Unix `root` user for security reasons. To change `mysqld` to run as a normal unprivileged Unix user *`user_name`*, you must do the following:

1. Stop the server if it is running (use `mysqladmin shutdown`).

2. Change the database directories and files so that *user_name* has privileges to read and write files in them (you might need to do this as the Unix root user):

   ```
   shell> chown -R user_name /path/to/mysql/datadir
   ```

   If you do not do this, the server will not be able to access databases or tables when it runs as *user_name*.

   If directories or files within the MySQL data directory are symbolic links, chown -R might not follow symbolic links for you. If it does not, you will also need to follow those links and change the directories and files they point to.

3. Start the server as user *user_name*. Another alternative is to start mysqld as the Unix root user and use the --user=*user_name* option. mysqld starts up, then switches to run as the Unix user *user_name* before accepting any connections.

4. To start the server as the given user automatically at system startup time, specify the user name by adding a user option to the [mysqld] group of the /etc/my.cnf option file or the my.cnf option file in the server's data directory. For example:

   ```
   [mysqld]
   user=user_name
   ```

If your Unix machine itself is not secured, you should assign passwords to the MySQL root accounts in the grant tables. Otherwise, any user with a login account on that machine can run the mysql client with a --user=root option and perform any operation. (It is a good idea to assign passwords to MySQL accounts in any case, but especially so when other login accounts exist on the server host.) See Section 2.9.2, “Securing the Initial MySQL Accounts”.

6.1.6 Security Issues with LOAD DATA LOCAL

The LOAD DATA statement can load a file that is located on the server host, or it can load a file that is located on the client host when the LOCAL keyword is specified.

There are two potential security issues with supporting the LOCAL version of LOAD DATA statements:

- The transfer of the file from the client host to the server host is initiated by the MySQL server. In theory, a patched server could be built that would tell the client program to transfer a file of the server's choosing rather than the file named by the client in the LOAD DATA statement. Such a server could access any file on the client host to which the client user has read access.

- In a Web environment where the clients are connecting from a Web server, a user could use LOAD DATA LOCAL to read any files that the Web server process has read access to (assuming that a user could run any command against the SQL server). In this environment, the client with respect to the MySQL server actually is the Web server, not the remote program being run by the user who connects to the Web server.

To deal with these problems, we changed how LOAD DATA LOCAL is handled as of MySQL 3.23.49 and MySQL 4.0.2 (4.0.13 on Windows):

- By default, all MySQL clients and libraries in binary distributions are compiled with the -DENABLED_LOCAL_INFILE=1 option, to be compatible with MySQL 3.23.48 and before.

- If you build MySQL from source but do not invoke CMake with the -DENABLED_LOCAL_INFILE=1 option, LOAD DATA LOCAL cannot be used by any client unless it is written explicitly to invoke mysql_options(... MYSQL_OPT_LOCAL_INFILE, 0). See Section 21.8.7.50, “mysql_options()”.

- You can disable all LOAD DATA LOCAL statements from the server side by starting mysqld with the --local-infile=0 option.

- For the mysql command-line client, enable LOAD DATA LOCAL by specifying the --local-infile[=1] option, or disable it with the --local-infile=0 option. For mysqlimport, local data file loading is off by default; enable it with the --local or -L option. In any case, successful use of a local load operation requires that the server permits it.

- If you use LOAD DATA LOCAL in Perl scripts or other programs that read the [client] group from option files, you can add the local-infile=1 option to that group. However, to keep this from causing problems for programs that do not understand local-infile, specify it using the loose- prefix:

```
[client]
loose-local-infile=1
```

- If LOAD DATA LOCAL is disabled, either in the server or the client, a client that attempts to issue such a statement receives the following error message:

```
ERROR 1148: The used command is not allowed with this MySQL version
```

6.1.7 Client Programming Security Guidelines

Applications that access MySQL should not trust any data entered by users, who can try to trick your code by entering special or escaped character sequences in Web forms, URLs, or whatever application you have built. Be sure that your application remains secure if a user enters something like “; DROP DATABASE mysql;”. This is an extreme example, but large security leaks and data loss might occur as a result of hackers using similar techniques, if you do not prepare for them.

A common mistake is to protect only string data values. Remember to check numeric data as well. If an application generates a query such as SELECT * FROM table WHERE ID=234 when a user enters the value 234, the user can enter the value 234 OR 1=1 to cause the application to generate the query SELECT * FROM table WHERE ID=234 OR 1=1. As a result, the server retrieves every row in the table. This exposes every row and causes excessive server load. The simplest way to protect from this type of attack is to use single quotation marks around the numeric constants: SELECT * FROM table WHERE ID='234'. If the user enters extra information, it all becomes part of the string. In a numeric context, MySQL automatically converts this string to a number and strips any trailing nonnumeric characters from it.

Sometimes people think that if a database contains only publicly available data, it need not be protected. This is incorrect. Even if it is permissible to display any row in the database, you should still protect against denial of service attacks (for example, those that are based on the technique in the preceding paragraph that causes the server to waste resources). Otherwise, your server becomes unresponsive to legitimate users.

Checklist:

- Enable strict SQL mode to tell the server to be more restrictive of what data values it accepts. See Section 5.1.7, “Server SQL Modes”.

- Try to enter single and double quotation marks (“'” and “"”) in all of your Web forms. If you get any kind of MySQL error, investigate the problem right away.

- Try to modify dynamic URLs by adding %22 (“"”), %23 (“#”), and %27 (“'”) to them.

- Try to modify data types in dynamic URLs from numeric to character types using the characters shown in the previous examples. Your application should be safe against these and similar attacks.

- Try to enter characters, spaces, and special symbols rather than numbers in numeric fields. Your application should remove them before passing them to MySQL or else generate an error. Passing unchecked values to MySQL is very dangerous!

- Check the size of data before passing it to MySQL.

- Have your application connect to the database using a user name different from the one you use for administrative purposes. Do not give your applications any access privileges they do not need.

Many application programming interfaces provide a means of escaping special characters in data values. Properly used, this prevents application users from entering values that cause the application to generate statements that have a different effect than you intend:

- MySQL C API: Use the mysql_real_escape_string() API call.

- MySQL++: Use the escape and quote modifiers for query streams.

- PHP: Use either the mysqli or pdo_mysql extensions, and not the older ext/mysql extension. The preferred API's support the improved MySQL authentication protocol and passwords, as well as prepared statements with placeholders. See also Choosing an API.

  If the older ext/mysql extension must be used, then for escaping use the mysql_real_escape_string() function and not mysql_escape_string() or addslashes() because only mysql_real_escape_string() is character set-aware; the other functions can be “bypassed” when using (invalid) multi-byte character sets.

- Perl DBI: Use placeholders or the quote() method.

- Ruby DBI: Use placeholders or the quote() method.

- Java JDBC: Use a PreparedStatement object and placeholders.

Other programming interfaces might have similar capabilities.

6.2 The MySQL Access Privilege System

The primary function of the MySQL privilege system is to authenticate a user who connects from a given host and to associate that user with privileges on a database such as SELECT, INSERT, UPDATE, and DELETE. Additional functionality includes the ability to have anonymous users and to grant privileges for MySQL-specific functions such as LOAD DATA INFILE and administrative operations.

There are some things that you cannot do with the MySQL privilege system:

- You cannot explicitly specify that a given user should be denied access. That is, you cannot explicitly match a user and then refuse the connection.

- You cannot specify that a user has privileges to create or drop tables in a database but not to create or drop the database itself.

- A password applies globally to an account. You cannot associate a password with a specific object such as a database, table, or routine.

The user interface to the MySQL privilege system consists of SQL statements such as CREATE USER, GRANT, and REVOKE. See Section 13.7.1, "Account Management Statements".

Internally, the server stores privilege information in the grant tables of the mysql database (that is, in the database named mysql). The MySQL server reads the contents of these tables into memory when it starts and bases access-control decisions on the in-memory copies of the grant tables.

The MySQL privilege system ensures that all users may perform only the operations permitted to them. As a user, when you connect to a MySQL server, your identity is determined by *the host from which you connect* and *the user name you specify*. When you issue requests after connecting, the system grants privileges according to your identity and *what you want to do*.

MySQL considers both your host name and user name in identifying you because there is no reason to assume that a given user name belongs to the same person on all hosts. For example, the user joe who connects from office.example.com need not be the same person as the user joe who connects from home.example.com. MySQL handles this by enabling you to distinguish users on different hosts that happen to have the same name: You can grant one set of privileges for connections by joe from office.example.com, and a different set of privileges for connections by joe from home.example.com. To see what privileges a given account has, use the SHOW GRANTS statement. For example:

```
SHOW GRANTS FOR 'joe'@'office.example.com';
SHOW GRANTS FOR 'joe'@'home.example.com';
```

MySQL access control involves two stages when you run a client program that connects to the server:

Stage 1: The server accepts or rejects the connection based on your identity and whether you can verify your identity by supplying the correct password.

Stage 2: Assuming that you can connect, the server checks each statement you issue to determine whether you have sufficient privileges to perform it. For example, if you try to select rows from a table in a database or drop a table from the database, the server verifies that you have the SELECT privilege for the table or the DROP privilege for the database.

For a more detailed description of what happens during each stage, see Section 6.2.4, "Access Control, Stage 1: Connection Verification", and Section 6.2.5, "Access Control, Stage 2: Request Verification".

If your privileges are changed (either by yourself or someone else) while you are connected, those changes do not necessarily take effect immediately for the next statement that you issue. For details about the conditions under which the server reloads the grant tables, see Section 6.2.6, "When Privilege Changes Take Effect".

For general security-related advice, see Section 6.1, "General Security Issues". For help in diagnosing privilege-related problems, see Section 6.2.7, "Causes of Access-Denied Errors".

6.2.1 Privileges Provided by MySQL

MySQL provides privileges that apply in different contexts and at different levels of operation:

- Administrative privileges enable users to manage operation of the MySQL server. These privileges are global because they are not specific to a particular database.

- Database privileges apply to a database and to all objects within it. These privileges can be granted for specific databases, or globally so that they apply to all databases.

- Privileges for database objects such as tables, indexes, views, and stored routines can be granted for specific objects within a database, for all objects of a given type within a database (for example, all tables in a database), or globally for all objects of a given type in all databases).

Information about account privileges is stored in the `user`, `db`, `tables_priv`, `columns_priv`, and `procs_priv` tables in the `mysql` database (see Section 6.2.2, "Privilege System Grant Tables"). The MySQL server reads the contents of these tables into memory when it starts and reloads them under the circumstances indicated in Section 6.2.6, "When Privilege Changes Take Effect". Access-control decisions are based on the in-memory copies of the grant tables.

Some releases of MySQL introduce changes to the structure of the grant tables to add new privileges or features. To make sure that you can take advantage of any new capabilities, update your grant tables to have the current structure whenever you update to a new version of MySQL. See Section 4.4.7, "`mysql_upgrade` — Check and Upgrade MySQL Tables".

The following table shows the privilege names used at the SQL level in the `GRANT` and `REVOKE` statements, along with the column name associated with each privilege in the grant tables and the context in which the privilege applies.

Table 6.2 Permissible Privileges for `GRANT` and `REVOKE`

Privilege	Column	Context
`CREATE`	`Create_priv`	databases, tables, or indexes
`DROP`	`Drop_priv`	databases, tables, or views
`GRANT OPTION`	`Grant_priv`	databases, tables, or stored routines
`LOCK TABLES`	`Lock_tables_priv`	databases
`REFERENCES`	`References_priv`	databases or tables
`EVENT`	`Event_priv`	databases
`ALTER`	`Alter_priv`	tables
`DELETE`	`Delete_priv`	tables
`INDEX`	`Index_priv`	tables
`INSERT`	`Insert_priv`	tables or columns
`SELECT`	`Select_priv`	tables or columns
`UPDATE`	`Update_priv`	tables or columns
`CREATE TEMPORARY TABLES`	`Create_tmp_table_priv`	tables
`TRIGGER`	`Trigger_priv`	tables
`CREATE VIEW`	`Create_view_priv`	views
`SHOW VIEW`	`Show_view_priv`	views
`ALTER ROUTINE`	`Alter_routine_priv`	stored routines
`CREATE ROUTINE`	`Create_routine_priv`	stored routines
`EXECUTE`	`Execute_priv`	stored routines
`FILE`	`File_priv`	file access on server host
`CREATE TABLESPACE`	`Create_tablespace_priv`	server administration
`CREATE USER`	`Create_user_priv`	server administration
`PROCESS`	`Process_priv`	server administration

Privilege	Column	Context
PROXY	see proxies_priv table	server administration
RELOAD	Reload_priv	server administration
REPLICATION CLIENT	Repl_client_priv	server administration
REPLICATION SLAVE	Repl_slave_priv	server administration
SHOW DATABASES	Show_db_priv	server administration
SHUTDOWN	Shutdown_priv	server administration
SUPER	Super_priv	server administration
ALL [PRIVILEGES]		server administration
USAGE		server administration

The following list provides a general description of each privilege available in MySQL. Particular SQL statements might have more specific privilege requirements than indicated here. If so, the description for the statement in question provides the details.

- The ALL or ALL PRIVILEGES privilege specifier is shorthand. It stands for “all privileges available at a given privilege level” (except GRANT OPTION). For example, granting ALL at the global or table level grants all global privileges or all table-level privileges.
- The ALTER privilege enables use of ALTER TABLE to change the structure of tables. ALTER TABLE also requires the CREATE and INSERT privileges. Renaming a table requires ALTER and DROP on the old table, ALTER, CREATE, and INSERT on the new table.
- The ALTER ROUTINE privilege is needed to alter or drop stored routines (procedures and functions).
- The CREATE privilege enables creation of new databases and tables.
- The CREATE ROUTINE privilege is needed to create stored routines (procedures and functions).
- The CREATE TABLESPACE privilege is needed to create, alter, or drop tablespaces and log file groups.
- The CREATE TEMPORARY TABLES privilege enables the creation of temporary tables using the CREATE TEMPORARY TABLE statement.

  After a session has created a temporary table, the server performs no further privilege checks on the table. The creating session can perform any operation on the table, such as DROP TABLE, INSERT, UPDATE, or SELECT.

  One implication of this behavior is that a session can manipulate its temporary tables even if the current user has no privilege to create them. Suppose that the current user does not have the CREATE TEMPORARY TABLES privilege but is able to execute a DEFINER-context stored procedure that executes with the privileges of a user who does have CREATE TEMPORARY TABLES and that creates a temporary table. While the procedure executes, the session uses the privileges of the defining user. After the procedure returns, the effective privileges revert to those of the current user, which can still see the temporary table and perform any operation on it.

  To keep privileges for temporary and nontemporary tables separate, a common workaround for this situation is to create a database dedicated to the use of temporary tables. Then for that database, a user can be granted the CREATE TEMPORARY TABLES privilege, along with any other privileges required for temporary table operations done by that user.
- The CREATE USER privilege enables use of CREATE USER, DROP USER, RENAME USER, and REVOKE ALL PRIVILEGES.

- The CREATE VIEW privilege enables use of CREATE VIEW.
- The DELETE privilege enables rows to be deleted from tables in a database.
- The DROP privilege enables you to drop (remove) existing databases, tables, and views. The DROP privilege is required in order to use the statement ALTER TABLE ... DROP PARTITION on a partitioned table. The DROP privilege is also required for TRUNCATE TABLE. *If you grant the DROP privilege for the mysql database to a user, that user can drop the database in which the MySQL access privileges are stored.*
- The EVENT privilege is required to create, alter, drop, or see events for the Event Scheduler.
- The EXECUTE privilege is required to execute stored routines (procedures and functions).
- The FILE privilege gives you permission to read and write files on the server host using the LOAD DATA INFILE and SELECT ... INTO OUTFILE statements and the LOAD_FILE() function. A user who has the FILE privilege can read any file on the server host that is either world-readable or readable by the MySQL server. (This implies the user can read any file in any database directory, because the server can access any of those files.) The FILE privilege also enables the user to create new files in any directory where the MySQL server has write access. This includes the server's data directory containing the files that implement the privilege tables. As a security measure, the server will not overwrite existing files.

  To limit the location in which files can be read and written, set the secure_file_priv system to a specific directory. See Section 5.1.4, “Server System Variables”.
- The GRANT OPTION privilege enables you to give to other users or remove from other users those privileges that you yourself possess.
- The INDEX privilege enables you to create or drop (remove) indexes. INDEX applies to existing tables. If you have the CREATE privilege for a table, you can include index definitions in the CREATE TABLE statement.
- The INSERT privilege enables rows to be inserted into tables in a database. INSERT is also required for the ANALYZE TABLE, OPTIMIZE TABLE, and REPAIR TABLE table-maintenance statements.
- The LOCK TABLES privilege enables the use of explicit LOCK TABLES statements to lock tables for which you have the SELECT privilege. This includes the use of write locks, which prevents other sessions from reading the locked table.
- The PROCESS privilege pertains to display of information about the threads executing within the server (that is, information about the statements being executed by sessions). The privilege enables use of SHOW PROCESSLIST or mysqladmin processlist to see threads belonging to other accounts; you can always see your own threads. The PROCESS privilege also enables use of SHOW ENGINE.
- The PROXY privilege enables a user to impersonate or become known as another user. See Section 6.3.10, “Proxy Users”.
- The REFERENCES privilege currently is unused.
- The RELOAD privilege enables use of the FLUSH statement. It also enables mysqladmin commands that are equivalent to FLUSH operations: flush-hosts, flush-logs, flush-privileges, flush-status, flush-tables, flush-threads, refresh, and reload.

  The reload command tells the server to reload the grant tables into memory. flush-privileges is a synonym for reload. The refresh command closes and reopens the log files and flushes all tables. The other flush-*xxx* commands perform functions similar to refresh, but are more specific and may be preferable in some instances. For example, if you want to flush just the log files, flush-logs is a better choice than refresh.

- The REPLICATION CLIENT privilege enables the use of SHOW MASTER STATUS, SHOW SLAVE STATUS, and SHOW BINARY LOGS.
- The REPLICATION SLAVE privilege should be granted to accounts that are used by slave servers to connect to the current server as their master. Without this privilege, the slave cannot request updates that have been made to databases on the master server.
- The SELECT privilege enables you to select rows from tables in a database. SELECT statements require the SELECT privilege only if they actually retrieve rows from a table. Some SELECT statements do not access tables and can be executed without permission for any database. For example, you can use SELECT as a simple calculator to evaluate expressions that make no reference to tables:

  ```
  SELECT 1+1;
  SELECT PI()*2;
  ```

  The SELECT privilege is also needed for other statements that read column values. For example, SELECT is needed for columns referenced on the right hand side of *col_name*=*expr* assignment in UPDATE statements or for columns named in the WHERE clause of DELETE or UPDATE statements.
- The SHOW DATABASES privilege enables the account to see database names by issuing the SHOW DATABASE statement. Accounts that do not have this privilege see only databases for which they have some privileges, and cannot use the statement at all if the server was started with the --skip-show-database option. Note that *any* global privilege is a privilege for the database.
- The SHOW VIEW privilege enables use of SHOW CREATE VIEW.
- The SHUTDOWN privilege enables use of the mysqladmin shutdown command. There is no corresponding SQL statement.
- The SUPER privilege enables an account to use CHANGE MASTER TO, KILL or mysqladmin kill to kill threads belonging to other accounts (you can always kill your own threads), PURGE BINARY LOGS, configuration changes using SET GLOBAL to modify global system variables, the mysqladmin debug command, enabling or disabling logging, performing updates even if the read_only system variable is enabled, starting and stopping replication on slave servers, specification of any account in the DEFINER attribute of stored programs and views, and enables you to connect (once) even if the connection limit controlled by the max_connections system variable is reached.

  To create or alter stored functions if binary logging is enabled, you may also need the SUPER privilege, as described in Section 18.7, “Binary Logging of Stored Programs”.
- The TRIGGER privilege enables trigger operations. You must have this privilege for a table to create, drop, or execute triggers for that table.
- The UPDATE privilege enables rows to be updated in tables in a database.
- The USAGE privilege specifier stands for “no privileges.” It is used at the global level with GRANT to modify account attributes such as resource limits or SSL characteristics without affecting existing account privileges.

It is a good idea to grant to an account only those privileges that it needs. You should exercise particular caution in granting the FILE and administrative privileges:

- The FILE privilege can be abused to read into a database table any files that the MySQL server can read on the server host. This includes all world-readable files and files in the server's data directory. The table can then be accessed using SELECT to transfer its contents to the client host.

- The GRANT OPTION privilege enables users to give their privileges to other users. Two users that have different privileges and with the GRANT OPTION privilege are able to combine privileges.
- The ALTER privilege may be used to subvert the privilege system by renaming tables.
- The SHUTDOWN privilege can be abused to deny service to other users entirely by terminating the server.
- The PROCESS privilege can be used to view the plain text of currently executing statements, including statements that set or change passwords.
- The SUPER privilege can be used to terminate other sessions or change how the server operates.
- Privileges granted for the mysql database itself can be used to change passwords and other access privilege information. Passwords are stored encrypted, so a malicious user cannot simply read them to know the plain text password. However, a user with write access to the user table Password column can change an account's password, and then connect to the MySQL server using that account.

6.2.2 Privilege System Grant Tables

Normally, you manipulate the contents of the grant tables in the mysql database indirectly by using statements such as GRANT and REVOKE to set up accounts and control the privileges available to each one. See Section 13.7.1, “Account Management Statements”. The discussion here describes the underlying structure of the grant tables and how the server uses their contents when interacting with clients.

These mysql database tables contain grant information:

- user: Contains user accounts, global privileges, and other non-privilege columns.
- db: Contains database-level privileges.
- host: Obsolete. New MySQL installations no longer create this table.
- tables_priv: Contains table-level privileges.
- columns_priv: Contains column-level privileges.
- procs_priv: Contains stored procedure and function privileges.
- proxies_priv: Contains proxy-user privileges.

Other tables in the mysql database do not hold grant information and are discussed elsewhere:

- event: Contains information about Event Scheduler events: See Section 18.4, “Using the Event Scheduler”.
- func: Contains information about user-defined functions: See Section 22.3, “Adding New Functions to MySQL”.
- help_*xxx*: These tables are used for server-side help: See Section 5.1.10, “Server-Side Help”.
- plugin: Contains information about server plugins: See Section 5.1.8.1, “Installing and Uninstalling Plugins”, and Section 22.2, “The MySQL Plugin API”.
- proc: Contains information about stored procedures and functions: See Section 18.2, “Using Stored Routines (Procedures and Functions)”.
- servers: Used by the FEDERATED storage engine: See Section 14.9.2.2, “Creating a FEDERATED Table Using CREATE SERVER”.

- `time_zone_xxx`: These tables contain time zone information: See Section 10.6, “MySQL Server Time Zone Support”.
- Tables with `_log` in their name are used for logging: See Section 5.2, “MySQL Server Logs”.

Note

Modifications to tables in the `mysql` database normally are made by the server in response to statements such as `CREATE USER`, `GRANT`, or `CREATE PROCEDURE`. Direct modification of these tables using statements such as `INSERT`, `UPDATE`, or `DELETE` is not encouraged. The server is free to ignore rows that become malformed as a result of such modifications.

Each grant table contains scope columns and privilege columns:

- Scope columns determine the scope of each row (entry) in the tables; that is, the context in which the row applies. For example, a `user` table row with `Host` and `User` values of `'thomas.loc.gov'` and `'bob'` would be used for authenticating connections made to the server from the host `thomas.loc.gov` by a client that specifies a user name of `bob`. Similarly, a `db` table row with `Host`, `User`, and `Db` column values of `'thomas.loc.gov'`, `'bob'` and `'reports'` would be used when `bob` connects from the host `thomas.loc.gov` to access the `reports` database. The `tables_priv` and `columns_priv` tables contain scope columns indicating tables or table/column combinations to which each row applies. The `procs_priv` scope columns indicate the stored routine to which each row applies.
- Privilege columns indicate which privileges are granted by a table row; that is, what operations can be performed. The server combines the information in the various grant tables to form a complete description of a user's privileges. Section 6.2.5, “Access Control, Stage 2: Request Verification”, describes the rules that are used to do this.

The server uses the grant tables in the following manner:

- The `user` table scope columns determine whether to reject or permit incoming connections. For permitted connections, any privileges granted in the `user` table indicate the user's global privileges. Any privilege granted in this table applies to *all* databases on the server.

  Note

  Because any global privilege is considered a privilege for all databases, any global privilege enables a user to see all database names with `SHOW DATABASES` or by examining the `SCHEMATA` table of `INFORMATION_SCHEMA`.

- The `db` table scope columns determine which users can access which databases from which hosts. The privilege columns determine which operations are permitted. A privilege granted at the database level applies to the database and to all objects in the database, such as tables and stored programs.
- The `tables_priv` and `columns_priv` tables are similar to the `db` table, but are more fine-grained: They apply at the table and column levels rather than at the database level. A privilege granted at the table level applies to the table and to all its columns. A privilege granted at the column level applies only to a specific column.
- The `procs_priv` table applies to stored routines. A privilege granted at the routine level applies only to a single routine.
- The `proxies_priv` table indicates which users can act as proxies for other users and whether proxy users can grant the `PROXY` privilege to other users.

The server uses the user and db tables in the mysql database at both the first and second stages of access control (see Section 6.2, “The MySQL Access Privilege System”). The columns in the user and db tables are shown here.

Table 6.3 user and db Table Columns

Table Name	**user**	**db**
Scope columns	Host	Host
	User	Db
	Password	User
Privilege columns	Select_priv	Select_priv
	Insert_priv	Insert_priv
	Update_priv	Update_priv
	Delete_priv	Delete_priv
	Index_priv	Index_priv
	Alter_priv	Alter_priv
	Create_priv	Create_priv
	Drop_priv	Drop_priv
	Grant_priv	Grant_priv
	Create_view_priv	Create_view_priv
	Show_view_priv	Show_view_priv
	Create_routine_priv	Create_routine_priv
	Alter_routine_priv	Alter_routine_priv
	Execute_priv	Execute_priv
	Trigger_priv	Trigger_priv
	Event_priv	Event_priv
	Create_tmp_table_priv	Create_tmp_table_priv
	Lock_tables_priv	Lock_tables_priv
	References_priv	References_priv
	Reload_priv	
	Shutdown_priv	
	Process_priv	
	File_priv	
	Show_db_priv	
	Super_priv	
	Repl_slave_priv	
	Repl_client_priv	
	Create_user_priv	
	Create_tablespace_priv	
Security columns	ssl_type	
	ssl_cipher	

Table Name	user	db
	x509_issuer	
	x509_subject	
	plugin	
	authentication_string	
	password_expired	
	password_last_changed	
	password_lifetime	
Resource control columns	max_questions	
	max_updates	
	max_connections	
	max_user_connections	

The mysql.user table plugin and authentication_string columns store authentication plugin information.

As of MySQL 5.7.2, the plugin column must be nonempty, and the server uses the named plugin to authenticate connection attempts for the account. Whether the plugin uses the value in the Password column is up to the plugin.

Before MySQL 5.7.2, the plugin column for an account row is permitted to be empty. In this case, the server authenticates the account using the mysql_native_password or mysql_old_password plugin implicitly, depending on the format of the password hash in the Password column. If the Password value is empty or a 4.1 password hash (41 characters), the server uses mysql_native_password. If the password value is a pre-4.1 password hash (16 characters), the server uses mysql_old_password. (For additional information about these hash formats, see Section 6.1.2.4, “Password Hashing in MySQL”.) Clients must match the password in the Password column of the account row.

At startup, and at runtime when FLUSH PRIVILEGES is executed, the server checks user table rows. As of MySQL 5.7.2, for any row with an empty plugin column, the server writes a warning to the error log of this form:

```
[Warning] User entry 'user_name'@'host_name' has an empty plugin
value. The user will be ignored and no one can login with this user
anymore.
```

To address this issue, execute mysql_upgrade.

If an account row names a plugin in the plugin column, the server uses it to authenticate connection attempts for the account. Whether the plugin uses the value in the Password column is up to the plugin.

The password_expired column permits DBAs to expire account passwords and require users to reset their password. The default password_expired value is 'N', but can be set to 'Y' with the ALTER USER statement. After an account's password has been expired, all operations performed by the account in subsequent connections to the server result in an error until the user issues a SET PASSWORD statement to establish a new account password. See Section 13.7.1.1, “ALTER USER Syntax”.

It is possible after password expiration to “reset” a password by using SET PASSWORD to set it to its current value. As a matter of good policy, it is preferable to choose a different password.

password_last_changed (added in MySQL 5.7.4) is a TIMESTAMP column indicating when the password was last changed. The value is non-NULL only for accounts that use MySQL built-in authentication methods (accounts that use an authentication plugin of mysql_native_password, mysql_old_password, or sha256_password). The value is NULL for other accounts, such as those authenticated using an external authentication system.

password_last_changed is updated by the CREATE USER and SET PASSWORD statements, and by GRANT statements that create an account or change an account password.

password_lifetime (added in MySQL 5.7.4) indicates the account password lifetime, in days. If the password is past its lifetime (assessed using the password_last_changed column), the server considers the password expired when clients connect using the account. A value of *N* greater than zero means that the password must be changed every *N* days. A value of 0 disables automatic password expiration. If the value is NULL (the default), the global expiration policy applies, as defined by the default_password_lifetime system variable.

During the second stage of access control, the server performs request verification to make sure that each client has sufficient privileges for each request that it issues. In addition to the user and db grant tables, the server may also consult the tables_priv and columns_priv tables for requests that involve tables. The latter tables provide finer privilege control at the table and column levels. They have the columns shown in the following table.

Table 6.4 tables_priv and columns_priv Table Columns

Table Name	tables_priv	columns_priv
Scope columns	Host	Host
	Db	Db
	User	User
	Table_name	Table_name
		Column_name
Privilege columns	Table_priv	Column_priv
	Column_priv	
Other columns	Timestamp	Timestamp
	Grantor	

The Timestamp and Grantor columns are set to the current timestamp and the CURRENT_USER value, respectively. However, they are unused and are discussed no further here.

For verification of requests that involve stored routines, the server may consult the procs_priv table, which has the columns shown in the following table.

Table 6.5 procs_priv Table Columns

Table Name	procs_priv
Scope columns	Host
	Db
	User
	Routine_name
	Routine_type
Privilege columns	Proc_priv
Other columns	Timestamp
	Grantor

The Routine_type column is an ENUM column with values of 'FUNCTION' or 'PROCEDURE' to indicate the type of routine the row refers to. This column enables privileges to be granted separately for a function and a procedure with the same name.

The Timestamp and Grantor columns currently are unused and are discussed no further here.

The proxies_priv table records information about proxy users. It has these columns:

- Host, User: These columns indicate the user account that has the PROXY privilege for the proxied account.
- Proxied_host, Proxied_user: These columns indicate the account of the proxied user.
- Grantor: Currently unused.
- Timestamp: Currently unused.
- With_grant: This column indicates whether the proxy account can grant the PROXY privilege to other accounts.

Scope columns in the grant tables contain strings. They are declared as shown here; the default value for each is the empty string.

Table 6.6 Grant Table Scope Column Types

Column Name	Type
Host, Proxied_host	CHAR(60)
User, Proxied_user	CHAR(16)
Password	CHAR(41)
Db	CHAR(64)
Table_name	CHAR(64)
Column_name	CHAR(64)
Routine_name	CHAR(64)

For access-checking purposes, comparisons of User, Proxied_user, Password, Db, and Table_name values are case sensitive. Comparisons of Host, Proxied_host, Column_name, and Routine_name values are not case sensitive.

In the user and db tables, each privilege is listed in a separate column that is declared as ENUM('N','Y') DEFAULT 'N'. In other words, each privilege can be disabled or enabled, with the default being disabled.

In the tables_priv, columns_priv, and procs_priv tables, the privilege columns are declared as SET columns. Values in these columns can contain any combination of the privileges controlled by the table. Only those privileges listed in the column value are enabled.

Table 6.7 Set-Type Privilege Column Values

Table Name	Column Name	Possible Set Elements
tables_priv	Table_priv	'Select', 'Insert', 'Update', 'Delete', 'Create', 'Drop', 'Grant', 'References', 'Index', 'Alter', 'Create View', 'Show view', 'Trigger'
tables_priv	Column_priv	'Select', 'Insert', 'Update', 'References'

Table Name	Column Name	Possible Set Elements
columns_priv	Column_priv	'Select', 'Insert', 'Update', 'References'
procs_priv	Proc_priv	'Execute', 'Alter Routine', 'Grant'

Administrative privileges (such as RELOAD or SHUTDOWN) are specified only in the user table. Administrative operations are operations on the server itself and are not database-specific, so there is no reason to list these privileges in the other grant tables. Consequently, to determine whether you can perform an administrative operation, the server need consult only the user table.

The FILE privilege also is specified only in the user table. It is not an administrative privilege as such, but your ability to read or write files on the server host is independent of the database you are accessing.

The mysqld server reads the contents of the grant tables into memory when it starts. You can tell it to reload the tables by issuing a FLUSH PRIVILEGES statement or executing a mysqladmin flush-privileges or mysqladmin reload command. Changes to the grant tables take effect as indicated in Section 6.2.6, “When Privilege Changes Take Effect”.

When you modify an account's privileges, it is a good idea to verify that the changes set up privileges the way you want. To check the privileges for a given account, use the SHOW GRANTS statement (see Section 13.7.5.20, “SHOW GRANTS Syntax”). For example, to determine the privileges that are granted to an account with user name and host name values of bob and pc84.example.com, use this statement:

```
SHOW GRANTS FOR 'bob'@'pc84.example.com';
```

6.2.3 Specifying Account Names

MySQL account names consist of a user name and a host name. This enables creation of accounts for users with the same name who can connect from different hosts. This section describes how to write account names, including special values and wildcard rules.

In SQL statements such as CREATE USER, GRANT, and SET PASSWORD, write account names using the following rules:

- Syntax for account names is '*user_name*'@'*host_name*'.

- An account name consisting only of a user name is equivalent to '*user_name*'@'%'. For example, 'me' is equivalent to 'me'@'%'.

- The user name and host name need not be quoted if they are legal as unquoted identifiers. Quotes are necessary to specify a *user_name* string containing special characters (such as “-”), or a *host_name* string containing special characters or wildcard characters (such as “%”); for example, 'test-user'@'%.com'.

- Quote user names and host names as identifiers or as strings, using either backticks (“`”), single quotation marks (“'”), or double quotation marks (“"”).

- The user name and host name parts, if quoted, must be quoted separately. That is, write 'me'@'localhost', not 'me@localhost'; the latter is interpreted as 'me@localhost'@'%'.

- A reference to the CURRENT_USER or CURRENT_USER() function is equivalent to specifying the current client's user name and host name literally.

MySQL stores account names in grant tables in the mysql database using separate columns for the user name and host name parts:

- The user table contains one row for each account. The User and Host columns store the user name and host name. This table also indicates which global privileges the account has.
- Other grant tables indicate privileges an account has for databases and objects within databases. These tables have User and Host columns to store the account name. Each row in these tables associates with the account in the user table that has the same User and Host values.

For additional detail about grant table structure, see Section 6.2.2, "Privilege System Grant Tables".

User names and host names have certain special values or wildcard conventions, as described following.

A user name is either a nonblank value that literally matches the user name for incoming connection attempts, or a blank value (empty string) that matches any user name. An account with a blank user name is an anonymous user. To specify an anonymous user in SQL statements, use a quoted empty user name part, such as `''@'localhost'`.

The host name part of an account name can take many forms, and wildcards are permitted:

- A host value can be a host name or an IP address (IPv4 or IPv6). The name `'localhost'` indicates the local host. The IP address `'127.0.0.1'` indicates the IPv4 loopback interface. The IP address `'::1'` indicates the IPv6 loopback interface.
- You can use the wildcard characters "%" and "_" in host name or IP address values. These have the same meaning as for pattern-matching operations performed with the LIKE operator. For example, a host value of `'%'` matches any host name, whereas a value of `'%.mysql.com'` matches any host in the mysql.com domain. `'192.168.1.%'` matches any host in the 192.168.1 class C network.

  Because you can use IP wildcard values in host values (for example, `'192.168.1.%'` to match every host on a subnet), someone could try to exploit this capability by naming a host 192.168.1.somewhere.com. To foil such attempts, MySQL disallows matching on host names that start with digits and a dot. Thus, if you have a host named something like 1.2.example.com, its name never matches the host part of account names. An IP wildcard value can match only IP addresses, not host names.
- For a host value specified as an IPv4 address, you can specify a netmask indicating how many address bits to use for the network number. Netmask notation cannot be used for IPv6 addresses.

  The syntax is *host_ip/netmask*. For example:

  ```
  CREATE USER 'david'@'192.58.197.0/255.255.255.0';
  ```

  This enables david to connect from any client host having an IP address *client_ip* for which the following condition is true:

  ```
  client_ip & netmask = host_ip
  ```

  That is, for the CREATE USER statement just shown:

  ```
  client_ip & 255.255.255.0 = 192.58.197.0
  ```

  IP addresses that satisfy this condition and can connect to the MySQL server are those in the range from 192.58.197.0 to 192.58.197.255.

  The netmask can only be used to tell the server to use 8, 16, 24, or 32 bits of the address. Examples:

  - 192.0.0.0/255.0.0.0: Any host on the 192 class A network

- 192.168.0.0/255.255.0.0: Any host on the 192.168 class B network
- 192.168.1.0/255.255.255.0: Any host on the 192.168.1 class C network
- 192.168.1.1: Only the host with this specific IP address

The following netmask will not work because it masks 28 bits, and 28 is not a multiple of 8:

```
192.168.0.1/255.255.255.240
```

The server performs matching of host values in account names against the client host using the value returned by the system DNS resolver for the client host name or IP address. Except in the case that the account host value is specified using netmask notation, this comparison is performed as a string match, even for an account host value given as an IP address. This means that you should specify account host values in the same format used by DNS. Here are examples of problems to watch out for:

- Suppose that a host on the local network has a fully qualified name of host1.example.com. If DNS returns name lookups for this host as host1.example.com, use that name in account host values. But if DNS returns just host1, use host1 instead.
- If DNS returns the IP address for a given host as 192.168.1.2, that will match an account host value of 192.168.1.2 but not 192.168.01.2. Similarly, it will match an account host pattern like 192.168.1.% but not 192.168.01.%.

To avoid problems like this, it is advisable to check the format in which your DNS returns host names and addresses, and use values in the same format in MySQL account names.

6.2.4 Access Control, Stage 1: Connection Verification

When you attempt to connect to a MySQL server, the server accepts or rejects the connection based on your identity and whether you can verify your identity by supplying the correct password. If not, the server denies access to you completely. Otherwise, the server accepts the connection, and then enters Stage 2 and waits for requests.

Your identity is based on two pieces of information:

- The client host from which you connect
- Your MySQL user name

Identity checking is performed using the three user table scope columns (Host, User, and Password). The server accepts the connection only if the Host and User columns in some user table row match the client host name and user name and the client supplies the password specified in that row. The rules for permissible Host and User values are given in Section 6.2.3, “Specifying Account Names”.

If the User column value is nonblank, the user name in an incoming connection must match exactly. If the User value is blank, it matches any user name. If the user table row that matches an incoming connection has a blank user name, the user is considered to be an anonymous user with no name, not a user with the name that the client actually specified. This means that a blank user name is used for all further access checking for the duration of the connection (that is, during Stage 2).

The Password column can be blank. This is not a wildcard and does not mean that any password matches. It means that the user must connect without specifying a password. If the server authenticates a client using a plugin, the authentication method that the plugin implements may or may not use the password in the Password column. In this case, it is possible that an external password is also used to authenticate to the MySQL server.

Nonblank Password values in the user table represent encrypted passwords. MySQL does not store passwords in plaintext form for anyone to see. Rather, the password supplied by a user who is attempting to connect is encrypted (using the PASSWORD() function). The encrypted password then is used during the connection process when checking whether the password is correct. This is done without the encrypted password ever traveling over the connection. See Section 6.3.1, “User Names and Passwords”.

From MySQL's point of view, the encrypted password is the *real* password, so you should never give anyone access to it. In particular, *do not give nonadministrative users read access to tables in the mysql database*.

The following table shows how various combinations of Host and User values in the user table apply to incoming connections.

Host Value	**User Value**	**Permissible Connections**
'thomas.loc.gov'	'fred'	fred, connecting from thomas.loc.gov
'thomas.loc.gov'	''	Any user, connecting from thomas.loc.gov
'%'	'fred'	fred, connecting from any host
'%'	''	Any user, connecting from any host
'%.loc.gov'	'fred'	fred, connecting from any host in the loc.gov domain
'x.y.%'	'fred'	fred, connecting from x.y.net, x.y.com, x.y.edu, and so on; this is probably not useful
'144.155.166.177'	'fred'	fred, connecting from the host with IP address 144.155.166.177
'144.155.166.%'	'fred'	fred, connecting from any host in the 144.155.166 class C subnet
'144.155.166.0/255.255.255.0'	'fred'	Same as previous example

It is possible for the client host name and user name of an incoming connection to match more than one row in the user table. The preceding set of examples demonstrates this: Several of the entries shown match a connection from thomas.loc.gov by fred.

When multiple matches are possible, the server must determine which of them to use. It resolves this issue as follows:

- Whenever the server reads the user table into memory, it sorts the rows.
- When a client attempts to connect, the server looks through the rows in sorted order.
- The server uses the first row that matches the client host name and user name.

The server uses sorting rules that order rows with the most-specific Host values first. Literal host names and IP addresses are the most specific. (The specificity of a literal IP address is not affected by whether it has a netmask, so 192.168.1.13 and 192.168.1.0/255.255.255.0 are considered equally specific.) The pattern '%' means “any host” and is least specific. The empty string '' also means “any host” but sorts after '%'. Rows with the same Host value are ordered with the most-specific User values first (a blank User value means “any user” and is least specific).

To see how this works, suppose that the user table looks like this:

```
+-----------+----------+-
| Host      | User     | ...
```

```
+-----------+----------+-
| %         | root     | ...
| %         | jeffrey  | ...
| localhost | root     | ...
| localhost |          | ...
+-----------+----------+-
```

When the server reads the table into memory, it sorts the rows using the rules just described. The result after sorting looks like this:

```
+-----------+----------+-
| Host      | User     | ...
+-----------+----------+-
| localhost | root     | ...
| localhost |          | ...
| %         | jeffrey  | ...
| %         | root     | ...
+-----------+----------+-
```

When a client attempts to connect, the server looks through the sorted rows and uses the first match found. For a connection from `localhost` by `jeffrey`, two of the rows from the table match: the one with `Host` and `User` values of `'localhost'` and `''`, and the one with values of `'%'` and `'jeffrey'`. The `'localhost'` row appears first in sorted order, so that is the one the server uses.

Here is another example. Suppose that the `user` table looks like this:

```
+----------------+----------+-
| Host           | User     | ...
+----------------+----------+-
| %              | jeffrey  | ...
| thomas.loc.gov |          | ...
+----------------+----------+-
```

The sorted table looks like this:

```
+----------------+----------+-
| Host           | User     | ...
+----------------+----------+-
| thomas.loc.gov |          | ...
| %              | jeffrey  | ...
+----------------+----------+-
```

A connection by `jeffrey` from `thomas.loc.gov` is matched by the first row, whereas a connection by `jeffrey` from any host is matched by the second.

Note

It is a common misconception to think that, for a given user name, all rows that explicitly name that user are used first when the server attempts to find a match for the connection. This is not true. The preceding example illustrates this, where a connection from `thomas.loc.gov` by `jeffrey` is first matched not by the row containing `'jeffrey'` as the `User` column value, but by the row with no user name. As a result, `jeffrey` is authenticated as an anonymous user, even though he specified a user name when connecting.

If you are able to connect to the server, but your privileges are not what you expect, you probably are being authenticated as some other account. To find out what account the server used to authenticate you, use the `CURRENT_USER()` function. (See Section 12.14, “Information Functions”.) It returns a value in *user_name@host_name* format that indicates the User and Host values from the matching user table row. Suppose that jeffrey connects and issues the following query:

```
mysql> SELECT CURRENT_USER();
+----------------+
| CURRENT_USER() |
+----------------+
| @localhost     |
+----------------+
```

The result shown here indicates that the matching user table row had a blank User column value. In other words, the server is treating jeffrey as an anonymous user.

Another way to diagnose authentication problems is to print out the user table and sort it by hand to see where the first match is being made.

6.2.5 Access Control, Stage 2: Request Verification

After you establish a connection, the server enters Stage 2 of access control. For each request that you issue through that connection, the server determines what operation you want to perform, then checks whether you have sufficient privileges to do so. This is where the privilege columns in the grant tables come into play. These privileges can come from any of the user, db, tables_priv, columns_priv, or procs_priv tables. (You may find it helpful to refer to Section 6.2.2, “Privilege System Grant Tables”, which lists the columns present in each of the grant tables.)

The user table grants privileges that are assigned to you on a global basis and that apply no matter what the default database is. For example, if the user table grants you the DELETE privilege, you can delete rows from any table in any database on the server host! It is wise to grant privileges in the user table only to people who need them, such as database administrators. For other users, you should leave all privileges in the user table set to 'N' and grant privileges at more specific levels only. You can grant privileges for particular databases, tables, columns, or routines.

The db table grants database-specific privileges. Values in the scope columns of this table can take the following forms:

- A blank User value matches the anonymous user. A nonblank value matches literally; there are no wildcards in user names.
- The wildcard characters “%” and “_” can be used in the Host and Db columns. These have the same meaning as for pattern-matching operations performed with the LIKE operator. If you want to use either character literally when granting privileges, you must escape it with a backslash. For example, to include the underscore character (“_”) as part of a database name, specify it as “_” in the GRANT statement.
- A '%' or blank Host value means “any host.”
- A '%' or blank Db value means “any database.”

The server reads the db table into memory and sorts it at the same time that it reads the user table. The server sorts the db table based on the Host, Db, and User scope columns. As with the user table, sorting puts the most-specific values first and least-specific values last, and when the server looks for matching entries, it uses the first match that it finds.

The tables_priv, columns_priv, and procs_priv tables grant table-specific, column-specific, and routine-specific privileges. Values in the scope columns of these tables can take the following forms:

- The wildcard characters “%” and “_” can be used in the Host column. These have the same meaning as for pattern-matching operations performed with the LIKE operator.

- A '%' or blank Host value means “any host.”
- The Db, Table_name, Column_name, and Routine_name columns cannot contain wildcards or be blank.

The server sorts the tables_priv, columns_priv, and procs_priv tables based on the Host, Db, and User columns. This is similar to db table sorting, but simpler because only the Host column can contain wildcards.

The server uses the sorted tables to verify each request that it receives. For requests that require administrative privileges such as SHUTDOWN or RELOAD, the server checks only the user table row because that is the only table that specifies administrative privileges. The server grants access if the row permits the requested operation and denies access otherwise. For example, if you want to execute mysqladmin shutdown but your user table row does not grant the SHUTDOWN privilege to you, the server denies access without even checking the db table. (It contains no Shutdown_priv column, so there is no need to do so.)

For database-related requests (INSERT, UPDATE, and so on), the server first checks the user's global privileges by looking in the user table row. If the row permits the requested operation, access is granted. If the global privileges in the user table are insufficient, the server determines the user's database-specific privileges by checking the db table:

The server looks in the db table for a match on the Host, Db, and User columns. The Host and User columns are matched to the connecting user's host name and MySQL user name. The Db column is matched to the database that the user wants to access. If there is no row for the Host and User, access is denied.

After determining the database-specific privileges granted by the db table entries, the server adds them to the global privileges granted by the user table. If the result permits the requested operation, access is granted. Otherwise, the server successively checks the user's table and column privileges in the tables_priv and columns_priv tables, adds those to the user's privileges, and permits or denies access based on the result. For stored-routine operations, the server uses the procs_priv table rather than tables_priv and columns_priv.

Expressed in boolean terms, the preceding description of how a user's privileges are calculated may be summarized like this:

```
global privileges
OR (database privileges AND host privileges)
OR table privileges
OR column privileges
OR routine privileges
```

It may not be apparent why, if the global user row privileges are initially found to be insufficient for the requested operation, the server adds those privileges to the database, table, and column privileges later. The reason is that a request might require more than one type of privilege. For example, if you execute an INSERT INTO ... SELECT statement, you need both the INSERT and the SELECT privileges. Your privileges might be such that the user table row grants one privilege and the db table row grants the other. In this case, you have the necessary privileges to perform the request, but the server cannot tell that from either table by itself; the privileges granted by the entries in both tables must be combined.

6.2.6 When Privilege Changes Take Effect

When mysqld starts, it reads all grant table contents into memory. The in-memory tables become effective for access control at that point.

If you modify the grant tables indirectly using account-management statements such as `GRANT`, `REVOKE`, `SET PASSWORD`, or `RENAME USER`, the server notices these changes and loads the grant tables into memory again immediately.

If you modify the grant tables directly using statements such as `INSERT`, `UPDATE`, or `DELETE`, your changes have no effect on privilege checking until you either restart the server or tell it to reload the tables. If you change the grant tables directly but forget to reload them, your changes have *no effect* until you restart the server. This may leave you wondering why your changes seem to make no difference!

To tell the server to reload the grant tables, perform a flush-privileges operation. This can be done by issuing a `FLUSH PRIVILEGES` statement or by executing a `mysqladmin flush-privileges` or `mysqladmin reload` command.

A grant table reload affects privileges for each existing client connection as follows:

- Table and column privilege changes take effect with the client's next request.
- Database privilege changes take effect the next time the client executes a `USE db_name` statement.

> **Note**
>
> Client applications may cache the database name; thus, this effect may not be visible to them without actually changing to a different database or flushing the privileges.

- Global privileges and passwords are unaffected for a connected client. These changes take effect only for subsequent connections.

If the server is started with the `--skip-grant-tables` option, it does not read the grant tables or implement any access control. Anyone can connect and do anything, *which is insecure.* To cause a server thus started to read the tables and enable access checking, flush the privileges.

6.2.7 Causes of Access-Denied Errors

If you encounter problems when you try to connect to the MySQL server, the following items describe some courses of action you can take to correct the problem.

- Make sure that the server is running. If it is not, clients cannot connect to it. For example, if an attempt to connect to the server fails with a message such as one of those following, one cause might be that the server is not running:

```
shell> mysql
ERROR 2003: Can't connect to MySQL server on 'host_name' (111)
shell> mysql
ERROR 2002: Can't connect to local MySQL server through socket
'/tmp/mysql.sock' (111)
```

- It might be that the server is running, but you are trying to connect using a TCP/IP port, named pipe, or Unix socket file different from the one on which the server is listening. To correct this when you invoke a client program, specify a `--port` option to indicate the proper port number, or a `--socket` option to indicate the proper named pipe or Unix socket file. To find out where the socket file is, you can use this command:

```
shell> netstat -ln | grep mysql
```

- Make sure that the server has not been configured to ignore network connections or (if you are attempting to connect remotely) that it has not been configured to listen only locally on its network interfaces. If the server was started with `--skip-networking`, it will not accept TCP/IP connections at all. If the server was started with `--bind-address=127.0.0.1`, it will listen for TCP/IP connections only locally on the loopback interface and will not accept remote connections.

- Check to make sure that there is no firewall blocking access to MySQL. Your firewall may be configured on the basis of the application being executed, or the port number used by MySQL for communication (3306 by default). Under Linux or Unix, check your IP tables (or similar) configuration to ensure that the port has not been blocked. Under Windows, applications such as ZoneAlarm or the Windows XP personal firewall may need to be configured not to block the MySQL port.

- The grant tables must be properly set up so that the server can use them for access control. For some distribution types (such as binary distributions on Windows, or RPM distributions on Linux), the installation process initializes the `mysql` database containing the grant tables. For distributions that do not do this, you must initialize the grant tables manually by running the `mysql_install_db` script. For details, see Section 2.9.1, “Postinstallation Procedures for Unix-like Systems”.

  To determine whether you need to initialize the grant tables, look for a `mysql` directory under the data directory. (The data directory normally is named `data` or `var` and is located under your MySQL installation directory.) Make sure that you have a file named `user.MYD` in the `mysql` database directory. If not, execute the `mysql_install_db` script. After running this script and starting the server, test the initial privileges by executing this command:

  ```
  shell> mysql -u root test
  ```

  The server should let you connect without error.

- After a fresh installation, you should connect to the server and set up your users and their access permissions:

  ```
  shell> mysql -u root mysql
  ```

  The server should let you connect because the MySQL `root` user has no password initially. That is also a security risk, so setting the password for the `root` accounts is something you should do while you're setting up your other MySQL accounts. For instructions on setting the initial passwords, see Section 2.9.2, “Securing the Initial MySQL Accounts”.

- If you have updated an existing MySQL installation to a newer version, did you run the `mysql_upgrade` script? If not, do so. The structure of the grant tables changes occasionally when new capabilities are added, so after an upgrade you should always make sure that your tables have the current structure. For instructions, see Section 4.4.7, “`mysql_upgrade` — Check and Upgrade MySQL Tables”.

- If a client program receives the following error message when it tries to connect, it means that the server expects passwords in a newer format than the client is capable of generating:

  ```
  shell> mysql
  Client does not support authentication protocol requested
  by server; consider upgrading MySQL client
  ```

  For information on how to deal with this, see Section 6.1.2.4, “Password Hashing in MySQL”, and Section C.5.2.4, “`Client does not support authentication protocol`”.

- Remember that client programs use connection parameters specified in option files or environment variables. If a client program seems to be sending incorrect default connection parameters when you have not specified them on the command line, check any applicable option files and your environment.

For example, if you get Access denied when you run a client without any options, make sure that you have not specified an old password in any of your option files!

You can suppress the use of option files by a client program by invoking it with the --no-defaults option. For example:

```
shell> mysqladmin --no-defaults -u root version
```

The option files that clients use are listed in Section 4.2.3.3, “Using Option Files”. Environment variables are listed in Section 2.11, “Environment Variables”.

- If you get the following error, it means that you are using an incorrect root password:

  ```
  shell> mysqladmin -u root -pxxxx ver
  Access denied for user 'root'@'localhost' (using password: YES)
  ```

  If the preceding error occurs even when you have not specified a password, it means that you have an incorrect password listed in some option file. Try the --no-defaults option as described in the previous item.

  For information on changing passwords, see Section 6.3.5, “Assigning Account Passwords”.

  If you have lost or forgotten the root password, see Section C.5.4.1, “How to Reset the Root Password”.

- If you change a password by using SET PASSWORD, INSERT, or UPDATE, you must encrypt the password using the PASSWORD() function. If you do not use PASSWORD() for these statements, the password will not work. For example, the following statement assigns a password, but fails to encrypt it, so the user is not able to connect afterward:

  ```
  SET PASSWORD FOR 'abe'@'host_name' = 'eagle';
  ```

  Instead, set the password like this:

  ```
  SET PASSWORD FOR 'abe'@'host_name' = PASSWORD('eagle');
  ```

  The PASSWORD() function is unnecessary when you specify a password using the CREATE USER or GRANT statements or the mysqladmin password command. Each of those automatically uses PASSWORD() to encrypt the password. See Section 6.3.5, “Assigning Account Passwords”, and Section 13.7.1.2, “CREATE USER Syntax”.

- localhost is a synonym for your local host name, and is also the default host to which clients try to connect if you specify no host explicitly.

  To avoid this problem on such systems, you can use a --host=127.0.0.1 option to name the server host explicitly. This will make a TCP/IP connection to the local mysqld server. You can also use TCP/IP by specifying a --host option that uses the actual host name of the local host. In this case, the host name must be specified in a user table row on the server host, even though you are running the client program on the same host as the server.

- The Access denied error message tells you who you are trying to log in as, the client host from which you are trying to connect, and whether you were using a password. Normally, you should have one row in the user table that exactly matches the host name and user name that were given in the error message. For example, if you get an error message that contains using password: NO, it means that you tried to log in without a password.

- If you get an Access denied error when trying to connect to the database with mysql -u *user_name*, you may have a problem with the user table. Check this by executing mysql -u root mysql and issuing this SQL statement:

  ```
  SELECT * FROM user;
  ```

  The result should include a row with the Host and User columns matching your client's host name and your MySQL user name.

- If the following error occurs when you try to connect from a host other than the one on which the MySQL server is running, it means that there is no row in the user table with a Host value that matches the client host:

  ```
  Host ... is not allowed to connect to this MySQL server
  ```

  You can fix this by setting up an account for the combination of client host name and user name that you are using when trying to connect.

  If you do not know the IP address or host name of the machine from which you are connecting, you should put a row with '%' as the Host column value in the user table. After trying to connect from the client machine, use a SELECT USER() query to see how you really did connect. Then change the '%' in the user table row to the actual host name that shows up in the log. Otherwise, your system is left insecure because it permits connections from any host for the given user name.

  On Linux, another reason that this error might occur is that you are using a binary MySQL version that is compiled with a different version of the glibc library than the one you are using. In this case, you should either upgrade your operating system or glibc, or download a source distribution of MySQL version and compile it yourself. A source RPM is normally trivial to compile and install, so this is not a big problem.

- If you specify a host name when trying to connect, but get an error message where the host name is not shown or is an IP address, it means that the MySQL server got an error when trying to resolve the IP address of the client host to a name:

  ```
  shell> mysqladmin -u root -pxxxx -h some_hostname ver
  Access denied for user 'root'@'' (using password: YES)
  ```

  If you try to connect as root and get the following error, it means that you do not have a row in the user table with a User column value of 'root' and that mysqld cannot resolve the host name for your client:

  ```
  Access denied for user ''@'unknown'
  ```

  These errors indicate a DNS problem. To fix it, execute mysqladmin flush-hosts to reset the internal DNS host cache. See Section 8.11.5.2, “DNS Lookup Optimization and the Host Cache”.

  Some permanent solutions are:

  - Determine what is wrong with your DNS server and fix it.

  - Specify IP addresses rather than host names in the MySQL grant tables.

  - Put an entry for the client machine name in /etc/hosts on Unix or \windows\hosts on Windows.

  - Start mysqld with the --skip-name-resolve option.
  - Start mysqld with the --skip-host-cache option.
  - On Unix, if you are running the server and the client on the same machine, connect to localhost. Unix connections to localhost use a Unix socket file rather than TCP/IP.
  - On Windows, if you are running the server and the client on the same machine and the server supports named pipe connections, connect to the host name . (period). Connections to . use a named pipe rather than TCP/IP.

- If mysql -u root test works but mysql -h *your_hostname* -u root test results in Access denied (where *your_hostname* is the actual host name of the local host), you may not have the correct name for your host in the user table. A common problem here is that the Host value in the user table row specifies an unqualified host name, but your system's name resolution routines return a fully qualified domain name (or vice versa). For example, if you have an entry with host 'pluto' in the user table, but your DNS tells MySQL that your host name is 'pluto.example.com', the entry does not work. Try adding an entry to the user table that contains the IP address of your host as the Host column value. (Alternatively, you could add an entry to the user table with a Host value that contains a wildcard; for example, 'pluto.%'. However, use of Host values ending with “%” is *insecure* and is *not* recommended!)

- If mysql -u *user_name* test works but mysql -u *user_name other_db* does not, you have not granted access to the given user for the database named *other_db*.

- If mysql -u *user_name* works when executed on the server host, but mysql -h *host_name* -u *user_name* does not work when executed on a remote client host, you have not enabled access to the server for the given user name from the remote host.

- If you cannot figure out why you get Access denied, remove from the user table all entries that have Host values containing wildcards (entries that contain '%' or '_' characters). A very common error is to insert a new entry with Host='%' and User='*some_user*', thinking that this enables you to specify localhost to connect from the same machine. The reason that this does not work is that the default privileges include an entry with Host='localhost' and User=''. Because that entry has a Host value 'localhost' that is more specific than '%', it is used in preference to the new entry when connecting from localhost! The correct procedure is to insert a second entry with Host='localhost' and User='*some_user*', or to delete the entry with Host='localhost' and User=''. After deleting the entry, remember to issue a FLUSH PRIVILEGES statement to reload the grant tables. See also Section 6.2.4, “Access Control, Stage 1: Connection Verification”.

- If you are able to connect to the MySQL server, but get an Access denied message whenever you issue a SELECT ... INTO OUTFILE or LOAD DATA INFILE statement, your entry in the user table does not have the FILE privilege enabled.

- If you change the grant tables directly (for example, by using INSERT, UPDATE, or DELETE statements) and your changes seem to be ignored, remember that you must execute a FLUSH PRIVILEGES statement or a mysqladmin flush-privileges command to cause the server to reload the privilege tables. Otherwise, your changes have no effect until the next time the server is restarted. Remember that after you change the root password with an UPDATE statement, you will not need to specify the new password until after you flush the privileges, because the server will not know you've changed the password yet!

- If your privileges seem to have changed in the middle of a session, it may be that a MySQL administrator has changed them. Reloading the grant tables affects new client connections, but it also affects existing connections as indicated in Section 6.2.6, “When Privilege Changes Take Effect”.

- If you have access problems with a Perl, PHP, Python, or ODBC program, try to connect to the server with `mysql -u user_name db_name` or `mysql -u user_name -pyour_pass db_name`. If you are able to connect using the `mysql` client, the problem lies with your program, not with the access privileges. (There is no space between `-p` and the password; you can also use the `--password=your_pass` syntax to specify the password. If you use the `-p` or `--password` option with no password value, MySQL prompts you for the password.)

- For testing purposes, start the `mysqld` server with the `--skip-grant-tables` option. Then you can change the MySQL grant tables and use the `SHOW GRANTS` statement to check whether your modifications have the desired effect. When you are satisfied with your changes, execute `mysqladmin flush-privileges` to tell the `mysqld` server to reload the privileges. This enables you to begin using the new grant table contents without stopping and restarting the server.

- If everything else fails, start the `mysqld` server with a debugging option (for example, `--debug=d,general,query`). This prints host and user information about attempted connections, as well as information about each command issued. See Section 22.4.3, “The DBUG Package”.

- If you have any other problems with the MySQL grant tables and feel you must post the problem to the mailing list, always provide a dump of the MySQL grant tables. You can dump the tables with the `mysqldump mysql` command. To file a bug report, see the instructions at Section 1.7, “How to Report Bugs or Problems”. In some cases, you may need to restart `mysqld` with `--skip-grant-tables` to run `mysqldump`.

6.3 MySQL User Account Management

This section describes how to set up accounts for clients of your MySQL server. It discusses the following topics:

- The meaning of account names and passwords as used in MySQL and how that compares to names and passwords used by your operating system

- How to set up new accounts and remove existing accounts

- How to change passwords

- Guidelines for using passwords securely

- How to use secure connections with SSL

See also Section 13.7.1, “Account Management Statements”, which describes the syntax and use for all user-management SQL statements.

6.3.1 User Names and Passwords

MySQL stores accounts in the `user` table of the `mysql` database. An account is defined in terms of a user name and the client host or hosts from which the user can connect to the server. The account may also have a password. For information about account representation in the `user` table, see Section 6.2.2, “Privilege System Grant Tables”. MySQL 5.7 supports authentication plugins, so it is possible that an account authenticates using some external authentication method. See Section 6.3.8, “Pluggable Authentication”.

There are several distinctions between the way user names and passwords are used by MySQL and the way they are used by your operating system:

- User names, as used by MySQL for authentication purposes, have nothing to do with user names (login names) as used by Windows or Unix. On Unix, most MySQL clients by default try to log in using the current Unix user name as the MySQL user name, but that is for convenience only. The default can be overridden easily, because client programs permit any user name to be specified with a `-u` or `--user` option. Because this means that anyone can attempt to connect to the server using any user name, you cannot make a database secure in any way unless all MySQL accounts have passwords. Anyone who specifies a user name for an account that has no password is able to connect successfully to the server.

- MySQL user names can be up to 16 characters long. Operating system user names, because they are completely unrelated to MySQL user names, may be of a different maximum length. For example, Unix user names typically are limited to eight characters.

  Warning

  The limit on MySQL user name length is hard-coded in the MySQL servers and clients, and trying to circumvent it by modifying the definitions of the tables in the `mysql` database *does not work*.

  You should never alter any of the tables in the `mysql` database in any manner whatsoever except by means of the procedure that is described in Section 4.4.7, “`mysql_upgrade` — Check and Upgrade MySQL Tables”. Attempting to redefine MySQL's system tables in any other fashion results in undefined (and unsupported!) behavior.

- The server uses MySQL passwords stored in the `user` table to authenticate client connections using MySQL native authentication (against passwords stored in the `mysql.user` table). These passwords have nothing to do with passwords for logging in to your operating system. There is no necessary connection between the “external” password you use to log in to a Windows or Unix machine and the password you use to access the MySQL server on that machine.

  If the server authenticates a client using a plugin, the authentication method that the plugin implements may or may not use the password in the `user` table. In this case, it is possible that an external password is also used to authenticate to the MySQL server.

- MySQL encrypts passwords stored in the `user` table using its own algorithm. This encryption is the same as that implemented by the `PASSWORD()` SQL function but differs from that used during the Unix login process. Unix password encryption is the same as that implemented by the `ENCRYPT()` SQL function. See the descriptions of the `PASSWORD()` and `ENCRYPT()` functions in Section 12.13, “Encryption and Compression Functions”.

  From version 4.1 on, MySQL employs a stronger authentication method that has better password protection during the connection process than in earlier versions. It is secure even if TCP/IP packets are sniffed or the `mysql` database is captured. (In earlier versions, even though passwords are stored in encrypted form in the `user` table, knowledge of the encrypted password value could be used to connect to the MySQL server.) Section 6.1.2.4, “Password Hashing in MySQL”, discusses password encryption further.

- It is possible to connect to the server regardless of character set settings if the user name and password contain only ASCII characters. To connect when the user name or password contain non-ASCII characters, the client should call the `mysql_options()` C API function with the `MYSQL_SET_CHARSET_NAME` option and appropriate character set name as arguments. This causes authentication to take place using the specified character set. Otherwise, authentication will fail unless the server default character set is the same as the encoding in the authentication defaults.

  Standard MySQL client programs support a `--default-character-set` option that causes `mysql_options()` to be called as just described. In addition, character set autodetection is supported as described in Section 10.1.4, “Connection Character Sets and Collations”. For programs that use a connector that is not based on the C API, the connector may provide an equivalent to `mysql_options()` that can be used instead. Check the connector documentation.

The preceding notes do not apply for ucs2, utf16, and utf32, which are not permitted as client character sets.

When you install MySQL, the grant tables are populated with an initial set of accounts. The names and access privileges for these accounts are described in Section 2.9.2, “Securing the Initial MySQL Accounts”, which also discusses how to assign passwords to them. Thereafter, you normally set up, modify, and remove MySQL accounts using statements such as CREATE USER, GRANT, and REVOKE. See Section 13.7.1, “Account Management Statements”.

When you connect to a MySQL server with a command-line client, specify the user name and password as necessary for the account that you want to use:

```
shell> mysql --user=monty --password=password db_name
```

If you prefer short options, the command looks like this:

```
shell> mysql -u monty -ppassword db_name
```

There must be *no space* between the -p option and the following password value.

If you omit the *password* value following the --password or -p option on the command line, the client prompts for one.

Specifying a password on the command line should be considered insecure. See Section 6.1.2.1, “End-User Guidelines for Password Security”. You can use an option file to avoid giving the password on the command line.

For additional information about specifying user names, passwords, and other connection parameters, see Section 4.2.2, “Connecting to the MySQL Server”.

6.3.2 Adding User Accounts

You can create MySQL accounts in two ways:

- By using statements intended for creating accounts, such as CREATE USER or GRANT. These statements cause the server to make appropriate modifications to the grant tables.
- By manipulating the MySQL grant tables directly with statements such as INSERT, UPDATE, or DELETE.

The preferred method is to use account-creation statements because they are more concise and less error-prone than manipulating the grant tables directly. CREATE USER and GRANT are described in Section 13.7.1, “Account Management Statements”.

Another option for creating accounts is to use the GUI tool MySQL Workbench. Or one of several available third-party programs that offer capabilities for MySQL account administration. phpMyAdmin is one such program.

The following examples show how to use the mysql client program to set up new accounts. These examples assume that privileges have been set up according to the defaults described in Section 2.9.2, “Securing the Initial MySQL Accounts”. This means that to make changes, you must connect to the MySQL server as the MySQL root user, and the root account must have the INSERT privilege for the mysql database and the RELOAD administrative privilege.

As noted in the examples where appropriate, some of the statements will fail if the server's SQL mode has been set to enable certain restrictions. In particular, strict mode (STRICT_TRANS_TABLES, STRICT_ALL_TABLES) and NO_AUTO_CREATE_USER will prevent the server from accepting some of the statements. Workarounds are indicated for these cases. For more information about SQL modes and their effect on grant table manipulation, see Section 5.1.7, “Server SQL Modes”, and Section 13.7.1.4, “GRANT Syntax”.

First, use the mysql program to connect to the server as the MySQL root user:

```
shell> mysql --user=root mysql
```

If you have assigned a password to the root account, you will also need to supply a --password or -p option, both for this mysql command and for those later in this section.

After connecting to the server as root, you can add new accounts. The following statements use GRANT to set up four new accounts:

```
mysql> CREATE USER 'monty'@'localhost' IDENTIFIED BY 'some_pass';
mysql> GRANT ALL PRIVILEGES ON *.* TO 'monty'@'localhost'
    ->     WITH GRANT OPTION;
mysql> CREATE USER 'monty'@'%' IDENTIFIED BY 'some_pass';
mysql> GRANT ALL PRIVILEGES ON *.* TO 'monty'@'%'
    ->     WITH GRANT OPTION;
mysql> CREATE USER 'admin'@'localhost';
mysql> GRANT RELOAD,PROCESS ON *.* TO 'admin'@'localhost';
mysql> CREATE USER 'dummy'@'localhost';
```

The accounts created by these statements have the following properties:

- Two of the accounts have a user name of monty and a password of some_pass. Both accounts are superuser accounts with full privileges to do anything. The 'monty'@'localhost' account can be used only when connecting from the local host. The 'monty'@'%' account uses the '%' wildcard for the host part, so it can be used to connect from any host.

  It is necessary to have both accounts for monty to be able to connect from anywhere as monty. Without the localhost account, the anonymous-user account for localhost that is created by mysql_install_db would take precedence when monty connects from the local host. As a result, monty would be treated as an anonymous user. The reason for this is that the anonymous-user account has a more specific Host column value than the 'monty'@'%' account and thus comes earlier in the user table sort order. (user table sorting is discussed in Section 6.2.4, “Access Control, Stage 1: Connection Verification”.)

- The 'admin'@'localhost' account has no password. This account can be used only by admin to connect from the local host. It is granted the RELOAD and PROCESS administrative privileges. These privileges enable the admin user to execute the mysqladmin reload, mysqladmin refresh, and mysqladmin flush-*xxx* commands, as well as mysqladmin processlist . No privileges are granted for accessing any databases. You could add such privileges later by issuing other GRANT statements.

- The 'dummy'@'localhost' account has no password. This account can be used only to connect from the local host. No privileges are granted. It is assumed that you will grant specific privileges to the account later.

The statements that create accounts with no password will fail if the NO_AUTO_CREATE_USER SQL mode is enabled. To deal with this, use an IDENTIFIED BY clause that specifies a nonempty password.

To check the privileges for an account, use SHOW GRANTS:

```
mysql> SHOW GRANTS FOR 'admin'@'localhost';
+-----------------------------------------------------+
| Grants for admin@localhost                          |
+-----------------------------------------------------+
| GRANT RELOAD, PROCESS ON *.* TO 'admin'@'localhost' |
+-----------------------------------------------------+
```

As an alternative to CREATE USER and GRANT, you can create the same accounts directly by issuing INSERT statements and then telling the server to reload the grant tables using FLUSH PRIVILEGES:

```
shell> mysql --user=root mysql
mysql> INSERT INTO user
    ->     VALUES('localhost','monty',PASSWORD('some_pass'),
    ->     'Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y');
mysql> INSERT INTO user
    ->     VALUES('%','monty',PASSWORD('some_pass'),
    ->     'Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y',
    ->     'Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y',
    ->     '','','','',0,0,0,0);
mysql> INSERT INTO user SET Host='localhost',User='admin',
    ->     Reload_priv='Y', Process_priv='Y';
mysql> INSERT INTO user (Host,User,Password)
    ->     VALUES('localhost','dummy','');
mysql> FLUSH PRIVILEGES;
```

When you create accounts with INSERT, it is necessary to use FLUSH PRIVILEGES to tell the server to reload the grant tables. Otherwise, the changes go unnoticed until you restart the server. With CREATE USER, FLUSH PRIVILEGES is unnecessary.

The reason for using the PASSWORD() function with INSERT is to encrypt the password. The CREATE USER statement encrypts the password for you, so PASSWORD() is unnecessary.

The 'Y' values enable privileges for the accounts. Depending on your MySQL version, you may have to use a different number of 'Y' values in the first two INSERT statements. The INSERT statement for the admin account employs the more readable extended INSERT syntax using SET.

In the INSERT statement for the dummy account, only the Host, User, and Password columns in the user table row are assigned values. None of the privilege columns are set explicitly, so MySQL assigns them all the default value of 'N'. This is equivalent to what CREATE USER does.

If strict SQL mode is enabled, all columns that have no default value must have a value specified. In this case, INSERT statements must explicitly specify values for the ssl_cipher, x509_issuer, and x509_subject columns.

To set up a superuser account, it is necessary only to insert a user table row with all privilege columns set to 'Y'. The user table privileges are global, so no entries in any of the other grant tables are needed.

The next examples create three accounts and give them access to specific databases. Each of them has a user name of custom and password of obscure.

To create the accounts with CREATE USER and GRANT, use the following statements:

```
shell> mysql --user=root mysql
mysql> CREATE USER 'custom'@'localhost' IDENTIFIED BY 'obscure';
mysql> GRANT SELECT,INSERT,UPDATE,DELETE,CREATE,DROP
    ->     ON bankaccount.*
    ->     TO 'custom'@'localhost';
mysql> CREATE USER 'custom'@'host47.example.com' IDENTIFIED BY 'obscure';
mysql> GRANT SELECT,INSERT,UPDATE,DELETE,CREATE,DROP
    ->     ON expenses.*
    ->     TO 'custom'@'host47.example.com';
mysql> CREATE USER 'custom'@'server.domain' IDENTIFIED BY 'obscure';
mysql> GRANT SELECT,INSERT,UPDATE,DELETE,CREATE,DROP
    ->     ON customer.*
    ->     TO 'custom'@'server.domain';
```

The three accounts can be used as follows:

- The first account can access the bankaccount database, but only from the local host.
- The second account can access the expenses database, but only from the host host47.example.com.
- The third account can access the customer database, but only from the host server.domain.

To set up the custom accounts without GRANT, use INSERT statements as follows to modify the grant tables directly:

```
shell> mysql --user=root mysql
mysql> INSERT INTO user (Host,User,Password)
    ->     VALUES('localhost','custom',PASSWORD('obscure'));
mysql> INSERT INTO user (Host,User,Password)
    ->     VALUES('host47.example.com','custom',PASSWORD('obscure'));
mysql> INSERT INTO user (Host,User,Password)
    ->     VALUES('server.domain','custom',PASSWORD('obscure'));
mysql> INSERT INTO db
    ->     (Host,Db,User,Select_priv,Insert_priv,
    ->     Update_priv,Delete_priv,Create_priv,Drop_priv)
    ->     VALUES('localhost','bankaccount','custom',
    ->     'Y','Y','Y','Y','Y','Y');
mysql> INSERT INTO db
    ->     (Host,Db,User,Select_priv,Insert_priv,
    ->     Update_priv,Delete_priv,Create_priv,Drop_priv)
    ->     VALUES('host47.example.com','expenses','custom',
    ->     'Y','Y','Y','Y','Y','Y');
mysql> INSERT INTO db
    ->     (Host,Db,User,Select_priv,Insert_priv,
    ->     Update_priv,Delete_priv,Create_priv,Drop_priv)
    ->     VALUES('server.domain','customer','custom',
    ->     'Y','Y','Y','Y','Y','Y');
mysql> FLUSH PRIVILEGES;
```

The first three INSERT statements add user table entries that permit the user custom to connect from the various hosts with the given password, but grant no global privileges (all privileges are set to the default value of 'N'). The next three INSERT statements add db table entries that grant privileges to custom for the bankaccount, expenses, and customer databases, but only when accessed from the proper hosts. As usual when you modify the grant tables directly, you must tell the server to reload them with FLUSH PRIVILEGES so that the privilege changes take effect.

To create a user who has access from all machines in a given domain (for example, mydomain.com), you can use the “%” wildcard character in the host part of the account name:

```
mysql> CREATE USER 'myname'@'%.mydomain.com' IDENTIFIED BY 'mypass';
```

To do the same thing by modifying the grant tables directly, do this:

```
mysql> INSERT INTO user (Host,User,Password,...)
    ->     VALUES('%.mydomain.com','myname',PASSWORD('mypass'),...);
mysql> FLUSH PRIVILEGES;
```

6.3.3 Removing User Accounts

To remove an account, use the DROP USER statement, which is described in Section 13.7.1.3, “DROP USER Syntax”.

6.3.4 Setting Account Resource Limits

One means of limiting use of MySQL server resources is to set the global max_user_connections system variable to a nonzero value. This limits the number of simultaneous connections that can be made by any given account, but places no limits on what a client can do once connected. In addition, setting max_user_connections does not enable management of individual accounts. Both types of control are of interest to many MySQL administrators, particularly those working for Internet Service Providers.

In MySQL 5.7, you can limit use of the following server resources for individual accounts:

- The number of queries that an account can issue per hour
- The number of updates that an account can issue per hour
- The number of times an account can connect to the server per hour
- The number of simultaneous connections to the server by an account

Any statement that a client can issue counts against the query limit (unless its results are served from the query cache). Only statements that modify databases or tables count against the update limit.

An “account” in this context corresponds to a row in the mysql.user table. That is, a connection is assessed against the User and Host values in the user table row that applies to the connection. For example, an account 'usera'@'%.example.com' corresponds to a row in the user table that has User and Host values of usera and %.example.com, to permit usera to connect from any host in the example.com domain. In this case, the server applies resource limits in this row collectively to all connections by usera from any host in the example.com domain because all such connections use the same account.

Before MySQL 5.0.3, an “account” was assessed against the actual host from which a user connects. This older method accounting may be selected by starting the server with the --old-style-user-limits option. In this case, if usera connects simultaneously from host1.example.com and host2.example.com, the server applies the account resource limits separately to each connection. If usera connects again from host1.example.com, the server applies the limits for that connection together with the existing connection from that host.

To set resource limits for an account, use the GRANT statement (see Section 13.7.1.4, “GRANT Syntax”). Provide a WITH clause that names each resource to be limited. The default value for each limit is zero (no limit). For example, to create a new account that can access the customer database, but only in a limited fashion, issue these statements:

```
mysql> CREATE USER 'francis'@'localhost' IDENTIFIED BY 'frank';
mysql> GRANT ALL ON customer.* TO 'francis'@'localhost'
    ->     WITH MAX_QUERIES_PER_HOUR 20
    ->          MAX_UPDATES_PER_HOUR 10
    ->          MAX_CONNECTIONS_PER_HOUR 5
    ->          MAX_USER_CONNECTIONS 2;
```

The limit types need not all be named in the WITH clause, but those named can be present in any order. The value for each per-hour limit should be an integer representing a count per hour. For MAX_USER_CONNECTIONS, the limit is an integer representing the maximum number of simultaneous connections by the account. If this limit is set to zero, the global max_user_connections system variable value determines the number of simultaneous connections. If max_user_connections is also zero, there is no limit for the account.

To modify existing limits for an account, use a GRANT USAGE statement at the global level (ON *.*). The following statement changes the query limit for francis to 100:

```
mysql> GRANT USAGE ON *.* TO 'francis'@'localhost'
```

```
    ->     WITH MAX_QUERIES_PER_HOUR 100;
```

The statement modifies only the limit value specified and leaves the account otherwise unchanged.

To remove a limit, set its value to zero. For example, to remove the limit on how many times per hour francis can connect, use this statement:

```
mysql> GRANT USAGE ON *.* TO 'francis'@'localhost'
    ->     WITH MAX_CONNECTIONS_PER_HOUR 0;
```

As mentioned previously, the simultaneous-connection limit for an account is determined from the MAX_USER_CONNECTIONS limit and the max_user_connections system variable. Suppose that the global max_user_connections value is 10 and three accounts have resource limits specified with GRANT:

```
GRANT ... TO 'user1'@'localhost' WITH MAX_USER_CONNECTIONS 0;
GRANT ... TO 'user2'@'localhost' WITH MAX_USER_CONNECTIONS 5;
GRANT ... TO 'user3'@'localhost' WITH MAX_USER_CONNECTIONS 20;
```

user1 has a connection limit of 10 (the global max_user_connections value) because it has a zero MAX_USER_CONNECTIONS limit). user2 and user3 have connection limits of 5 and 20, respectively, because they have nonzero MAX_USER_CONNECTIONS limits.

The server stores resource limits for an account in the user table row corresponding to the account. The max_questions, max_updates, and max_connections columns store the per-hour limits, and the max_user_connections column stores the MAX_USER_CONNECTIONS limit. (See Section 6.2.2, “Privilege System Grant Tables”.)

Resource-use counting takes place when any account has a nonzero limit placed on its use of any of the resources.

As the server runs, it counts the number of times each account uses resources. If an account reaches its limit on number of connections within the last hour, further connections for the account are rejected until that hour is up. Similarly, if the account reaches its limit on the number of queries or updates, further queries or updates are rejected until the hour is up. In all such cases, an appropriate error message is issued.

Resource counting is done per account, not per client. For example, if your account has a query limit of 50, you cannot increase your limit to 100 by making two simultaneous client connections to the server. Queries issued on both connections are counted together.

The current per-hour resource-use counts can be reset globally for all accounts, or individually for a given account:

- To reset the current counts to zero for all accounts, issue a FLUSH USER_RESOURCES statement. The counts also can be reset by reloading the grant tables (for example, with a FLUSH PRIVILEGES statement or a mysqladmin reload command).

- The counts for an individual account can be set to zero by re-granting it any of its limits. To do this, use GRANT USAGE as described earlier and specify a limit value equal to the value that the account currently has.

Counter resets do not affect the MAX_USER_CONNECTIONS limit.

All counts begin at zero when the server starts; counts are not carried over through a restart.

For the MAX_USER_CONNECTIONS limit, an edge case can occur if the account currently has open the maximum number of connections permitted to it: A disconnect followed quickly by a connect can result in an error (ER_TOO_MANY_USER_CONNECTIONS or ER_USER_LIMIT_REACHED) if the server has not fully processed the disconnect by the time the connect occurs. When the server finishes disconnect processing, another connection will once more be permitted.

6.3.5 Assigning Account Passwords

Required credentials for clients that connect to the MySQL server can include a password. This section describes how to assign passwords for MySQL accounts. Client authentication occurs using plugins; see Section 6.3.8, “Pluggable Authentication”.

To assign a password when you create a new account with CREATE USER, include an IDENTIFIED BY clause:

```
mysql> CREATE USER 'jeffrey'@'localhost'
    -> IDENTIFIED BY 'mypass';
```

To assign or change a password for an existing account, one way is to issue a SET PASSWORD statement:

```
mysql> SET PASSWORD FOR
    -> 'jeffrey'@'localhost' = PASSWORD('mypass');
```

MySQL stores passwords in the user table in the mysql database. Only users such as root that have the UPDATE privilege for the mysql database can change the password for other users. If you are not connected as an anonymous user, you can change your own password by omitting the FOR clause:

```
mysql> SET PASSWORD = PASSWORD('mypass');
```

The old_passwords system variable value determines the hashing method used by PASSWORD(). If you specify the password using that function and SET PASSWORD rejects the password as not being in the correct format, it may be necessary to set old_passwords to change the hashing method. For descriptions of the permitted values, see Section 5.1.4, “Server System Variables”.

Enabling the read_only system variable prevents the use of the SET PASSWORD statement by any user not having the SUPER privilege.

You can also use a GRANT USAGE statement at the global level (ON *.*) to assign a password to an account without affecting the account's current privileges:

```
mysql> GRANT USAGE ON *.* TO 'jeffrey'@'localhost'
    -> IDENTIFIED BY 'mypass';
```

To assign a password from the command line, use the mysqladmin command:

```
shell> mysqladmin -u user_name -h host_name password "newpwd"
```

The account for which this command sets the password is the one with a user table row that matches *user_name* in the User column and the client host *from which you connect* in the Host column.

During authentication when a client connects to the server, MySQL treats the password in the user table as an encrypted hash value (the value that PASSWORD() would return for the password). When assigning a password to an account, it is important to store an encrypted value, not the plaintext password. Use the following guidelines:

- When you assign a password using CREATE USER, GRANT with an IDENTIFIED BY clause, or the mysqladmin password command, they encrypt the password for you. Specify the literal plaintext password:

```
mysql> CREATE USER 'jeffrey'@'localhost'
    -> IDENTIFIED BY 'mypass';
```

- For CREATE USER or GRANT, you can avoid sending the plaintext password if you know the hash value that PASSWORD() would return for the password. Specify the hash value preceded by the keyword PASSWORD:

```
mysql> CREATE USER 'jeffrey'@'localhost'
    -> IDENTIFIED BY PASSWORD '*90E462C37378CED12064BB3388827D2BA3A9B689';
```

- When you assign an account a nonempty password using SET PASSWORD, you must use the PASSWORD() function to encrypt the password, otherwise the password is stored as plaintext. Suppose that you assign a password like this:

```
mysql> SET PASSWORD FOR
    -> 'jeffrey'@'localhost' = 'mypass';
```

  The result is that the literal value 'mypass' is stored as the password in the user table, not the encrypted value. When jeffrey attempts to connect to the server using this password, the value is encrypted and compared to the value stored in the user table. However, the stored value is the literal string 'mypass', so the comparison fails and the server rejects the connection with an Access denied error.

Note

PASSWORD() encryption differs from Unix password encryption. See Section 6.3.1, “User Names and Passwords”.

It is preferable to assign passwords using SET PASSWORD, GRANT, or mysqladmin, but it is also possible to modify the user table directly. In this case, you must also use FLUSH PRIVILEGES to cause the server to reread the grant tables. Otherwise, the change remains unnoticed by the server until you restart it.

6.3.6 Password Expiration Policy

MySQL enables database administrators to expire account passwords manually, and to establish a policy for automatic password expiration.

To expire a password manually, the database administratior uses the ALTER USER statement:

```
ALTER USER 'jeffrey'@'localhost' PASSWORD EXPIRE;
```

This operation marks the password expired in the corresponding mysql.user table row.

Automatic password expiration is available in MySQL 5.7.4 and later. The mysql.user table indicates for each account when its password was last changed, and the server automatically the server treats the password as expired at client connection time if it is past its permitted lifetime. This works with no explicit manual password expiration.

The default_password_lifetime system variable defines the global automatic password expiration policy. It applies to accounts that use MySQL built-in authentication methods (accounts that use an authentication plugin of mysql_native_password, mysql_old_password, or sha256_password).

The default global policy is that passwords have a lifetime of 360 days. To change the policy, change the value of The default_password_lifetime. If the value is a positive integer, it indicates the permitted password lifetime in days. A value of 0 disables automatic expiration.

Examples:

- To establish a global policy that passwords have a lifetime of approximately six months, start the server with these lines in an option file:

```
[mysqld]
default_password_lifetime=180
```

- To establish a global policy such that passwords never expire, set `default_password_lifetime` to 0:

```
[mysqld]
default_password_lifetime=0
```

- `default_password_lifetime` can also be changed at runtime (this requires the `SUPER` privilege):

```
SET GLOBAL default_password_lifetime = 180;
```

No matter the global policy, it can be overridden for individual accounts with `ALTER USER`:

- Require the password to be changed every 90 days:

```
ALTER USER 'jeffrey'@'localhost' PASSWORD EXPIRE INTERVAL 90 DAY;
```

- Disable password expiration:

```
ALTER USER 'jeffrey'@'localhost' PASSWORD EXPIRE NEVER;
```

- Defer to the global expiration policy:

```
ALTER USER 'jeffrey'@'localhost' PASSWORD EXPIRE DEFAULT;
```

These `ALTER USER` statements update the corresponding `mysql.user` table row.

When a client successfully connects, the server determines whether the account password is expired:

- The server checks whether the password has been manually expired and, if so, restricts the session.

- Otherwise, the server checks whether the password is past its lifetime according to the automatic password expiration policy. If so, the server considers the password expired and restricts the session.

A restricted session remains that way until the client executes a `SET PASSWORD` statement to change the account password. In restricted mode, operations result in an error until the user issues a `SET PASSWORD` statement to establish a new account password:

```
mysql> SELECT 1;
ERROR 1820 (HY000): You must SET PASSWORD before executing this statement

mysql> SET PASSWORD = PASSWORD('new_password');
Query OK, 0 rows affected (0.01 sec)

mysql> SELECT 1;
+---+
| 1 |
+---+
| 1 |
+---+
1 row in set (0.00 sec)
```

This restricted mode of operation permits SET statements, which is useful if the account password uses a hashing format that requires old_passwords to be set to a value different from its default.

It is also possible for an administrative user to reset the account password, but any existing sessions for the account remain restricted. Clients using the account must disconnect and reconnect before statements can be executed successfully.

Note

It is possible to “reset” a password with SET PASSWORD by setting it to its current value. As a matter of good policy, it is preferable to choose a different password.

6.3.7 Password Expiration and Sandbox Mode

MySQL 5.7 provides a password-expiration capability, to enable database administrators to expire account passwords and require users to reset their password. This section describes how password expiration works.

To expire an account password, use the ALTER USER statement. For example:

```
ALTER USER 'myuser'@'localhost' PASSWORD EXPIRE;
```

This statement modifies the row of the mysql.user table associated with the named account, setting the password_expired column to 'Y'. This does not affect any current connections the account has open. For each subsequent connection that uses the account, the server either disconnects the client or handles the client in “sandbox mode,” in which the server permits the client only those operations necessary to reset the expired password. (The action taken by the server depends on both client and server settings.)

If the server disconnects the client, it returns an ER_MUST_CHANGE_PASSWORD_LOGIN error:

```
shell> mysql -u myuser -p
Password: ******
ERROR 1862 (HY000): Your password has expired. To log in you must
change it using a client that supports expired passwords.
```

If the server puts the client in sandbox mode, these operations are permitted within the client session:

- The client can reset the account password with SET PASSWORD. This modifies the row of the mysql.user table associated with the current account, setting the password_expired column to 'N'. After the password has been reset, the server restores normal access for the session, as well as for subsequent connections that use the account.

  It is possible to “reset” a password by setting it to its current value. As a matter of good policy, it is preferable to choose a different password.

- The client can use SET statements. This might be necessary prior to resetting the password; for example, if the account password uses a hashing format that requires the old_passwords system variable to be set to a value different from its default.

For any operation not permitted within the session, the server returns an ER_MUST_CHANGE_PASSWORD error:

```
mysql> USE test;
ERROR 1820 (HY000): You must SET PASSWORD before executing this statement
```

As mentioned previously, whether the server disconnects an expired-password client or puts it in sandbox mode depends on a combination of client and server settings. The following discussion describes the relevant settings and how they interact.

On the client side, a given client indicates whether it can handle sandbox mode for expired passwords. For clients that use the C client library, there are two ways to do this:

- Pass the MYSQL_OPT_CAN_HANDLE_EXPIRED_PASSWORDS flag to mysql_options() prior to connecting:

  ```
  arg = 1;
  result = mysql_options(mysql,
                         MYSQL_OPT_CAN_HANDLE_EXPIRED_PASSWORDS, &arg);
  ```

- Pass the CLIENT_CAN_HANDLE_EXPIRED_PASSWORDS flag to mysql_real_connect() at connection time:

  ```
  mysql = mysql_real_connect(mysql,
                             host, user, password, "test",
                             port, unix_socket,
                             CLIENT_CAN_HANDLE_EXPIRED_PASSWORDS);
  ```

Other MySQL Connectors have their own conventions for indicating readiness to handle sandbox mode. See the relevant Connector documentation.

On the server side, if a client indicates that it can handle expired passwords, the server puts it in sandbox mode.

If a client does not indicate that it can handle expired passwords (or uses an older version of the client library that cannot so indicate), the server action depends on the value of the disconnect_on_expired_password system variable:

- If disconnect_on_expired_password is enabled (the default), the server disconnects the client with an ER_MUST_CHANGE_PASSWORD_LOGIN error.
- If disconnect_on_expired_password is disabled, the server puts the client in sandbox mode.

The preceding client and server settings apply only for accounts with expired passwords. If a client connects using a nonexpired password, the server handles the client normally.

6.3.8 Pluggable Authentication

When a client connects to the MySQL server, the server uses the user name provided by the client and the client host to select the appropriate account row from the mysql.user table. The server then authenticates the client, determining from the account row which authentication plugin applies for the client:

- If the account row specifies a plugin, the server invokes it to authenticate the user. If the server cannot find the plugin, an error occurs.
- If the account row specifies no plugin name, the server authenticates the account using either the mysql_native_password or mysql_old_password plugin, depending on whether the password hash value in the Password column used native hashing or the older pre-4.1 hashing method. Clients must match the password in the Password column of the account row. As of MySQL 5.7.2, the server requires the plugin value to be nonempty.

The plugin returns a status to the server indicating whether the user is permitted to connect.

Pluggable authentication enables two important capabilities:

- **External authentication:** Pluggable authentication makes it possible for clients to connect to the MySQL server with credentials that are appropriate for authentication methods other than native authentication based on passwords stored in the `mysql.user` table. For example, plugins can be created to use external authentication methods such as PAM, Windows login IDs, LDAP, or Kerberos.

- **Proxy users:** If a user is permitted to connect, an authentication plugin can return to the server a user name different from the name of the connecting user, to indicate that the connecting user is a proxy for another user. While the connection lasts, the proxy user is treated, for purposes of access control, as having the privileges of a different user. In effect, one user impersonates another. For more information, see Section 6.3.10, “Proxy Users”.

Several authentication plugins are available in MySQL:

- Plugins that perform native authentication that matches the password against the `Password` column of the account row. The `mysql_native_password` plugin implements authentication based on the native password hashing method. The `mysql_old_password` plugin implements native authentication based on the older (pre-4.1) password hashing method (and is now deprecated). See Section 6.3.9.1, “The Native Authentication Plugin”, and Section 6.3.9.2, “The “Old” Native Authentication Plugin”. Native authentication using `mysql_native_password` is the default for new accounts, unless the `default_authentication_plugin` system variable is set otherwise.

- A plugin that performs authentication using SHA-256 password hashing. This plugin matches the password against the `authentication_string` column of the account row. This is stronger encryption than that available with native authentication. See Section 6.3.9.4, “The SHA-256 Authentication Plugin”.

- A client-side plugin that sends the password to the server without hashing or encryption. This plugin can be used by server-side plugins that require access to the password exactly as provided by the client user. See Section 6.3.9.5, “The Cleartext Client-Side Authentication Plugin”.

- A plugin that authenticates clients that connect from the local host through the Unix socket file. See Section 6.3.9.6, “The Socket Peer-Credential Authentication Plugin”.

- A test plugin that authenticates using MySQL native authentication. This plugin is intended for testing and development purposes, and as an example of how to write an authentication plugin. See Section 6.3.9.7, “The Test Authentication Plugin”.

> **Note**
>
> For information about current restrictions on the use of pluggable authentication, including which connectors support which plugins, see Section E.9, “Restrictions on Pluggable Authentication”.
>
> Third-party connector developers should read that section to determine the extent to which a connector can take advantage of pluggable authentication capabilities and what steps to take to become more compliant.

If you are interested in writing your own authentication plugins, see Section 22.2.4.9, “Writing Authentication Plugins”.

Authentication Plugin Usage Instructions

This section provides general instructions for installing and using authentication plugins.

In general, pluggable authentication uses corresponding plugins on the server and client sides, so you use a given authentication method like this:

- On the server host, install the library containing the appropriate server plugin, if necessary, so that the server can use it to authenticate client connections. Similarly, on each client host, install the library containing the appropriate client plugin for use by client programs.

- Create MySQL accounts that specify use of the plugin for authentication.

- When a client connects, the server plugin tells the client program which client plugin to use for authentication.

The instructions here use an an example authentication plugin included in MySQL distributions (see Section 6.3.9.7, “The Test Authentication Plugin”). The procedure is similar for other authentication plugins; substitute the appropriate plugin and file names.

The example authentication plugin has these characteristics:

- The server-side plugin name is `test_plugin_server`.

- The client-side plugin name is `auth_test_plugin`.

- Both plugins are located in the shared library object file named `auth_test_plugin.so` in the plugin directory (the directory named by the `plugin_dir` system variable). The file name suffix might differ on your system.

Install and use the example authentication plugin as follows:

1. Make sure that the plugin library is installed on the server and client hosts.

2. Install the server-side test plugin at server startup or at runtime:

   - To install the plugin at startup, use the `--plugin-load` option. With this plugin-loading method, the option must be given each time you start the server. For example, use these lines in a `my.cnf` option file:

     ```
     [mysqld]
     plugin-load=test_plugin_server=auth_test_plugin.so
     ```

   - To install the plugin at runtime, use the `INSTALL PLUGIN` statement:

     ```
     mysql> INSTALL PLUGIN test_plugin_server SONAME 'auth_test_plugin.so';
     ```

     This installs the plugin permanently and need be done only once.

3. Verify that the plugin is installed. For example, use `SHOW PLUGINS`:

   ```
   mysql> SHOW PLUGINS\G
   ...
   *************************** 21. row ***************************
      Name: test_plugin_server
    Status: ACTIVE
      Type: AUTHENTICATION
   Library: auth_test_plugin.so
   License: GPL
   ```

   For other ways to check the plugin, see Section 5.1.8.2, “Obtaining Server Plugin Information”.

4. To specify that a MySQL user must be authenticated using a specific server plugin, name the plugin in the `IDENTIFIED WITH` clause of the `CREATE USER` statement that creates the user:

```
CREATE USER 'testuser'@'localhost' IDENTIFIED WITH test_plugin_server;
```

5. Connect to the server using a client program. The test plugin authenticates the same way as native MySQL authentication, so provide the usual `--user` and `--password` options that you normally use to connect to the server. For example:

```
shell> mysql --user=your_name --password=your_pass
```

For connections by `testuser`, the server sees that the account must be authenticated using the server-side plugin named `test_plugin_server` and communicates to the client program which client-side plugin it must use—in this case, `auth_test_plugin`.

In the case that the account uses the authentication method that is the default for both the server and the client program, the server need not communicate to the client which plugin to use, and a round trip in client/server negotiation can be avoided. Currently this is true for accounts that use native MySQL authentication (`mysql_native_password`).

The `--default-auth=plugin_name` option can be specified on the `mysql` command line to make explicit which client-side plugin the program can expect to use, although the server will override this if the user account requires a different plugin.

If the client program does not find the plugin, specify a `--plugin-dir=dir_name` option to indicate where the plugin is located.

Note

If you start the server with the `--skip-grant-tables` option, authentication plugins are not used even if loaded because the server performs no client authentication and permits any client to connect. Because this is insecure, you might want to use `--skip-grant-tables` in conjunction with `--skip-networking` to prevent remote clients from connecting.

6.3.9 Authentication Plugins Available in MySQL

The following sections describe the authentication plugins available in MySQL.

6.3.9.1 The Native Authentication Plugin

MySQL includes two plugins that implement native authentication; that is, authentication against passwords stored in the `Password` column of the `mysql.user` table. This section describes `mysql_native_password`, which implements authentication against the `mysql.user` table using the native password hashing method. For information about `mysql_old_password`, which implements authentication using the older (pre-4.1) password hashing method, see Section 6.3.9.2, “The “Old” Native Authentication Plugin”. For information about these password hashing methods, see Section 6.1.2.4, “Password Hashing in MySQL”.

The `mysql_native_password` native authentication plugin is backward compatible. Clients older than MySQL 5.5.7 do not support authentication *plugins* but do use the native authentication *protocol*, so they can connect to servers from MySQL 5.5.7 and up.

The following table shows the plugin names on the server and client sides.

Table 6.8 MySQL Native Password Authentication Plugin

Server-side plugin name	`mysql_native_password`

Client-side plugin name	mysql_native_password
Library object file name	None (plugins are built in)

The plugin exists in both client and server form:

- The server-side plugin is built into the server, need not be loaded explicitly, and cannot be disabled by unloading it.
- The client-side plugin is built into the libmysqlclient client library as of MySQL 5.5.7 and available to any program linked against libmysqlclient from that version or newer.
- MySQL client programs use mysql_native_password by default. The --default-auth option can be used to specify the plugin explicitly:

```
shell> mysql --default-auth=mysql_native_password ...
```

If an account row specifies no plugin name, the server authenticates the account using either the mysql_native_password or mysql_old_password plugin, depending on whether the password hash value in the Password column used native hashing or the older pre-4.1 hashing method. Clients must match the password in the Password column of the account row. As of MySQL 5.7.2, the server requires the plugin value to be nonempty.

For general information about pluggable authentication in MySQL, see Section 6.3.8, “Pluggable Authentication”.

6.3.9.2 The “Old” Native Authentication Plugin

MySQL includes two plugins that implement native authentication; that is, authentication against passwords stored in the Password column of the mysql.user table. This section describes mysql_old_password, which implements authentication against the mysql.user table using the older (pre-4.1) password hashing method. For information about mysql_native_password, which implements authentication using the native password hashing method, see Section 6.3.9.1, “The Native Authentication Plugin”. For information about these password hashing methods, see Section 6.1.2.4, “Password Hashing in MySQL”.

> **Note**
>
> Passwords that use the pre-4.1 hashing method are less secure than passwords that use the native password hashing method and should be avoided. Pre-4.1 passwords are deprecated and support for them will be removed in a future MySQL release. For account upgrade instructions, see Section 6.3.9.3, “Migrating Away from Pre-4.1 Password Hashing and the mysql_old_password Plugin”.

The mysql_old_password native authentication plugin is backward compatible. Clients older than MySQL 5.5.7 do not support authentication *plugins* but do use the native authentication *protocol*, so they can connect to servers from MySQL 5.5.7 and up.

The following table shows the plugin names on the server and client sides.

Table 6.9 MySQL “Old” Native Authentication Plugin

Server-side plugin name	mysql_old_password
Client-side plugin name	mysql_old_password
Library object file name	None (plugins are built in)

The plugin exists in both client and server form:

- The server-side plugin is built into the server, need not be loaded explicitly, and cannot be disabled by unloading it.
- The client-side plugin is built into the libmysqlclient client library as of MySQL 5.5.7 and available to any program linked against libmysqlclient from that version or newer.
- MySQL client programs can use the --default-auth option to specify the mysql_old_password plugin explicitly:

```
shell> mysql --default-auth=mysql_old_password ...
```

If an account row specifies no plugin name, the server authenticates the account using either the mysql_native_password or mysql_old_password plugin, depending on whether the password hash value in the Password column used native hashing or the older pre-4.1 hashing method. Clients must match the password in the Password column of the account row. As of MySQL 5.7.2, the server requires the plugin value to be nonempty.

For general information about pluggable authentication in MySQL, see Section 6.3.8, “Pluggable Authentication”.

6.3.9.3 Migrating Away from Pre-4.1 Password Hashing and the mysql_old_password Plugin

The MySQL server authenticates connection attempts for each account listed in the mysql.user table using the authentication plugin named in the plugin column. If the plugin column is empty, the server authenticates as follows:

- Before MySQL 5.7.2, the server uses the mysql_native_password or mysql_old_password plugin implicitly, depending on the format of the password hash in the Password column. If the Password value is empty or a 4.1 password hash (41 characters), the server uses mysql_native_password. If the password value is a pre-4.1 password hash (16 characters), the server uses mysql_old_password. (For additional information about these hash formats, see Section 6.1.2.4, “Password Hashing in MySQL”.)
- As of MySQL 5.7.2, the server requires the plugin column to be nonempty and disables accounts that have an empty plugin value.

Pre-4.1 password hashes and the mysql_old_password plugin are deprecated as of MySQL 5.6.5; they provide a level of security inferior to that offered by 4.1 password hashing and the mysql_native_password plugin. Coupled with the requirement in MySQL 5.7.2 that the plugin column must be nonempty, DBAs are advised to upgrade accounts as follows:

- Upgrade accounts that use mysql_native_password implicitly to use it explicitly
- Upgrade accounts that use mysql_old_password (either implicitly or explicitly) to use mysql_native_password explicitly

The instructions in this section describe how perform those upgrades. The result is that no account has an empty plugin value and no account uses pre-4.1 password hashing or the mysql_old_password plugin.

As a variant on these instructions, DBAs might offer users the choice to upgrade to the sha256_password plugin, which authenticates using SHA-256 password hashes. For information about this plugin, see Section 6.3.9.4, “The SHA-256 Authentication Plugin”.

The following table lists the types of mysql.user accounts considered in this discussion.

plugin Column	Password Column	Authentication Result	Upgrade Action
Empty	Empty	Implicitly uses mysql_native_password	Assign plugin
Empty	4.1 hash	Implicitly uses mysql_native_password	Assign plugin
Empty	Pre-4.1 hash	Implicitly uses mysql_old_password	Assign plugin, rehash password
mysql_native_password	Empty	Explicitly uses mysql_native_password	None
mysql_native_password	4.1 hash	Explicitly uses mysql_native_password	None
mysql_old_password	Empty	Explicitly uses mysql_old_password	Upgrade plugin
mysql_old_password	Pre-4.1 hash	Explicitly uses mysql_old_password	Upgrade plugin, rehash password

Accounts corresponding to lines for the mysql_native_password plugin require no upgrade action (because no change of plugin or hash format is required). For accounts corresponding to lines for which the password is empty, the DBA should consider asking the account owners to choose a password (or require it by expiring empty account passwords with ALTER USER).

Upgrading Accounts from Implicit to Explicit mysql_native_password Use

Accounts that have an empty plugin and a 4.1 password hash use mysql_native_password implicitly. To upgrade these accounts to use mysql_native_password explicitly, the DBA should execute these statements:

```
UPDATE mysql.user SET plugin = 'mysql_native_password'
WHERE plugin = '' AND (Password = '' OR LENGTH(Password) = 41);
FLUSH PRIVILEGES;
```

In MySQL 5.7.2 or later, the DBA can run mysql_upgrade, which does the same thing among its upgrade actions. Before 5.7.2, the DBA can execute those statements to uprade accounts proactively.

Notes:

- This step is safe to execute at any time because it makes the mysql_native_password plugin explicit only for accounts that use it implicitly already.
- This step requires no password changes, so the DBA can take this action without affecting users or requiring them to be involved in the upgrade process.

Upgrading Accounts from mysql_old_password to mysql_native_password

Accounts that use mysql_old_password (either implicitly or explicitly) should be upgraded to use mysql_native_password explicitly. This requires changing the plugin *and* changing the password from pre-4.1 to 4.1 hash format.

For the accounts covered in this step that must be upgraded, one of these conditions is true:

- The account uses mysql_old_password implicitly because the plugin column is empty and the password has the pre-4.1 hash format (16 characters).
- The account uses mysql_old_password explicitly.

To identify such accounts, use this query:

```
SELECT User, Host, Password FROM mysql.user
WHERE (plugin = '' AND LENGTH(Password) = 16)
OR plugin = 'mysql_old_password';
```

The following discussion provides two methods for updating that set of accounts. They have differing characteristics, so DBAs should read both and decide which is most suitable for a given MySQL installation.

Method 1.

Characteristics of this method:

- Requires that server and clients be run with `secure_auth=0` until all users have been upgraded to `mysql_native_password`. (Otherwise, users cannot connect to the server using their old-format password hashes for the purpose of upgrading to a new-format hash.)
- Works for MySQL 5.5 through 5.7.1. As of 5.7.2, it does not work because the server requires accounts to have a nonempty plugin and disables them otherwise. Therefore, if you have already upgraded to 5.7.2 or later, choose Method 2.

The DBA should ensure that the server is running with `secure_auth=0`.

For all accounts that use `mysql_old_password` explicitly, the DBA should set them to the empty plugin:

```
UPDATE mysql.user SET plugin = ''
WHERE plugin = 'mysql_old_password';
FLUSH PRIVILEGES;
```

To also expire the password for affected accounts, use these statements instead:

```
UPDATE mysql.user SET plugin = '', password_expired = 'Y'
WHERE plugin = 'mysql_old_password';
FLUSH PRIVILEGES;
```

Now affected users can connect to the server and reset their password to use 4.1 hashing. The DBA should ask each user who now has an empty plugin to connect and execute these statements:

```
SET old_passwords = 0;
SET PASSWORD = PASSWORD('user-chosen-password');
```

Note

In MySQL 5.6.5 or later, the client-side `--secure-auth` is enabled by default, so the DBA should remind users to disable it or they will be unable to connect:

```
shell> mysql -u user_name -p --secure-auth=0
```

After an affected user has executed those statements, the DBA can set the corresponding account plugin to `mysql_native_password` to make the plugin explicit. Or the DBA can periodically run these statements to find and fix any accounts for which affected users have reset their password:

```
UPDATE mysql.user SET plugin = 'mysql_native_password'
WHERE plugin = '' AND (Password = '' OR LENGTH(Password) = 41);
FLUSH PRIVILEGES;
```

When there are no more accounts with an empty plugin, this query returns an empty result:

```
SELECT User, Host, Password FROM mysql.user
WHERE (plugin = '' AND LENGTH(Password) = 16);
```

At that point, all accounts have been migrated away from pre-4.1 password hashing and the server no longer need be run with `secure_auth=0`.

Method 2.

Characteristics of this method:

- The DBA assigns each affected account a new password, so the DBA must tell each such user the new password and ask the user to choose a new one. Communication of the password from DBA to users is outside the scope of MySQL. DBAs should communicate passwords carefully.
- Does not require server or clients to be run with `secure_auth=0`.
- Works for any version of MySQL 5.5 or later.

With this method, the DBA updates each account separately due to the need to set passwords individually. *The DBA should choose a different password for each account.*

Suppose that `'user1'@'localhost'` is one of the accounts to be upgraded. The DBA should modify it like this:

```
SET old_passwords = 0;
UPDATE mysql.user SET plugin = 'mysql_native_password',
Password = PASSWORD('DBA-chosen-password')
WHERE (User, Host) = ('user1', 'localhost');
FLUSH PRIVILEGES;
```

To also expire the password, use these statements instead:

```
SET old_passwords = 0;
UPDATE mysql.user SET plugin = 'mysql_native_password',
Password = PASSWORD('DBA-chosen-password'), password_expired = 'Y'
WHERE (User, Host) = ('user1', 'localhost');
FLUSH PRIVILEGES;
```

Then the DBA should tell the user the new password and ask the user to connect to the server with that password and execute these statements to choose a new password:

```
SET old_passwords = 0;
SET PASSWORD = PASSWORD('user-chosen-password');
```

Repeat for each account to be upgraded.

6.3.9.4 The SHA-256 Authentication Plugin

MySQL provides an authentication plugin that implements SHA-256 hashing for user account passwords.

⚠

Important

To connect to the server using an account that authenticates with the `sha256_password` plugin, you must use either an SSL connection or a plain connection that encrypts the password using RSA, as described later in this section. Either way, use of the `sha256_password` plugin requires that MySQL be built with SSL capabilities. See Section 6.3.11, “Using SSL for Secure Connections”.

The following table shows the plugin names on the server and client sides.

Table 6.10 MySQL SHA-256 Authentication Plugin

Server-side plugin name	sha256_password
Client-side plugin name	sha256_password
Library object file name	None (plugins are built in)

The server-side sha256_password plugin is built into the server, need not be loaded explicitly, and cannot be disabled by unloading it. Similarly, clients need not specify the location of the client-side plugin.

To set up an account that uses SHA-256 password hashing, use the following procedure.

1. Create the account and specify that it authenticates using the sha256_password plugin:

   ```
   CREATE USER 'sha256user'@'localhost' IDENTIFIED WITH sha256_password;
   ```

2. Set the old_passwords system variable to 2 to cause the PASSWORD() function to use SHA-256 hashing of password strings:

   ```
   SET old_passwords = 2;
   ```

3. Set the account password:

   ```
   SET PASSWORD FOR 'sha256user'@'localhost' = PASSWORD('Sh@256Pa33');
   ```

Alternatively, start the server with the default authentication plugin set to sha256_password. For example, put these lines in the server option file:

```
[mysqld]
default-authentication-plugin=sha256_password
```

That causes the sha256_password plugin to be used by default for new accounts and sets old_passwords to 2. As a result, it is possible to set the password at account-creation time using the IDENTIFIED BY clause in the CREATE USER statement:

```
mysql> CREATE USER 'sha256user2'@'localhost' IDENTIFIED BY 'Sh@256Pa33';
Query OK, 0 rows affected (0.06 sec)
```

In this case, the server assigns the sha256_password plugin to the account and encrypts the password using SHA-256. (Another consequence is that to create an account that uses a different authentication plugin, you must specify that plugin using an IDENTIFIED BY clause in the CREATE USER statement, then set old_passwords appropriately for the plugin before using SET PASSWORD to set the account password.)

For more information about old_passwords and PASSWORD(), see Section 5.1.4, “Server System Variables”, and Section 12.13, “Encryption and Compression Functions”.

To change the password for any account that that authenticates using the sha256_password plugin, be sure that the value of old_passwords is 2 before using SET PASSWORD. If old_passwords has a value other than 2, an error occurs for attempts to set the password:

```
mysql> SET old_passwords = 0;
mysql> SET PASSWORD FOR 'sha256user'@'localhost' = PASSWORD('NewSh@256Pa33');
ERROR 1827 (HY000): The password hash doesn't have the expected format.
```

```
Check if the correct password algorithm is being used with the
PASSWORD() function.
```

Accounts in the mysql.user table that use SHA-256 passwords can be identified as rows with 'sha256_password' in the plugin column and a SHA-256 password hash in the authentication_string column.

MySQL can be built with either yaSSL or OpenSSL and the sha256_password plugin works with distributions built using either package. The default is to use yaSSL. If MySQL is built using OpenSSL instead, RSA encryption is available and sha256_password implements the additional capabilities in the following list. (To enable these capabilities, you must also follow the RSA configuration procedure given later in this section.)

- It is possible for the client to transmit passwords to the server using RSA encryption during the client connection process, as described later.
- The server exposes two additional system variables, sha256_password_private_key_path and sha256_password_public_key_path. It is intended that the database administrator will set these to the names of the RSA private and public key files at server startup.
- The server exposes a status variable, Rsa_public_key, that displays the RSA public key value.
- The mysql and mysqltest client programs support a --server-public-key-path option for specifying an RSA public key file explicitly.

For clients that use the sha256_password plugin, passwords are never exposed as cleartext when connecting to the server. How password transmission occurs depends on whether an SSL connection is used and whether RSA encryption is available:

- If an SSL connection is used, the password is sent as cleartext but cannot be snooped because the connection is encrypted using SSL.
- If an SSL connection is not used but RSA encryption is available, the password is sent within an unencrypted connection, but the password is RSA-encrypted to prevent snooping. When the server receives the password, it decrypts it. A scramble is used in the encryption to prevent repeat attacks.
- If an SSL connection is not used and RSA encryption is not available, the sha256_password plugin causes the connection attempt to fail because the password cannot be sent without being exposed as cleartext.

As mentioned previously, RSA password encryption is available only if MySQL was built using OpenSSL. The implication for MySQL distributions built using yaSSL is that SHA-256 passwords can be used only when clients access the server using an SSL connection. For information about connecting to the server using SSL, see Section 6.3.11, “Using SSL for Secure Connections”.

Assuming that MySQL has been built with OpenSSL, the following procedure describes how to enable RSA encryption of passwords during the client connection process:

1. Create the RSA private and public key files. Run these commands while logged into the system account used to run the MySQL server so the files will be owned by that account:

   ```
   openssl genrsa -out mykey.pem 1024
   openssl rsa -in mykey.pem -pubout > mykey.pub
   ```

2. Set the access modes for the key files. The private key should be readable only by the server, whereas the public key can be freely distributed to client users:

   ```
   chmod 400 mykey.pem
   ```

```
chmod 444 mykey.pub
```

3. In the server option file, configure the appropriate system variables with the names of the key files. If you place the files in the server data directory, you need not specify their full path names:

   ```
   [mysqld]
   sha256_password_private_key_path=mykey.pem
   sha256_password_public_key_path=mykey.pub
   ```

   If the files are not in the data directory, or to make their locations explicit in the option values, use full path names:

   ```
   [mysqld]
   sha256_password_private_key_path=/usr/local/mysql/mykey.pem
   sha256_password_public_key_path=/usr/local/mysql/mykey.pub
   ```

4. Restart the server, then connect to it and check the `Rsa_public_key` status variable value. The value will differ from that shown here, but should be nonempty:

   ```
   mysql> SHOW STATUS LIKE 'Rsa_public_key'\G
   *************************** 1. row ***************************
   Variable_name: Rsa_public_key
           Value: -----BEGIN PUBLIC KEY-----
   MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDO9nRUDd+KvSZgY7cNBZMNpwX6
   MvE1PbJFXO7u18nJ9lwc99Du/E7lw6CVXw7VKrXPeHbVQUzGyUNkf45Nz/ckaaJa
   aLgJOBCIDmNVnyU54OT/1lcs2xiyfaDMe8fCJ64ZwTnKbY2gkt1IMjUAB5Ogd5kJ
   g8aV7EtKwyhHb0c30QIDAQAB
   -----END PUBLIC KEY-----
   ```

   If the value is empty, the server found some problem with the key files. Check the error log for diagnostic information.

After the server has been configured with the RSA key files, clients have the option of using them to connect to the server using accounts that authenticate with the `sha256_password` plugin. As mentioned previously, such accounts can use either an SSL connection (in which case RSA is not used) or a plain connection that encrypts the password using RSA. Assume for the following discussion that SSL is not used. Connecting to the server involves no special preparation on the client side. For example:

```
shell> mysql -u sha256user -p
Enter password: Sh@256Pa33
```

For connection attempts by `sha256user`, the server determines that `sha256_password` is the appropriate authentication plugin and invokes it. The plugin finds that the connection does not use SSL and thus requires the password to be transmitted using RSA encryption. It sends the RSA public key to the client, which uses it to encrypt the password and returns the result to the server. The plugin uses the RSA key on the server side to decrypt the password and accepts or rejects the connection based on whether the password is correct.

The server sends the public key to the client as needed, but if a copy of the RSA public key is available on the client host, the client can use it to save a round trip in the client/server protocol:

```
shell> mysql -u sha256user -p --server-public-key-path=file_name
```

The public key value in the file named by the `--server-public-key-path` option should be the same as the key value in the server-side file named by the `sha256_password_public_key_path` system variable. If the key file contains a valid public key value but the value is incorrect, an access-denied error occurs. If the key file does not contain a valid public key, the client program cannot use it. In this case, the server sends the public key to the client as if no `--server-public-key-path` option had been specified.

Client users can get the RSA public key two ways:

- The database administrator can provide a copy of the public key file.
- A client user who can connect to the server some other way can use a `SHOW STATUS LIKE 'Rsa_public_key'` statement and save the returned key value in a file.

6.3.9.5 The Cleartext Client-Side Authentication Plugin

A client-side authentication plugin is available that sends the password to the server without hashing or encryption. This plugin is built into the MySQL client library.

The following table shows the plugin name.

Table 6.11 MySQL Cleartext Authentication Plugin

Server-side plugin name	None, see discussion
Client-side plugin name	`mysql_clear_password`
Library object file name	None (plugin is built in)

With native MySQL authentication, the client performs one-way hashing on the password before sending it to the server. This enables the client to avoid sending the password in clear text. See Section 6.1.2.4, “Password Hashing in MySQL”. However, because the hash algorithm is one way, the original password cannot be recovered on the server side.

One-way hashing cannot be done for authentication schemes that require the server to receive the password as entered on the client side. In such cases, the `mysql_clear_password` client-side plugin can be used to send the password to the server in clear text. There is no corresponding server-side plugin. Rather, the client-side plugin can be used by any server-side plugin that needs a clear text password. (The PAM authentication plugin is one such; see The PAM Authentication Plugin.)

For general information about pluggable authentication in MySQL, see Section 6.3.8, “Pluggable Authentication”.

Note

Sending passwords in clear text may be a security problem in some configurations. To avoid problems if there is any possibility that the password would be intercepted, clients should connect to MySQL Server using a method that protects the password. Possibilities include SSL (see Section 6.3.11, “Using SSL for Secure Connections”), IPsec, or a private network.

To make inadvertent use of this plugin less likely, it is required that clients explicitly enable it. This can be done several ways:

- Set the `LIBMYSQL_ENABLE_CLEARTEXT_PLUGIN` environment variable to a value that begins with `1`, `Y`, or `y`. This enables the plugin for all client connections.
- The `mysql`, `mysqladmin`, and `mysqlslap` client programs support an `--enable-cleartext-plugin` option that enables the plugin on a per-invocation basis.
- The `mysql_options()` C API function supports a `MYSQL_ENABLE_CLEARTEXT_PLUGIN` option that enables the plugin on a per-connection basis. Also, any program that uses `libmysqlclient` and reads option files can enable the plugin by including an `enable-cleartext-plugin` option in an option group read by the client library.

6.3.9.6 The Socket Peer-Credential Authentication Plugin

A server-side authentication plugin is available that authenticates clients that connect from the local host through the Unix socket file.

The source code for this plugin can be examined as a relatively simple example demonstrating how to write a loadable authentication plugin.

The following table shows the plugin and library file names. The file name suffix might differ on your system. The file location is the directory named by the plugin_dir system variable. For installation information, see Section 6.3.8, “Pluggable Authentication”.

Table 6.12 MySQL Socket Peer-Credential Authentication Plugin

Server-side plugin name	auth_socket
Client-side plugin name	None, see discussion
Library object file name	auth_socket.so

The auth_socket authentication plugin authenticates clients that connect from the local host through the Unix socket file. The plugin uses the SO_PEERCRED socket option to obtain information about the user running the client program. The plugin checks whether the user name matches the MySQL user name specified by the client program to the server, and permits the connection only if the names match. The plugin can be built only on systems that support the SO_PEERCRED option, such as Linux.

Suppose that a MySQL account is created for a user named valerie who is to be authenticated by the auth_socket plugin for connections from the local host through the socket file:

```
CREATE USER 'valerie'@'localhost' IDENTIFIED WITH auth_socket;
```

If a user on the local host with a login name of stefanie invokes mysql with the option --user=valerie to connect through the socket file, the server uses auth_socket to authenticate the client. The plugin determines that the --user option value (valerie) differs from the client user's name (stephanie) and refuses the connection. If a user named valerie tries the same thing, the plugin finds that the user name and the MySQL user name are both valerie and permits the connection. However, the plugin refuses the connection even for valerie if the connection is made using a different protocol, such as TCP/IP.

For general information about pluggable authentication in MySQL, see Section 6.3.8, “Pluggable Authentication”.

6.3.9.7 The Test Authentication Plugin

MySQL includes a test plugin that authenticates using MySQL native authentication, but is a loadable plugin (not built in) and must be installed prior to use. It can authenticate against either normal or older (shorter) password hash values.

This plugin is intended for testing and development purposes, and not for use in production environments. The test plugin source code is separate from the server source, unlike the built-in native plugin, so it can be examined as a relatively simple example demonstrating how to write a loadable authentication plugin.

The following table shows the plugin and library file names. The file name suffix might differ on your system. The file location is the directory named by the plugin_dir system variable. For installation information, see Section 6.3.8, “Pluggable Authentication”.

Table 6.13 MySQL Test Authentication Plugin

Server-side plugin name	test_plugin_server

Client-side plugin name	`auth_test_plugin`
Library object file name	`auth_test_plugin.so`

Because the test plugin authenticates the same way as native MySQL authentication, provide the usual `--user` and `--password` options that you normally use for accounts that use native authentication when you connect to the server. For example:

```
shell> mysql --user=your_name --password=your_pass
```

For general information about pluggable authentication in MySQL, see Section 6.3.8, “Pluggable Authentication”.

6.3.10 Proxy Users

When authentication to the MySQL server occurs by means of an authentication plugin, the plugin may request that the connecting (external) user be treated as a different user for privilege-checking purposes. This enables the external user to be a proxy for the second user; that is, to have the privileges of the second user. In other words, the external user is a “proxy user” (a user who can impersonate or become known as another user) and the second user is a “proxied user” (a user whose identity can be taken on by a proxy user).

This section describes how the proxy user capability works. For general information about authentication plugins, see Section 6.3.8, “Pluggable Authentication”. If you are interested in writing your own authentication plugins that support proxy users, see Implementing Proxy User Support in Authentication Plugins.

For proxying to occur, these conditions must be satisfied:

- When a connecting client should be treated as a proxy user, the plugin must return a different name, to indicate the proxied user name.
- A proxy user account must be set up to be authenticated by the plugin. Use the `CREATE USER` or `GRANT` statement to associate an account with a plugin.
- A proxy user account must have the `PROXY` privilege for the proxied account. Use the `GRANT` statement for this.

Consider the following definitions:

```
CREATE USER 'empl_external'@'localhost'
  IDENTIFIED WITH auth_plugin AS 'auth_string';
CREATE USER 'employee'@'localhost'
  IDENTIFIED BY 'employee_pass';
GRANT PROXY
  ON 'employee'@'localhost'
  TO 'empl_external'@'localhost';
```

When a client connects as `empl_external` from the local host, MySQL uses `auth_plugin` to perform authentication. If `auth_plugin` returns the `employee` user name to the server (based on the content of `'auth_string'` and perhaps by consulting some external authentication system), that serves as a request to the server to treat this client, for purposes of privilege checking, as the `employee` local user.

In this case, `empl_external` is the proxy user and `employee` is the proxied user.

The server verifies that proxy authentication for `employee` is possible for the `empl_external` user by checking whether `empl_external` has the `PROXY` privilege for `employee`. (If this privilege had not been granted, an error would occur.)

When proxying occurs, the USER() and CURRENT_USER() functions can be used to see the difference between the connecting user and the account whose privileges apply during the current session. For the example just described, those functions return these values:

```
mysql> SELECT USER(), CURRENT_USER();
+-------------------------+--------------------+
| USER()                  | CURRENT_USER()     |
+-------------------------+--------------------+
| empl_external@localhost | employee@localhost |
+-------------------------+--------------------+
```

The IDENTIFIED WITH clause that names the authentication plugin may be followed by an AS clause specifying a string that the server passes to the plugin when the user connects. It is up to each plugin whether the AS clause is required. If it is required, the format of the authentication string depends on how the plugin intends to use it. Consult the documentation for a given plugin for information about the authentication string values it accepts.

Granting the Proxy Privilege

A special PROXY privilege is needed to enable an external user to connect as and have the privileges of another user. To grant this privilege, use the GRANT statement. For example:

```
GRANT PROXY ON 'proxied_user' TO 'proxy_user';
```

proxy_user must represent a valid externally authenticated MySQL user at connection time or connection attempts fail. *proxied_user* must represent a valid locally authenticated user at connection time or connection attempts fail.

The corresponding REVOKE syntax is:

```
REVOKE PROXY ON 'proxied_user' FROM 'proxy_user';
```

MySQL GRANT and REVOKE syntax extensions work as usual. For example:

```
GRANT PROXY ON 'a' TO 'b', 'c', 'd';
GRANT PROXY ON 'a' TO 'd' IDENTIFIED BY ...;
GRANT PROXY ON 'a' TO 'd' WITH GRANT OPTION;
GRANT PROXY ON 'a' TO ''@'';
REVOKE PROXY ON 'a' FROM 'b', 'c', 'd';
```

In the preceding example, ''@'' is the default proxy user and means “any user.” The default proxy user is discussed later in this section.

The PROXY privilege can be granted in these cases:

- By *proxied_user* for itself: The value of USER() must exactly match CURRENT_USER() and *proxied_user*, for both the user name and host name parts of the account name.
- By a user that has GRANT PROXY ... WITH GRANT OPTION for *proxied_user*.

The root account created by default during MySQL installation has the PROXY ... WITH GRANT OPTION privilege for ''@'', that is, for all users. This enables root to set up proxy users, as well as to delegate to other accounts the authority to set up proxy users. For example, root can do this:

```
CREATE USER 'admin'@'localhost' IDENTIFIED BY 'test';
GRANT PROXY ON ''@'' TO 'admin'@'localhost' WITH GRANT OPTION;
```

Now the admin user can manage all the specific GRANT PROXY mappings. For example, admin can do this:

```
GRANT PROXY ON sally TO joe;
```

Default Proxy Users

To specify that some or all users should connect using a given external plugin, create a “blank” MySQL user, set it up to use that plugin for authentication, and let the plugin return the real authenticated user name (if different from the blank user). For example, suppose that there exists a hypothetical plugin named ldap_auth that implements LDAP authentication:

```
CREATE USER ''@'' IDENTIFIED WITH ldap_auth AS 'O=Oracle, OU=MySQL';
CREATE USER 'developer'@'localhost' IDENTIFIED BY 'developer_pass';
CREATE USER 'manager'@'localhost' IDENTIFIED BY 'manager_pass';
GRANT PROXY ON 'manager'@'localhost' TO ''@'';
GRANT PROXY ON 'developer'@'localhost' TO ''@'';
```

Now assume that a client tries to connect as follows:

```
mysql --user=myuser --password='myuser_pass' ...
```

The server will not find myuser defined as a MySQL user. But because there is a blank user account (''@''), that matches the client user name and host name, the server authenticates the client against that account: The server invokes ldap_auth, passing it myuser and myuser_pass as the user name and password.

If the ldap_auth plugin finds in the LDAP directory that myuser_pass is not the correct password for myuser, authentication fails and the server rejects the connection.

If the password is correct and ldap_auth finds that myuser is a developer, it returns the user name developer to the MySQL server, rather than myuser. The server verifies that ''@'' can authenticate as developer (because it has the PROXY privilege to do so) and accepts the connection. The session proceeds with myuser having the privileges of developer. (These privileges should be set up by the DBA using GRANT statements, not shown.) The USER() and CURRENT_USER() functions return these values:

```
mysql> SELECT USER(), CURRENT_USER();
+------------------+---------------------+
| USER()           | CURRENT_USER()      |
+------------------+---------------------+
| myuser@localhost | developer@localhost |
+------------------+---------------------+
```

If the plugin instead finds in the LDAP directory that myuser is a manager, it returns manager as the user name and the session proceeds with myuser having the privileges of manager.

```
mysql> SELECT USER(), CURRENT_USER();
+------------------+-------------------+
| USER()           | CURRENT_USER()    |
+------------------+-------------------+
| myuser@localhost | manager@localhost |
+------------------+-------------------+
```

For simplicity, external authentication cannot be multilevel: Neither the credentials for developer nor those for manager are taken into account in the preceding example. However, they are still used if a client tries to authenticate directly against the developer or manager account, which is why those accounts should be assigned passwords.

The default proxy account uses `''` in the host part, which matches any host. If you set up a default proxy user, take care to also check for accounts with `'%'` in the host part, because that also matches any host, but has precedence over `''` by the rules that the server uses to sort account rows internally (see Section 6.2.4, “Access Control, Stage 1: Connection Verification”).

Suppose that a MySQL installation includes these two accounts:

```
CREATE USER ''@'' IDENTIFIED WITH some_plugin;
CREATE USER ''@'%' IDENTIFIED BY 'some_password';
```

The intent of the first account is to serve as the default proxy user, to be used to authenticate connections for users who do not otherwise match a more-specific account. The second account might have been created, for example, to enable users without their own account as the anonymous user.

However, in this configuration, the first account will never be used because the matching rules sort `''@'%'` ahead of `''@''`. For accounts that do not match any more-specific account, the server will attempt to authenticate them against `''@'%'` rather than `''@''`.

If you intend to create a default proxy user, check for other existing “match any user” accounts that will take precedence over the default proxy user and thus prevent that user from working as intended. It may be necessary to remove any such accounts.

Proxy User System Variables

Two system variables help trace the proxy login process:

- `proxy_user`: This value is `NULL` if proxying is not used. Otherwise, it indicates the proxy user account. For example, if a client authenticates through the default proxy account, this variable will be set as follows:

  ```
  mysql> SELECT @@proxy_user;
  +--------------+
  | @@proxy_user |
  +--------------+
  | ''@''        |
  +--------------+
  ```

- `external_user`: Sometimes the authentication plugin may use an external user to authenticate to the MySQL server. For example, when using Windows native authentication, a plugin that authenticates using the windows API does not need the login ID passed to it. However, it still uses an Windows user ID to authenticate. The plugin may return this external user ID (or the first 512 UTF-8 bytes of it) to the server using the `external_user` read-only session variable. If the plugin does not set this variable, its value is `NULL`.

6.3.11 Using SSL for Secure Connections

MySQL supports secure (encrypted) connections between MySQL clients and the server using the Secure Sockets Layer (SSL) protocol. This section discusses how to use SSL connections. For information on how to require users to use SSL connections, see the discussion of the `REQUIRE` clause of the `GRANT` statement in Section 13.7.1.4, “`GRANT` Syntax”.

The standard configuration of MySQL is intended to be as fast as possible, so encrypted connections are not used by default. For applications that require the security provided by encrypted connections, the extra computation to encrypt the data is worthwhile.

MySQL enables encryption on a per-connection basis. You can choose an unencrypted connection or a secure encrypted SSL connection according the requirements of individual applications.

Secure connections are based on the OpenSSL API and are available through the MySQL C API. Replication uses the C API, so secure connections can be used between master and slave servers. See Section 16.3.7, “Setting Up Replication Using SSL”.

Another way to connect securely is from within an SSH connection to the MySQL server host. For an example, see Section 6.3.12, “Connecting to MySQL Remotely from Windows with SSH”.

6.3.11.1 Basic SSL Concepts

To understand how MySQL uses SSL, it is necessary to explain some basic SSL and X509 concepts. People who are familiar with these concepts can skip this part of the discussion.

By default, MySQL uses unencrypted connections between the client and the server. This means that someone with access to the network could watch all your traffic and look at the data being sent or received. They could even change the data while it is in transit between client and server. To improve security a little, you can compress client/server traffic by using the `--compress` option when invoking client programs. However, this does not foil a determined attacker.

When you need to move information over a network in a secure fashion, an unencrypted connection is unacceptable. Encryption is the way to make any kind of data unreadable. Encryption algorithms must include security elements to resist many kinds of known attacks such as changing the order of encrypted messages or replaying data twice.

SSL is a protocol that uses different encryption algorithms to ensure that data received over a public network can be trusted. It has mechanisms to detect any data change, loss, or replay. SSL also incorporates algorithms that provide identity verification using the X509 standard.

X509 makes it possible to identify someone on the Internet. It is most commonly used in e-commerce applications. In basic terms, there should be some entity called a “Certificate Authority” (or CA) that assigns electronic certificates to anyone who needs them. Certificates rely on asymmetric encryption algorithms that have two encryption keys (a public key and a secret key). A certificate owner can show the certificate to another party as proof of identity. A certificate consists of its owner's public key. Any data encrypted with this public key can be decrypted only using the corresponding secret key, which is held by the owner of the certificate.

For more information about SSL, X509, encryption, or public-key cryptography, perform an Internet search for the keywords in which you are interested.

6.3.11.2 Configuring MySQL for SSL

To use SSL connections between the MySQL server and client programs, your system must support either OpenSSL or yaSSL, and your version of MySQL must be built with SSL support. To make it easier to use secure connections, MySQL is bundled with yaSSL, which uses the same licensing model as MySQL. (OpenSSL uses an Apache-style license.) yaSSL support is available on all MySQL platforms supported by Oracle Corporation.

To get secure connections to work with MySQL and SSL, you must do the following:

1. If you are not using a binary (precompiled) version of MySQL that has been built with SSL support, and you are going to use OpenSSL rather than the bundled yaSSL library, install OpenSSL if it has not already been installed. We have tested MySQL with OpenSSL 1.0.0. To obtain OpenSSL, visit http://www.openssl.org.

   Building MySQL using OpenSSL requires a shared OpenSSL library, otherwise linker errors occur. Alternatively, build MySQL using yaSSL.

2. If you are not using a binary (precompiled) version of MySQL that has been built with SSL support, configure a MySQL source distribution to use SSL. When you configure MySQL, invoke CMake like this:

   ```
   shell> cmake . -DWITH_SSL=bundled
   ```

   That command configures the distribution to use the bundled yaSSL library. To use the system SSL library instead, specify the option like this instead:

   ```
   shell> cmake . -DWITH_SSL=system
   ```

   See Section 2.8.4, “MySQL Source-Configuration Options”.

   Then compile and install the distribution.

   On Unix platforms, yaSSL retrieves true random numbers from either /dev/urandom or /dev/random. Bug#13164 lists workarounds for some very old platforms which do not support these devices.

3. To check whether a mysqld server supports SSL, examine the value of the have_ssl system variable:

   ```
   mysql> SHOW VARIABLES LIKE 'have_ssl';
   +---------------+-------+
   | Variable_name | Value |
   +---------------+-------+
   | have_ssl      | YES   |
   +---------------+-------+
   ```

   If the value is YES, the server supports SSL connections. If the value is DISABLED, the server is capable of supporting SSL connections but was not started with the appropriate --ssl-*xxx* options to enable them to be used; see Section 6.3.11.3, “Using SSL Connections”.

6.3.11.3 Using SSL Connections

To enable SSL connections, your MySQL distribution must be built with SSL support, as described in Section 6.3.11.2, “Configuring MySQL for SSL”. In addition, the proper SSL-related options must be used to specify the appropriate certificate and key files. For a complete list of SSL options, see Section 6.3.11.4, “SSL Command Options”.

To start the MySQL server so that it permits clients to connect using SSL, use the options that identify the certificate and key files the server uses when establishing a secure connection:

- --ssl-ca identifies the Certificate Authority (CA) certificate.
- --ssl-cert identifies the server public key certificate. This can be sent to the client and authenticated against the CA certificate that it has.
- --ssl-key identifies the server private key.

For example, start the server like this:

```
shell> mysqld --ssl-ca=ca-cert.pem \
         --ssl-cert=server-cert.pem \
         --ssl-key=server-key.pem
```

Each option names a file in PEM format. For instructions on generating the required SSL certificate and key files, see Section 6.3.11.5, “Setting Up SSL Certificates and Keys for MySQL”. If you have a MySQL source distribution, you can also test your setup using the demonstration certificate and key files in the `mysql-test/std_data` directory of the distribution.

Similar options are used on the client side, but `--ssl-cert` and `--ssl-key` identify the client public and private key. The Certificate Authority certificate, if specified, must be the same as used by the server.

To establish a secure connection to a MySQL server with SSL support, the options that a client must specify depend on the SSL requirements of the MySQL account used by the client. (See the discussion of the `REQUIRE` clause in Section 13.7.1.4, "`GRANT` Syntax".)

Suppose that you want to connect using an account that has no special SSL requirements or was created using a `GRANT` statement that includes the `REQUIRE SSL` option. As a recommended set of SSL options, start the server with at least `--ssl-cert` and `--ssl-key`, and invoke the client with `--ssl-ca`. A client can connect securely like this:

```
shell> mysql --ssl-ca=ca-cert.pem
```

To require that a client certificate also be specified, create the account using the `REQUIRE X509` option. Then the client must also specify the proper client key and certificate files or the server will reject the connection:

```
shell> mysql --ssl-ca=ca-cert.pem \
       --ssl-cert=client-cert.pem \
       --ssl-key=client-key.pem
```

To prevent use of SSL and override other SSL options, invoke the client program with `--ssl=0` or a synonym (`--skip-ssl`, `--disable-ssl`):

```
shell> mysql --ssl=0
```

A client can determine whether the current connection with the server uses SSL by checking the value of the `Ssl_cipher` status variable. The value is nonempty if SSL is used, and empty otherwise. For example:

```
mysql> SHOW STATUS LIKE 'Ssl_cipher';
+---------------+--------------------+
| Variable_name | Value              |
+---------------+--------------------+
| Ssl_cipher    | DHE-RSA-AES256-SHA |
+---------------+--------------------+
```

For the `mysql` client, an alternative is to use the `STATUS` or `\s` command and check the `SSL` line:

```
mysql> \s
...
SSL: Cipher in use is DHE-RSA-AES256-SHA
...
```

Or:

```
mysql> \s
...
SSL: Not in use
...
```

The C API enables application programs to use SSL:

- To establish a secure connection, use the `mysql_ssl_set()` C API function to set the appropriate certificate options before calling `mysql_real_connect()`. See Section 21.8.7.72, “`mysql_ssl_set()`”. To require the use of SSL, call `mysql_options()` with the `MYSQL_OPT_SSL_ENFORCE` option.
- To determine whether SSL is in use after the connection is established, use `mysql_get_ssl_cipher()`. A non-`NULL` return value indicates a secure connection and names the SSL cipher used for encryption. A `NULL` return value indicates that SSL is not being used. See Section 21.8.7.34, “`mysql_get_ssl_cipher()`”.

Replication uses the C API, so secure connections can be used between master and slave servers. See Section 16.3.7, “Setting Up Replication Using SSL”.

6.3.11.4 SSL Command Options

This section describes options that specify whether to use SSL and the names of SSL certificate and key files. These options can be given on the command line or in an option file. For examples of suggested use and how to check whether a connection is secure, see Section 6.3.11.3, “Using SSL Connections”.

Table 6.14 SSL Option/Variable Summary

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
have_openssl			Yes		Global	No
have_ssl			Yes		Global	No
skip-ssl	Yes	Yes				
ssl	Yes	Yes				
ssl-ca	Yes	Yes			Global	No
- *Variable*: ssl_ca			Yes		Global	No
ssl-capath	Yes	Yes			Global	No
- *Variable*: ssl_capath			Yes		Global	No
ssl-cert	Yes	Yes			Global	No
- *Variable*: ssl_cert			Yes		Global	No
ssl-cipher	Yes	Yes			Global	No
- *Variable*: ssl_cipher			Yes		Global	No
ssl-crl	Yes	Yes			Global	No
- *Variable*: ssl_crl			Yes		Global	No
ssl-crlpath	Yes	Yes			Global	No
- *Variable*: ssl_crlpath			Yes		Global	No
ssl-key	Yes	Yes			Global	No
- *Variable*: ssl_key			Yes		Global	No
ssl-verify-server-cert	Yes	Yes				

- `--ssl`

For the server, this option specifies that the server permits but does not require SSL connections.

For clients, the option meaning is version specific:

- As of MySQL 5.7.3, `--ssl` requires the client to connect to the server using SSL. If an encrypted connection cannot be established, the connection attempt fails. If the connection attempt succeeds, the connection is guaranteed to use SSL.
- Before MySQL 5.7.3, `--ssl` permits but does not require the client to connect to the server using SSL. Therefore, this option is not sufficient in itself to cause an SSL connection to be used. For example, if you specify this option for a client program but the server has not been configured to permit SSL connections, an unencrypted connection is used.

  `--ssl` is implied by other `--ssl-xxx` options, as indicated in the descriptions for those options.

If other `--ssl-xxx` options are given in the absence of `--ssl`, the client attempts to connect using SSL. If the server is not configured to permit SSL, the client falls back to an unencrypted connection.

As a recommended set of options to enable SSL connections, use at least `--ssl-cert` and `--ssl-key` on the server side and `--ssl-ca` on the client side. See Section 6.3.11.3, “Using SSL Connections”.

The `--ssl` option can be given in its negated form to override other SSL options and indicate that SSL should *not* be used. To do this, specify the option as `--ssl=0` or a synonym (`--skip-ssl`, `--disable-ssl`). For example, you might have SSL options specified in the `[client]` group of your option file to use SSL connections by default when you invoke MySQL client programs. To use an unencrypted connection instead, invoke the client program with `--skip-ssl` on the command line to override the options in the option file.

To require use of an SSL connection for a MySQL account, issue a `GRANT` statement for the account that includes at least a `REQUIRE SSL` clause. Connections for the account will be rejected unless MySQL supports SSL connections and the server and client have been started with the proper SSL options.

The `REQUIRE` clause permits other SSL-related options, which can be used to enforce stricter requirements than `REQUIRE SSL`. For additional details about which SSL command options may or must be specified by clients that connect using accounts configured using the various `REQUIRE` options, see the description of `REQUIRE` in Section 13.7.1.4, “GRANT Syntax”.

- `--ssl-ca=file_name`

  The path to a file in PEM format that contains a list of trusted SSL certificate authorities. This option implies `--ssl` when used on the server side, and on the client side before MySQL 5.7.3.

  If you use SSL when establishing a client connection, to tell the client not to authenticate the server certificate, specify neither `--ssl-ca` nor `--ssl-capath`. The server still verifies the client according to any applicable requirements established using `GRANT` statements for the client account, and it still uses any `--ssl-ca` or `--ssl-capath` option values specified at server startup.

- `--ssl-capath=dir_name`

  The path to a directory that contains trusted SSL certificate authority certificates in PEM format. This option implies `--ssl` when used on the server side, and on the client side before MySQL 5.7.3.

  If you use SSL when establishing a client connection, to tell the client not to authenticate the server certificate, specify neither `--ssl-ca` nor `--ssl-capath`. The server still verifies the client according to any applicable requirements established using GRANT statements for the client account, and it still uses any --ssl-ca or --ssl-capath option values specified at server startup.

MySQL distributions built with OpenSSL support the --ssl-capath option. Distributions built with yaSSL do not because yaSSL does not look in any directory and does not follow a chained certificate tree. yaSSL requires that all components of the CA certificate tree be contained within a single CA certificate tree and that each certificate in the file has a unique SubjectName value. To work around this yaSSL limitation, concatenate the individual certificate files comprising the certificate tree into a new file and specify that file as the value of the --ssl-ca option.

- --ssl-cert=*file_name*

  The name of the SSL certificate file in PEM format to use for establishing a secure connection. This option implies --ssl when used on the server side, and on the client side before MySQL 5.7.3.

- --ssl-cipher=*cipher_list*

  A list of permissible ciphers to use for SSL encryption. If no cipher in the list is supported, SSL connections will not work. This option implies --ssl when used on the server side, and on the client side before MySQL 5.7.3.

  For greatest portability, *cipher_list* should be a list of one or more cipher names, separated by colons. This format is understood both by OpenSSL and yaSSL. Examples:

```
--ssl-cipher=AES128-SHA
--ssl-cipher=DHE-RSA-AES256-SHA:AES128-SHA
```

  OpenSSL supports a more flexible syntax for specifying ciphers, as described in the OpenSSL documentation at http://www.openssl.org/docs/apps/ciphers.html. However, yaSSL does not, so attempts to use that extended syntax fail for a MySQL distribution built with yaSSL.

  For OpenSSL, the supported ciphers may depend on which version your server is linked against. For example, the list might include these ciphers:

```
AES256-GCM-SHA384
AES256-SHA
AES256-SHA256
CAMELLIA256-SHA
DES-CBC3-SHA
DHE-DSS-AES256-GCM-SHA384
DHE-DSS-AES256-SHA
DHE-DSS-AES256-SHA256
DHE-DSS-CAMELLIA256-SHA
DHE-RSA-AES256-GCM-SHA384
DHE-RSA-AES256-SHA
DHE-RSA-AES256-SHA256
DHE-RSA-CAMELLIA256-SHA
ECDH-ECDSA-AES256-GCM-SHA384
ECDH-ECDSA-AES256-SHA
ECDH-ECDSA-AES256-SHA384
ECDH-ECDSA-DES-CBC3-SHA
ECDH-RSA-AES256-GCM-SHA384
ECDH-RSA-AES256-SHA
ECDH-RSA-AES256-SHA384
ECDH-RSA-DES-CBC3-SHA
ECDHE-ECDSA-AES128-GCM-SHA256
ECDHE-ECDSA-AES128-SHA
ECDHE-ECDSA-AES128-SHA256
ECDHE-ECDSA-AES256-GCM-SHA384
ECDHE-ECDSA-AES256-SHA
```

```
ECDHE-ECDSA-AES256-SHA384
ECDHE-ECDSA-DES-CBC3-SHA
ECDHE-RSA-AES128-GCM-SHA256
ECDHE-RSA-AES128-SHA
ECDHE-RSA-AES128-SHA256
ECDHE-RSA-AES256-GCM-SHA384
ECDHE-RSA-AES256-SHA
ECDHE-RSA-AES256-SHA384
ECDHE-RSA-DES-CBC3-SHA
EDH-DSS-DES-CBC3-SHA
EDH-RSA-DES-CBC3-SHA
PSK-3DES-EDE-CBC-SHA
PSK-AES256-CBC-SHA
SRP-DSS-3DES-EDE-CBC-SHA
SRP-DSS-AES-128-CBC-SHA
SRP-DSS-AES-256-CBC-SHA
SRP-RSA-3DES-EDE-CBC-SHA
SRP-RSA-AES-128-CBC-S
SRP-RSA-AES-256-CBC-SHA
```

yaSSL supports these ciphers:

```
AES128-RMD
AES128-SHA
AES256-RMD
AES256-SHA
DES-CBC-SHA
DES-CBC3-RMD
DES-CBC3-SHA
DHE-RSA-AES128-RMD
DHE-RSA-AES128-SHA
DHE-RSA-AES256-RMD
DHE-RSA-AES256-SHA
DHE-RSA-DES-CBC3-RMD
EDH-RSA-DES-CBC-SHA
EDH-RSA-DES-CBC3-SHA
RC4-MD5
RC4-SHA
```

To verify exactly which ciphers a given server supports, check the value of the `Ssl_cipher_list` status variable using this query:

```
SHOW STATUS LIKE 'Ssl_cipher_list';
```

- `--ssl-crl=`*`file_name`*

  The path to a file containing certificate revocation lists in PEM format. This option implies `--ssl` when used on the server side, and on the client side before MySQL 5.7.3.

  If neither `--ssl-crl` nor `--ssl-crlpath` is given, no CRL checks are performed, even if the CA path contains certificate revocation lists.

  MySQL distributions built with OpenSSL support the `--ssl-crl` option. Distributions built with yaSSL do not because revocation lists do not work with yaSSL.

- `--ssl-crlpath=`*`dir_name`*

  The path to a directory that contains files containing certificate revocation lists in PEM format. This option implies `--ssl` when used on the server side, and on the client side before MySQL 5.7.3.

If neither `--ssl-crl` nor `--ssl-crlpath` is given, no CRL checks are performed, even if the CA path contains certificate revocation lists.

MySQL distributions built with OpenSSL support the `--ssl-crlpath` option. Distributions built with yaSSL do not because revocation lists do not work with yaSSL.

- `--ssl-key=`*`file_name`*

  The name of the SSL key file in PEM format to use for establishing a secure connection. This option implies `--ssl` when used on the server side, and on the client side before MySQL 5.7.3.

  If the key file is protected by a passphrase, the program prompts the user for the passphrase. The password must be given interactively; it cannot be stored in a file. If the passphrase is incorrect, the program continues as if it could not read the key.

- `--ssl-verify-server-cert`

  This option is available for client programs only, not the server. It causes the client to check the server's Common Name value in the certificate that the server sends to the client. The client verifies that name against the host name the client uses for connecting to the server, and the connection fails if there is a mismatch. This feature can be used to prevent man-in-the-middle attacks. Verification is disabled by default.

6.3.11.5 Setting Up SSL Certificates and Keys for MySQL

This section demonstrates how to set up SSL certificate and key files for use by MySQL servers and clients. The first example shows a simplified procedure such as you might use from the command line. The second shows a script that contains more detail. The first two examples are intended for use on Unix and both use the `openssl` command that is part of OpenSSL. The third example describes how to set up SSL files on Windows.

> **Important**
>
> Whatever method you use to generate the certificate and key files, the Common Name value used for the server and client certificates/keys must each differ from the Common Name value used for the CA certificate. Otherwise, the certificate and key files will not work for servers compiled using OpenSSL. A typical error in this case is:
>
> ```
> ERROR 2026 (HY000): SSL connection error:
> error:00000001:lib(0):func(0):reason(1)
> ```

Example 1: Creating SSL Files from the Command Line on Unix

The following example shows a set of commands to create MySQL server and client certificate and key files. You will need to respond to several prompts by the `openssl` commands. To generate test files, you can press Enter to all prompts. To generate files for production use, you should provide nonempty responses.

```
# Create clean environment
shell> rm -rf newcerts
shell> mkdir newcerts && cd newcerts

# Create CA certificate
shell> openssl genrsa 2048 > ca-key.pem
shell> openssl req -new -x509 -nodes -days 3600 \
```

```
         -key ca-key.pem -out ca-cert.pem

# Create server certificate, remove passphrase, and sign it
# server-cert.pem = public key, server-key.pem = private key
shell> openssl req -newkey rsa:2048 -days 3600 \
         -nodes -keyout server-key.pem -out server-req.pem
shell> openssl rsa -in server-key.pem -out server-key.pem
shell> openssl x509 -req -in server-req.pem -days 3600 \
         -CA ca-cert.pem -CAkey ca-key.pem -set_serial 01 -out server-cert.pem

# Create client certificate, remove passphrase, and sign it
# client-cert.pem = public key, client-key.pem = private key
shell> openssl req -newkey rsa:2048 -days 3600 \
         -nodes -keyout client-key.pem -out client-req.pem
shell> openssl rsa -in client-key.pem -out client-key.pem
shell> openssl x509 -req -in client-req.pem -days 3600 \
         -CA ca-cert.pem -CAkey ca-key.pem -set_serial 01 -out client-cert.pem
```

After generating the certificates, verify them:

```
shell> openssl verify -CAfile ca-cert.pem server-cert.pem client-cert.pem
server-cert.pem: OK
client-cert.pem: OK
```

Now you have a set of files that can be used as follows:

- `ca-cert.pem`: Use this as the argument to `--ssl-ca` on the server and client sides. (The CA certificate, if used, must be the same on both sides.)

- `server-cert.pem`, `server-key.pem`: Use these as the arguments to `--ssl-cert` and `--ssl-key` on the server side.

- `client-cert.pem`, `client-key.pem`: Use these as the arguments to `--ssl-cert` and `--ssl-key` on the client side.

To use the files to test SSL connections, see Section 6.3.11.3, “Using SSL Connections”.

Example 2: Creating SSL Files Using a Script on Unix

Here is an example script that shows how to set up SSL certificate and key files for MySQL. After executing the script, use the files to test SSL connections as described in Section 6.3.11.3, “Using SSL Connections”.

```
DIR=`pwd`/openssl
PRIV=$DIR/private

mkdir $DIR $PRIV $DIR/newcerts
cp /usr/share/ssl/openssl.cnf $DIR
replace ./demoCA $DIR -- $DIR/openssl.cnf

# Create necessary files: $database, $serial and $new_certs_dir
# directory (optional)

touch $DIR/index.txt
echo "01" > $DIR/serial

#
# Generation of Certificate Authority(CA)
#

openssl req -new -x509 -keyout $PRIV/cakey.pem -out $DIR/ca-cert.pem \
    -days 3600 -config $DIR/openssl.cnf
```

```
# Sample output:
# Using configuration from /home/monty/openssl/openssl.cnf
# Generating a 1024 bit RSA private key
# ................++++++
# .........++++++
# writing new private key to '/home/monty/openssl/private/cakey.pem'
# Enter PEM pass phrase:
# Verifying password - Enter PEM pass phrase:
# -----
# You are about to be asked to enter information that will be
# incorporated into your certificate request.
# What you are about to enter is what is called a Distinguished Name
# or a DN.
# There are quite a few fields but you can leave some blank
# For some fields there will be a default value,
# If you enter '.', the field will be left blank.
# -----
# Country Name (2 letter code) [AU]:FI
# State or Province Name (full name) [Some-State]:.
# Locality Name (eg, city) []:
# Organization Name (eg, company) [Internet Widgits Pty Ltd]:MySQL AB
# Organizational Unit Name (eg, section) []:
# Common Name (eg, YOUR name) []:MySQL admin
# Email Address []:

#
# Create server request and key
#
openssl req -new -keyout $DIR/server-key.pem -out \
    $DIR/server-req.pem -days 3600 -config $DIR/openssl.cnf

# Sample output:
# Using configuration from /home/monty/openssl/openssl.cnf
# Generating a 1024 bit RSA private key
# ..++++++
# ..........++++++
# writing new private key to '/home/monty/openssl/server-key.pem'
# Enter PEM pass phrase:
# Verifying password - Enter PEM pass phrase:
# -----
# You are about to be asked to enter information that will be
# incorporated into your certificate request.
# What you are about to enter is what is called a Distinguished Name
# or a DN.
# There are quite a few fields but you can leave some blank
# For some fields there will be a default value,
# If you enter '.', the field will be left blank.
# -----
# Country Name (2 letter code) [AU]:FI
# State or Province Name (full name) [Some-State]:.
# Locality Name (eg, city) []:
# Organization Name (eg, company) [Internet Widgits Pty Ltd]:MySQL AB
# Organizational Unit Name (eg, section) []:
# Common Name (eg, YOUR name) []:MySQL server
# Email Address []:
#
# Please enter the following 'extra' attributes
# to be sent with your certificate request
# A challenge password []:
# An optional company name []:

#
# Remove the passphrase from the key
#
openssl rsa -in $DIR/server-key.pem -out $DIR/server-key.pem

#
```

```
# Sign server cert
#
openssl ca -cert $DIR/ca-cert.pem -policy policy_anything \
    -out $DIR/server-cert.pem -config $DIR/openssl.cnf \
    -infiles $DIR/server-req.pem

# Sample output:
# Using configuration from /home/monty/openssl/openssl.cnf
# Enter PEM pass phrase:
# Check that the request matches the signature
# Signature ok
# The Subjects Distinguished Name is as follows
# countryName           :PRINTABLE:'FI'
# organizationName      :PRINTABLE:'MySQL AB'
# commonName            :PRINTABLE:'MySQL admin'
# Certificate is to be certified until Sep 13 14:22:46 2003 GMT
# (365 days)
# Sign the certificate? [y/n]:y
#
#
# 1 out of 1 certificate requests certified, commit? [y/n]y
# Write out database with 1 new entries
# Data Base Updated

#
# Create client request and key
#
openssl req -new -keyout $DIR/client-key.pem -out \
    $DIR/client-req.pem -days 3600 -config $DIR/openssl.cnf

# Sample output:
# Using configuration from /home/monty/openssl/openssl.cnf
# Generating a 1024 bit RSA private key
# .....................................++++++
# .............................................++++++
# writing new private key to '/home/monty/openssl/client-key.pem'
# Enter PEM pass phrase:
# Verifying password - Enter PEM pass phrase:
# -----
# You are about to be asked to enter information that will be
# incorporated into your certificate request.
# What you are about to enter is what is called a Distinguished Name
# or a DN.
# There are quite a few fields but you can leave some blank
# For some fields there will be a default value,
# If you enter '.', the field will be left blank.
# -----
# Country Name (2 letter code) [AU]:FI
# State or Province Name (full name) [Some-State]:.
# Locality Name (eg, city) []:
# Organization Name (eg, company) [Internet Widgits Pty Ltd]:MySQL AB
# Organizational Unit Name (eg, section) []:
# Common Name (eg, YOUR name) []:MySQL user
# Email Address []:
#
# Please enter the following 'extra' attributes
# to be sent with your certificate request
# A challenge password []:
# An optional company name []:

#
# Remove the passphrase from the key
#
openssl rsa -in $DIR/client-key.pem -out $DIR/client-key.pem

#
# Sign client cert
```

```
#

openssl ca -cert $DIR/ca-cert.pem -policy policy_anything \
    -out $DIR/client-cert.pem -config $DIR/openssl.cnf \
    -infiles $DIR/client-req.pem

# Sample output:
# Using configuration from /home/monty/openssl/openssl.cnf
# Enter PEM pass phrase:
# Check that the request matches the signature
# Signature ok
# The Subjects Distinguished Name is as follows
# countryName           :PRINTABLE:'FI'
# organizationName      :PRINTABLE:'MySQL AB'
# commonName            :PRINTABLE:'MySQL user'
# Certificate is to be certified until Sep 13 16:45:17 2003 GMT
# (365 days)
# Sign the certificate? [y/n]:y
#
#
# 1 out of 1 certificate requests certified, commit? [y/n]y
# Write out database with 1 new entries
# Data Base Updated

#
# Create a my.cnf file that you can use to test the certificates
#

cat <<EOF > $DIR/my.cnf
[client]
ssl-ca=$DIR/ca-cert.pem
ssl-cert=$DIR/client-cert.pem
ssl-key=$DIR/client-key.pem
[mysqld]
ssl-ca=$DIR/ca-cert.pem
ssl-cert=$DIR/server-cert.pem
ssl-key=$DIR/server-key.pem
EOF
```

Example 3: Creating SSL Files on Windows

Download OpenSSL for Windows if it is not installed on your system. An overview of available packages can be seen here:

```
http://www.slproweb.com/products/Win32OpenSSL.html
```

Choose the Win32 OpenSSL Light or Win64 OpenSSL Light package, depending on your architecture (32-bit or 64-bit). The default installation location will be `C:\OpenSSL-Win32` or `C:\OpenSSL-Win64`, depending on which package you downloaded. The following instructions assume a default location of `C:\OpenSSL-Win32`. Modify this as necessary if you are using the 64-bit package.

If a message occurs during setup indicating `'...critical component is missing: Microsoft Visual C++ 2008 Redistributables'`, cancel the setup and download one of the following packages as well, again depending on your architecture (32-bit or 64-bit):

- Visual C++ 2008 Redistributables (x86), available at:

  ```
  http://www.microsoft.com/downloads/details.aspx?familyid=9B2DA534-3E03-4391-8A4D-074B9F2BC1BF
  ```

- Visual C++ 2008 Redistributables (x64), available at:

```
http://www.microsoft.com/downloads/details.aspx?familyid=bd2a6171-e2d6-4230-b809-9a8d7548c1b6
```

After installing the additional package, restart the OpenSSL setup procedure.

During installation, leave the default `C:\OpenSSL-Win32` as the install path, and also leave the default option `'Copy OpenSSL DLL files to the Windows system directory'` selected.

When the installation has finished, add `C:\OpenSSL-Win32\bin` to the Windows System Path variable of your server:

1. On the Windows desktop, right-click the My Computer icon, and select Properties.
2. Select the Advanced tab from the System Properties menu that appears, and click the Environment Variables button.
3. Under **System Variables**, select Path, then click the Edit button. The Edit System Variable dialogue should appear.
4. Add `';C:\OpenSSL-Win32\bin'` to the end (notice the semicolon).
5. Press OK 3 times.
6. Check that OpenSSL was correctly integrated into the Path variable by opening a new command console (`Start>Run>cmd.exe`) and verifying that OpenSSL is available:

   ```
   Microsoft Windows [Version ...]
   Copyright (c) 2006 Microsoft Corporation. All rights reserved.

   C:\Windows\system32>cd \

   C:\>openssl
   OpenSSL> exit <<< If you see the OpenSSL prompt, installation was successful.

   C:\>
   ```

Depending on your version of Windows, the preceding path-setting instructions might differ slightly.

After OpenSSL has been installed, use instructions similar to those from from Example 1 (shown earlier in this section), with the following changes:

- Change the following Unix commands:

  ```
  # Create clean environment
  shell> rm -rf newcerts
  shell> mkdir newcerts && cd newcerts
  ```

  On Windows, use these commands instead:

  ```
  # Create clean environment
  shell> md c:\newcerts
  shell> cd c:\newcerts
  ```

- When a `'\'` character is shown at the end of a command line, this `'\'` character must be removed and the command lines entered all on a single line.

After generating the certificate and key files, to use them to test SSL connections, see Section 6.3.11.3, “Using SSL Connections”.

6.3.12 Connecting to MySQL Remotely from Windows with SSH

This section describes how to get a secure connection to a remote MySQL server with SSH. The information was provided by David Carlson `<dcarlson@mplcomm.com>`.

1. Install an SSH client on your Windows machine. As a user, the best nonfree one I have found is from `SecureCRT` from http://www.vandyke.com/. Another option is `f-secure` from http://www.f-secure.com/. You can also find some free ones on `Google` at http://directory.google.com/Top/Computers/Internet/Protocols/SSH/Clients/Windows/.

2. Start your Windows SSH client. Set `Host_Name =` *`yourmysqlserver_URL_or_IP`*. Set `userid=`*`your_userid`* to log in to your server. This `userid` value might not be the same as the user name of your MySQL account.

3. Set up port forwarding. Either do a remote forward (Set `local_port: 3306`, `remote_host:` *`yourmysqlservername_or_ip`*, `remote_port: 3306`) or a local forward (Set `port: 3306`, `host: localhost`, `remote port: 3306`).

4. Save everything, otherwise you will have to redo it the next time.

5. Log in to your server with the SSH session you just created.

6. On your Windows machine, start some ODBC application (such as Access).

7. Create a new file in Windows and link to MySQL using the ODBC driver the same way you normally do, except type in `localhost` for the MySQL host server, not *`yourmysqlservername`*.

At this point, you should have an ODBC connection to MySQL, encrypted using SSH.

6.3.13 MySQL Enterprise Audit Log Plugin

> **Note**
>
> MySQL Enterprise Audit is a commercial extension. To learn more about commercial products (MySQL Enterprise Edition), see http://www.mysql.com/products/.

In MySQL 5.7, MySQL Enterprise Edition includes MySQL Enterprise Audit, implemented using a server plugin named `audit_log`. MySQL Enterprise Audit uses the open MySQL Audit API to enable standard, policy-based monitoring and logging of connection and query activity executed on specific MySQL servers. Designed to meet the Oracle audit specification, MySQL Enterprise Audit provides an out of box, easy to use auditing and compliance solution for applications that are governed by both internal and external regulatory guidelines.

When installed, the audit plugin enables MySQL Server to produce a log file containing an audit record of server activity. The log contents include when clients connect and disconnect, and what actions they perform while connected, such as which databases and tables they access.

After you install the plugin (see Section 6.3.13.1, “Installing the Audit Log Plugin”), it writes an audit log file. By default, the file is named `audit.log` in the server data directory. To change the name of the file, set the `audit_log_file` system variable at server startup.

Audit log file contents are not encrypted. See Section 6.3.13.2, “Audit Log Plugin Security Considerations”.

The audit log file is written in XML, with auditable events encoded as `<AUDIT_RECORD>` elements. To select the file format, set the `audit_log_format` system variable at server startup. For details on file format and contents, see Section 6.3.13.3, “The Audit Log File”.

To control what information `audit_log` writes to its log file, set the `audit_log_policy` system variable. By default, this variable is set to `ALL` (write all auditable events), but also permits values of `LOGINS` or `QUERIES` to log only login or query events, or `NONE` to disable logging.

For more information about controlling how logging occurs, see Section 6.3.13.4, “Audit Log Plugin Logging Control”. For descriptions of the parameters used to configure the audit log plugin, see Section 6.3.13.5, “Audit Log Plugin Options and Variables”.

The Performance Schema (see Chapter 20, *MySQL Performance Schema*) has instrumentation for the audit log plugin. To identify the relevant instruments, use this query:

```
SELECT NAME FROM performance_schema.setup_instruments
WHERE NAME LIKE '%/alog/%';
```

Changes from Older Audit Log Plugin Versions

Several changes were made to the audit log plugin in MySQL 5.7 for better compatibility with Oracle Audit Vault.

MySQL 5.7 changed audit log file output to a new format. Subsequently, in 5.7.3, it became possible to select either the old or new format using the `audit_log_format` system variable, which has permitted values of `OLD` and `NEW` (default `NEW`). The two formats differ as follows:

- Information within `<AUDIT_RECORD>` elements written in the old format using attributes is written in the new format using subelements.
- The new format includes more information in `<AUDIT_RECORD>` elements. Every element includes a `RECORD_ID` value providing a unique identifier. The `TIMESTAMP` value includes time zone information. Query records include `HOST`, `IP`, `OS_LOGIN`, and `USER` information, as well as `COMMAND_CLASS` and `STATUS_CODE` values.

Example of old `<AUDIT_RECORD>` format:

```
<AUDIT_RECORD
 TIMESTAMP="2013-09-15T15:27:27"
 NAME="Query"
 CONNECTION_ID="3"
 STATUS="0"
 SQLTEXT="SELECT 1"
/>
```

Example of new `<AUDIT_RECORD>` format:

```
<AUDIT_RECORD>
 <TIMESTAMP>2013-09-15T15:27:27 UTC</TIMESTAMP>
 <RECORD_ID>3998_2013-09-15T15:27:27</RECORD_ID>
 <NAME>Query</NAME>
 <CONNECTION_ID>3</CONNECTION_ID>
 <STATUS>0</STATUS>
 <STATUS_CODE>0</STATUS_CODE>
 <USER>root[root] @ localhost [127.0.0.1]</USER>
 <OS_LOGIN></OS_LOGIN>
 <HOST>localhost</HOST>
 <IP>127.0.0.1</IP>
 <COMMAND_CLASS>select</COMMAND_CLASS>
 <SQLTEXT>SELECT 1</SQLTEXT>
</AUDIT_RECORD>
```

When the audit log plugin rotates the audit log file, it uses a different file name format. For a log file named `audit.log`, the plugin previously renamed the file to `audit.log.`*`TIMESTAMP`*. The plugin now renames the file to `audit.log.`*`TIMESTAMP`*`.xml` to indicate that it is an XML file.

If you previously used an older version of the audit log plugin, use this procedure to avoid writing new-format log entries to an existing log file that contains old-format entries:

1. Stop the server.
2. Rename the current audit log file manually. This file will contain only old-format log entries.
3. Update the server and restart it. The audit log plugin will create a new log file, which will contain only new-format log entries.

Similarly, to change the value of audit_log_format, stop the server and rename the current audit log file manually before restarting the server with the new audit_log_format value.

The API for writing audit plugins has also changed. The mysql_event_general structure has new members to represent client host name and IP address, command class, and external user. For more information, see Section 22.2.4.8, “Writing Audit Plugins”.

6.3.13.1 Installing the Audit Log Plugin

The audit log plugin is named audit_log. To be usable by the server, the plugin library object file must be located in the MySQL plugin directory (the directory named by the plugin_dir system variable). If necessary, set the value of plugin_dir at server startup to tell the server the location of the plugin directory.

To load the plugin at server startup, use the --plugin-load option to name the object file that contains the plugin. With this plugin-loading method, the option must be given each time the server starts. For example, put the following lines in your my.cnf file:

```
[mysqld]
plugin-load=audit_log.so
```

If object files have a suffix different from .so on your system, substitute the correct suffix (for example, .dll on Windows).

Alternatively, to register the plugin at runtime, use this statement (changing the suffix as necessary):

```
mysql> INSTALL PLUGIN audit_log SONAME 'audit_log.so';
```

INSTALL PLUGIN loads the plugin, and also registers it in the mysql.plugins table to cause the plugin to be loaded for each subsequent normal server startup.

If the plugin is loaded with --plugin-load or has been previously registered with INSTALL PLUGIN, you can use the --audit-log option at server startup to control plugin activation. For example, to load the plugin and prevent it from being removed at runtime, use these options:

```
[mysqld]
plugin-load=audit_log.so
audit-log=FORCE_PLUS_PERMANENT
```

If it is desired to prevent the server from running without the audit plugin, use --audit-log with a value of FORCE or FORCE_PLUS_PERMANENT to force server startup to fail if the plugin does not initialize successfully.

For general information about installing plugins, see Section 5.1.8, “Server Plugins”. To verify plugin installation, examine the INFORMATION_SCHEMA.PLUGINS table or use the SHOW PLUGINS statement. See Section 5.1.8.2, “Obtaining Server Plugin Information”.

Audit log file contents are not encrypted. See Section 6.3.13.2, “Audit Log Plugin Security Considerations”.

For additional information about the parameters used to configure operation of the audit_log plugin, see Section 6.3.13.5, “Audit Log Plugin Options and Variables”.

6.3.13.2 Audit Log Plugin Security Considerations

Contents of the audit log file produced by the `audit_log` audit log plugin are not encrypted and may contain sensitive information, such as the text of SQL statements. For security reasons, this file should be written to a directory accessible only to the MySQL server and users with a legitimate reason to view the log. The default file is `audit.log` in the data directory. This can be changed by setting the `audit_log_file` system variable at server startup.

6.3.13.3 The Audit Log File

Audit log file contents are not encrypted. See Section 6.3.13.2, “Audit Log Plugin Security Considerations”.

The audit log file is written as XML, using UTF-8 (up to 4 bytes per character). The root element is `<AUDIT>`. The closing `</AUDIT>` tag of the root element is written when the audit log plugin terminates, so the tag is not present in the file while the plugin is active.

The root element contains `<AUDIT_RECORD>` elements, each of which contains other elements that provide information about the audited event.

MySQL 5.7 changed audit log file output to a new format. Subsequently, in 5.7.3, it became possible to select either the old or new format using the `audit_log_format` system variable, which has permitted values of `OLD` and `NEW` (default `NEW`).

If you change the value of `audit_log_format`, use this procedure to avoid writing log entries in one format to an existing log file that contains entries in a different format:

1. Stop the server.

2. Rename the current audit log file manually.

3. Restart the server with the new value of `audit_log_format`. The audit log plugin will create a new log file, which will contain log entries in the selected format.

Here is a sample log file in the default (new) format, reformatted slightly for readability:

```
<?xml version="1.0" encoding="UTF-8"?>
<AUDIT>
 <AUDIT_RECORD>
  <TIMESTAMP>2013-09-17T15:03:24 UTC</TIMESTAMP>
  <RECORD_ID>1_2013-09-17T15:03:24</RECORD_ID>
  <NAME>Audit</NAME>
  <SERVER_ID>1</SERVER_ID>
  <VERSION>1</VERSION>
  <STARTUP_OPTIONS>/usr/local/mysql/bin/mysqld
    --socket=/usr/local/mysql/mysql.sock
    --port=3306</STARTUP_OPTIONS>
  <OS_VERSION>x86_64-osx10.6</OS_VERSION>
  <MYSQL_VERSION>5.7.2-m12-log</MYSQL_VERSION>
 </AUDIT_RECORD>
 <AUDIT_RECORD>
  <TIMESTAMP>2013-09-17T15:03:40 UTC</TIMESTAMP>
  <RECORD_ID>2_2013-09-17T15:03:24</RECORD_ID>
  <NAME>Connect</NAME>
  <CONNECTION_ID>2</CONNECTION_ID>
  <STATUS>0</STATUS>
  <STATUS_CODE>0</STATUS_CODE>
  <USER>root</USER>
  <OS_LOGIN></OS_LOGIN>
  <HOST>localhost</HOST>
  <IP>127.0.0.1</IP>
  <COMMAND_CLASS>connect</COMMAND_CLASS>
  <PRIV_USER>root</PRIV_USER>
  <PROXY_USER></PROXY_USER>
  <DB>test</DB>
 </AUDIT_RECORD>

...

 <AUDIT_RECORD>
  <TIMESTAMP>2013-09-17T15:03:41 UTC</TIMESTAMP>
  <RECORD_ID>4_2013-09-17T15:03:24</RECORD_ID>
  <NAME>Query</NAME>
  <CONNECTION_ID>2</CONNECTION_ID>
  <STATUS>0</STATUS>
  <STATUS_CODE>0</STATUS_CODE>
  <USER>root[root] @ localhost [127.0.0.1]</USER>
  <OS_LOGIN></OS_LOGIN>
  <HOST>localhost</HOST>
  <IP>127.0.0.1</IP>
  <COMMAND_CLASS>drop_table</COMMAND_CLASS>
  <SQLTEXT>DROP TABLE IF EXISTS t</SQLTEXT>
 </AUDIT_RECORD>
 <AUDIT_RECORD>
  <TIMESTAMP>2013-09-17T15:03:41 UTC</TIMESTAMP>
  <RECORD_ID>5_2013-09-17T15:03:24</RECORD_ID>
  <NAME>Query</NAME>
  <CONNECTION_ID>2</CONNECTION_ID>
  <STATUS>0</STATUS>
  <STATUS_CODE>0</STATUS_CODE>
  <USER>root[root] @ localhost [127.0.0.1]</USER>
  <OS_LOGIN></OS_LOGIN>
  <HOST>localhost</HOST>
  <IP>127.0.0.1</IP>
  <COMMAND_CLASS>create_table</COMMAND_CLASS>
  <SQLTEXT>CREATE TABLE t (i INT)</SQLTEXT>
 </AUDIT_RECORD>

...

 <AUDIT_RECORD>
  <TIMESTAMP>2013-09-17T15:03:41 UTC</TIMESTAMP>
  <RECORD_ID>7_2013-09-17T15:03:24</RECORD_ID>
  <NAME>Quit</NAME>
  <CONNECTION_ID>2</CONNECTION_ID>
  <STATUS>0</STATUS>
  <STATUS_CODE>0</STATUS_CODE>
  <USER></USER>
  <OS_LOGIN></OS_LOGIN>
  <HOST></HOST>
  <IP></IP>
  <COMMAND_CLASS>connect</COMMAND_CLASS>
 </AUDIT_RECORD>

...

 <AUDIT_RECORD>
  <TIMESTAMP>2013-09-17T15:03:47 UTC</TIMESTAMP>
  <RECORD_ID>9_2013-09-17T15:03:24</RECORD_ID>
  <NAME>Shutdown</NAME>
  <CONNECTION_ID>3</CONNECTION_ID>
  <STATUS>0</STATUS>
  <STATUS_CODE>0</STATUS_CODE>
  <USER>root[root] @ localhost [127.0.0.1]</USER>
  <OS_LOGIN></OS_LOGIN>
  <HOST>localhost</HOST>
  <IP>127.0.0.1</IP>
  <COMMAND_CLASS></COMMAND_CLASS>
 </AUDIT_RECORD>
 <AUDIT_RECORD>
```

```
  <TIMESTAMP>2013-09-17T15:03:47 UTC</TIMESTAMP>
  <RECORD_ID>10_2013-09-17T15:03:24</RECORD_ID>
  <NAME>Quit</NAME>
  <CONNECTION_ID>3</CONNECTION_ID>
  <STATUS>0</STATUS>
  <STATUS_CODE>0</STATUS_CODE>
  <USER></USER>
  <OS_LOGIN></OS_LOGIN>
  <HOST></HOST>
  <IP></IP>
  <COMMAND_CLASS>connect</COMMAND_CLASS>
 </AUDIT_RECORD>
 <AUDIT_RECORD>
  <TIMESTAMP>2013-09-17T15:03:49 UTC</TIMESTAMP>
  <RECORD_ID>11_2013-09-17T15:03:24</RECORD_ID>
  <NAME>NoAudit</NAME>
  <SERVER_ID>1</SERVER_ID>
 </AUDIT_RECORD>
</AUDIT>
```

Elements within <AUDIT_RECORD> elements have these characteristics:

- Some elements appear in every <AUDIT_RECORD> element, but many are optional and do not necessarily appear in every element.
- Order of elements within an <AUDIT_RECORD> element is not guaranteed.
- Element values are not fixed length. Long values may be truncated as indicated in the element descriptions given later.
- The <, >, ", and & characters are encoded as &lt;, &gt;, &quot;, and &amp;, respectively. NUL bytes (U+00) are encoded as the ? character.
- Characters not valid as XML characters are encoded using numeric character references. Valid XML characters are:

  ```
  #x9 | #xA | #xD | [#x20-#xD7FF] | [#xE000-#xFFFD] | [#x10000-#x10FFFF]
  ```

New Audit Log File Format

Every <AUDIT_RECORD> element contains a set of mandatory elements. Other optional elements may appear, depending on the audit record type.

The following elements are mandatory in every <AUDIT_RECORD> element:

- <NAME>

  A string representing the type of instruction that generated the audit event, such as a command that the server received from a client.

  Example:

  ```
  <NAME>Query</NAME>
  ```

  Some common <NAME> values:

  ```
  Audit    When auditing starts, which may be server startup time
  Connect  When a client connects, also known as logging in
  Query    An SQL statement (executed directly)
  ```

```
Prepare  Preparation of an SQL statement; usually followed by Execute
Execute  Execution of an SQL statement; usually follows Prepare
Shutdown Server shutdown
Quit     When a client disconnects
NoAudit  Auditing has been turned off
```

The possible values are `Audit`, `Binlog Dump`, `Change user`, `Close stmt`, `Connect Out`, `Connect`, `Create DB`, `Daemon`, `Debug`, `Delayed insert`, `Drop DB`, `Execute`, `Fetch`, `Field List`, `Init DB`, `Kill`, `Long Data`, `NoAudit`, `Ping`, `Prepare`, `Processlist`, `Query`, `Quit`, `Refresh`, `Register Slave`, `Reset stmt`, `Set option`, `Shutdown`, `Sleep`, `Statistics`, `Table Dump`, `Time`.

With the exception of `Audit` and `NoAudit`, these values correspond to the `COM_xxx` command values listed in the `mysql_com.h` header file. For example, `Create DB` and `Shutdown` correspond to `COM_CREATE_DB` and `COM_SHUTDOWN`, respectively.

- `<RECORD_ID>`

  A unique identifier for the audit record. The value is composed from a sequence number and timestamp, in the format *`SEQ_TIMESTAMP`*. The sequence number is initialized to the size of the audit log file at the time the audit log plugin opens it and increments by 1 for each record logged. The timestamp is a UTC value in *`yyyy-mm-dd`*`T`*`hh:mm:ss`* format indicating the time when the audit log plugin opened the file.

  Example:

  ```
  <RECORD_ID>28743_2013-09-18T21:03:24</RECORD_ID>
  ```

- `<TIMESTAMP>`

  The date and time that the audit event was generated. For example, the event corresponding to execution of an SQL statement received from a client has a `<TIMESTAMP>` value occurring after the statement finishes, not when it is received. The value has the format *`yyyy-mm-dd`*`T`*`hh:mm:ss`* `UTC` (with `T`, no decimals). The format includes a time zone specifier at the end. Currently, the time zone is always UTC.

  Example:

  ```
  <TIMESTAMP>2013-09-17T15:03:49 UTC</TIMESTAMP>
  ```

The following elements are optional in `<AUDIT_RECORD>` elements. Many of them occur only with specific `<NAME>` values.

- `<COMMAND_CLASS>`

  A string that indicates the type of action performed.

  Example:

  ```
  <COMMAND_CLASS>drop_table</COMMAND_CLASS>
  ```

  The values come from the `com_status_vars` array in the `sql/mysqld.cc` file in a MySQL source distribution. They correspond to the status variables displayed by this statment:

  ```
  SHOW STATUS LIKE 'Com%';
  ```

- `<CONNECTION_ID>`

An unsigned integer representing the client connection identifier. This is the same as the `CONNECTION_ID()` function value within the session.

Example:

```
<CONNECTION_ID>127</CONNECTION_ID>
```

- `<DB>`

  A string representing the default database name. This element appears only if the `<NAME>` value is `Connect` or `Change user`.

- `<HOST>`

  A string representing the client host name. This element appears only if the `<NAME>` value is `Connect`, `Change user`, or `Query`.

  Example:

  ```
  <HOST>localhost</HOST>
  ```

- `<IP>`

  A string representing the client IP address. This element appears only if the `<NAME>` value is `Connect`, `Change user`, or `Query`.

  Example:

  ```
  <IP>127.0.0.1</IP>
  ```

- `<MYSQL_VERSION>`

  A string representing the MySQL server version. This is the same as the value of the `VERSION()` function or `version` system variable. This element appears only if the `<NAME>` value is `Audit`.

  Example:

  ```
  <MYSQL_VERSION>5.7.1-m11-log</MYSQL_VERSION>
  ```

- `<OS_LOGIN>`

  A string representing the external user (empty if none). The value may differ from the `<USER>` value, for example, if the server authenticates the client using an external authentication method. This element appears only if the `<NAME>` value is `Connect`, `Change user`, or `Query`.

- `<OS_VERSION>`

  A string representing the operating system on which the server was built or is running. This element appears only if the `<NAME>` value is `Audit`.

  Example:

  ```
  <OS_VERSION>x86_64-Linux</OS_VERSION>
  ```

- `<PRIV_USER>`

A string representing the user that the server authenticated the client as. This is the user name that the server uses for privilege checking, and may differ from the <USER> value. This element appears only if the <NAME> value is Connect or Change user.

- <PROXY_USER>

  A string representing the proxy user. The value is empty if user proxying is not in effect. This element appears only if the <NAME> value is Connect or Change user.

- <SERVER_ID>

  An unsigned integer representing the server ID. This is the same as the value of the server_id system variable. This element appears only if the <NAME> value is Audit or NoAudit.

  Example:

  ```
  <SERVER_ID>1</SERVER_ID>
  ```

- <SQLTEXT>

  A string representing the text of an SQL statement. The value can be empty. Long values may be truncated. This element appears only if the <NAME> value is Query or Execute.

  The string, like the audit log file itself, is written using UTF-8 (up to 4 bytes per character), so the value may be the result of conversion. For example, the original statement might have been received from the client as an SJIS string.

  Example:

  ```
  <SQLTEXT>DELETE FROM t1</SQLTEXT>
  ```

- <STARTUP_OPTIONS>

  A string representing the options that were given on the command line or in option files when the MySQL server was started. This element appears only if the <NAME> value is Audit.

  Example:

  ```
  <STARTUP_OPTIONS>/usr/local/mysql/bin/mysqld
    --port=3306 --log-output=FILE</STARTUP_OPTIONS>
  ```

- <STATUS>

  An unsigned integer representing the command status: 0 for success, nonzero if an error occurred. This is the same as the value of the mysql_errno() C API function.

  The audit log does not contain the SQLSTATE value or error message. To see the associations between error codes, SQLSTATE values, and messages, see Section C.3, “Server Error Codes and Messages”.

  Warnings are not logged.

  See the description for <STATUS_CODE> for information about how it differs from <STATUS>.

  Example:

```
<STATUS>1051</STATUS>
```

- `<STATUS_CODE>`

  An unsigned integer representing the command status: 0 for success, 1 if an error occurred.

  The `STATUS_CODE` value differs from the `STATUS` value: `STATUS_CODE` is 0 for success and 1 for error, which is compatible with the EZ_collector consumer for Audit Vault. `STATUS` is the value of the `mysql_errno()` C API function. This is 0 for success and nonzero for error, and thus is not necessarily 1 for error.

  Example:

  ```
  <STATUS_CODE>0</STATUS_CODE>
  ```

- `<USER>`

  A string representing the user name sent by the client. This may differ from the `<PRIV_USER>` value. This element appears only if the `<NAME>` value is `Connect`, `Change user`, or `Query`.

  Example:

  ```
  <USER>root[root] @ localhost [127.0.0.1]</USER>
  ```

- `<VERSION>`

  An unsigned integer representing the version of the audit log file format. This element appears only if the `<NAME>` value is `Audit`.

  Example:

  ```
  <VERSION>1</VERSION>
  ```

Old Audit Log File Format

Every `<AUDIT_RECORD>` element contains a set of mandatory attributes. Other optional attributes may appear depending on the audit record type.

The following attributes are mandatory in every `<AUDIT_RECORD>` element:

- `NAME`

  A string representing the type of instruction that generated the audit event, such as a command that the server received from a client.

  Example: `NAME="Query"`

  Some common `NAME` values:

  ```
  "Audit"    When auditing starts, which may be server startup time
  "Connect"  When a client connects, also known as logging in
  "Query"    An SQL statement (executed directly)
  "Prepare"  Preparation of an SQL statement; usually followed by Execute
  "Execute"  Execution of an SQL statement; usually follows Prepare
  "Shutdown" Server shutdown
  "Quit"     When a client disconnects
  ```

```
"NoAudit"  Auditing has been turned off
```

The possible values are `"Audit"`, `"Binlog Dump"`, `"Change user"`, `"Close stmt"`, `"Connect Out"`, `"Connect"`, `"Create DB"`, `"Daemon"`, `"Debug"`, `"Delayed insert"`, `"Drop DB"`, `"Execute"`, `"Fetch"`, `"Field List"`, `"Init DB"`, `"Kill"`, `"Long Data"`, `"NoAudit"`, `"Ping"`, `"Prepare"`, `"Processlist"`, `"Query"`, `"Quit"`, `"Refresh"`, `"Register Slave"`, `"Reset stmt"`, `"Set option"`, `"Shutdown"`, `"Sleep"`, `"Statistics"`, `"Table Dump"`, `"Time"`.

With the exception of `"Audit"` and `"NoAudit"`, these values correspond to the `COM_`*`xxx`* command values listed in the `mysql_com.h` header file. For example, `"Create DB"` and `"Shutdown"` correspond to `COM_CREATE_DB` and `COM_SHUTDOWN`, respectively.

- `TIMESTAMP`

  The date and time that the audit event was generated. For example, the event corresponding to execution of an SQL statement received from a client has a `TIMESTAMP` value occurring after the statement finishes, not when it is received. The value is UTC, in the format *`yyyy-mm-dd`*`T`*`hh:mm:ss`* (with `T`, no decimals).

  Example: `TIMESTAMP="2012-08-09T12:55:16"`

The following attributes are optional in `<AUDIT_RECORD>` elements. Many of them occur only for elements with specific values of the `NAME` attribute.

- `CONNECTION_ID`

  An unsigned integer representing the client connection identifier. This is the same as the `CONNECTION_ID()` function value within the session.

  Example: `CONNECTION_ID="127"`

- `DB`

  A string representing the default database name. This attribute appears only if the `NAME` value is `"Connect"` or `"Change user"`.

- `HOST`

  A string representing the client host name. This attribute appears only if the `NAME` value is `"Connect"` or `"Change user"`.

  Example: `HOST="localhost"`

- `IP`

  A string representing the client IP address. This attribute appears only if the `NAME` value is `"Connect"` or `"Change user"`.

  Example: `IP="127.0.0.1"`

- `MYSQL_VERSION`

  A string representing the MySQL server version. This is the same as the value of the `VERSION()` function or `version` system variable. This attribute appears only if the `NAME` value is `"Audit"`.

  Example: `MYSQL_VERSION="5.6.11-log"`

- `OS_LOGIN`

A string representing the external user (empty if none). The value may differ from USER, for example, if the server authenticates the client using an external authentication method. This attribute appears only if the NAME value is "Connect" or "Change user".

- OS_VERSION

  A string representing the operating system on which the server was built or is running. This attribute appears only if the NAME value is "Audit".

  Example: OS_VERSION="x86_64-Linux"

- PRIV_USER

  A string representing the user that the server authenticated the client as. This is the user name that the server uses for privilege checking, and may be different from the USER value. This attribute appears only if the NAME value is "Connect" or "Change user".

- PROXY_USER

  A string representing the proxy user. The value is empty if user proxying is not in effect. This attribute appears only if the NAME value is "Connect" or "Change user".

- SERVER_ID

  An unsigned integer representing the server ID. This is the same as the value of the server_id system variable. This attribute appears only if the NAME value is "Audit" or "NoAudit".

  Example: SERVER_ID="1"

- SQLTEXT

  A string representing the text of an SQL statement. The value can be empty. Long values may be truncated. This attribute appears only if the NAME value is "Query" or "Execute".

  The string, like the audit log file itself, is written using UTF-8 (up to 4 bytes per character), so the value may be the result of conversion. For example, the original statement might have been received from the client as an SJIS string.

  Example: SQLTEXT="DELETE FROM t1"

- STARTUP_OPTIONS

  A string representing the options that were given on the command line or in option files when the MySQL server was started. This attribute appears only if the NAME value is "Audit".

  Example: STARTUP_OPTIONS="--port=3306 --log-output=FILE"

- STATUS

  An unsigned integer representing the command status: 0 for success, nonzero if an error occurred. This is the same as the value of the mysql_errno() C API function.

  The audit log does not contain the SQLSTATE value or error message. To see the associations between error codes, SQLSTATE values, and messages, see Section C.3, “Server Error Codes and Messages”.

  Warnings are not logged.

  Example: STATUS="1051"

- USER

  A string representing the user name sent by the client. This may be different from the PRIV_USER value. This attribute appears only if the NAME value is "Connect" or "Change user".

- VERSION

  An unsigned integer representing the version of the audit log file format. This attribute appears only if the NAME value is "Audit".

  Example: VERSION="1"

6.3.13.4 Audit Log Plugin Logging Control

This section describes how the audit_log plugin performs logging and the system variables that control how logging occurs. It assumes familiarity with the log file format described in Section 6.3.13.3, "The Audit Log File".

When the audit log plugin opens its log file, it checks whether the XML declaration and opening <AUDIT> root element tag need to be written and writes them if so. When the audit log plugin terminates, it writes a closing </AUDIT> tag to the file.

If the log file exists at open time, the plugin checks whether the file ends with an </AUDIT> tag and truncates it if so before writing any <AUDIT_RECORD> elements. If the log file exists but does not end with </AUDIT> or the </AUDIT> tag cannot be truncated, the plugin considers the file malformed and fails to initialize. This can occur if the server crashes or is killed with the audit log plugin running. No logging occurs until the problem is rectified. Check the error log for diagnostic information:

```
[ERROR] Plugin 'audit_log' init function returned error.
```

To deal with this problem, you must either remove or rename the malformed log file and restart the server.

The MySQL server calls the audit log plugin to write an <AUDIT_RECORD> element whenever an auditable event occurs, such as when it completes execution of an SQL statement received from a client. Typically the first <AUDIT_RECORD> element written after server startup has the server description and startup options. Elements following that one represent events such as client connect and disconnect events, executed SQL statements, and so forth. Only top-level statements are logged, not statements within stored programs such as triggers or stored procedures. Contents of files referenced by statements such as LOAD DATA INFILE are not logged.

To permit control over how logging occurs, the audit_log plugin provides several system variables, described following. For more information, see Section 6.3.13.5, "Audit Log Plugin Options and Variables".

- audit_log_file: The name of the log file. By default, the name is audit.log in the server data directory. For security reasons, the audit log file should be written to a directory accessible only to the MySQL server and users with a legitimate reason to view the log.

- audit_log_policy: What kinds of information the plugin writes. By default, this variable is set to ALL (write all auditable events), but also permits values of LOGINS or QUERIES to log only login or query events, or NONE to disable logging.

- audit_log_strategy: The method used for log writes. By default, the strategy value is ASYNCHRONOUS and the plugin logs asynchronously to a buffer, waiting if the buffer is full. It's possible to tell the plugin not to wait (PERFORMANCE) or to log synchronously, either using file system caching (SEMISYNCHRONOUS) or forcing output with a sync() call after each write request (SYNCHRONOUS).

  Asynchronous logging strategy has these characteristics:

  - Minimal impact on server performance and scalability.

  - Blocking of threads that generate audit events for the shortest possible time; that is, time to allocate the buffer plus time to copy the event to the buffer.

  - Output goes to the buffer. Writes from the buffer to the log file are handled by a separate thread.

  A disadvantage of PERFORMANCE strategy is that it drops events when the buffer is full. For a heavily loaded server, it is more likely that the audit log will be missing events.

  With asynchronous logging, the integrity of the log file may be compromised if a problem occurs during a write to the file or if the plugin does not shut down cleanly (for example, in the event that the server host crashes). To reduce this risk, set audit_log_strategy use synchronous logging. Regardless of strategy, logging occurs on a best-effort basis, with no guarantee of consistency.

- audit_log_buffer_size: The size of the buffer for asynchronous logging. The plugin uses a single buffer, which it allocates when it initializes and removes when it terminates. The plugin allocates this buffer only if logging is asynchronous.

- audit_log_rotate_on_size, audit_log_flush: These variables permit audit log file rotation and flushing. The audit log file has the potential to grow very large and consume a lot of disk space. To manage the space used, either enable automatic log rotation, or manually rename the audit file and flush the log to open a new file. The renamed file can be removed or backed up as desired.

  By default, audit_log_rotate_on_size=0 and there is no log rotation. In this case, the audit log plugin closes and reopens the log file when the audit_log_flush value changes from disabled to enabled. Log file renaming must be done externally to the server. Suppose that you want to maintain the three most recent log files, which cycle through the names audit.log.1.xml through audit.log.3.xml. On Unix, perform rotation manually like this:

  1. From the command line, rename the current log files:

     ```
     shell> mv audit.log.2.xml audit.log.3.xml
     shell> mv audit.log.1.xml audit.log.2.xml
     shell> mv audit.log audit.log.1.xml
     ```

     At this point, the plugin is still writing to the current log file, which has been renamed to audit.log.1.xml.

  2. Connect to the server and flush the log file so the plugin closes it and reopens a new audit.log file:

     ```
     mysql> SET GLOBAL audit_log_flush = ON;
     ```

  If audit_log_rotate_on_size is greater than 0, setting audit_log_flush has no effect. In this case, the audit log plugin closes and reopens its log file whenever a write to the file causes its size to exceed the audit_log_rotate_on_size value. The plugin renames the original file to have an extension consisting of a timestamp and .xml suffix. For example, audit.log might be renamed to audit.log.13792588477726520.xml. The last 7 digits of the timestamp are a fractional second part. The first 10 digits are a Unix timestamp value that can be interpreted using the FROM_UNIXTIME() function:

  ```
  mysql> SELECT FROM_UNIXTIME(1379258847);
  +---------------------------+
  ```

```
| FROM_UNIXTIME(1379258847) |
+---------------------------+
| 2013-09-15 10:27:27       |
+---------------------------+
```

6.3.13.5 Audit Log Plugin Options and Variables

This section describes the command options and system variables that control operation of the audit log plugin. If values specified at startup time are incorrect, the plugin may fail to initialize properly and the server does not load it. In this case, the server may also produce error messages for other audit log settings because it will not recognize them.

To control the activation of the `audit_log` plugin, use this option:

- `--audit-log[=value]`

Command-Line Format	`--audit-log[=value]`	
Option-File Format	`audit-log`	
	Permitted Values	
	Type	`enumeration`
	Default	`ON`
	Valid Values	`ON`
		`OFF`
		`FORCE`
		`FORCE_PLUS_PERMANENT`

This option controls how the server loads the `audit_log` plugin at startup. It is available only if the audit log plugin has been previously registered with `INSTALL PLUGIN` or is loaded with `--plugin-load`. See Section 6.3.13.1, “Installing the Audit Log Plugin”.

The option value should be one of those available for plugin-loading options, as described in Section 5.1.8.1, “Installing and Uninstalling Plugins”. For example, `--audit-log=FORCE_PLUS_PERMANENT` tells the server to load the plugin and prevent it from being removed while the server is running.

If the `audit_log` plugin is installed, it exposes several system variables that permit control over logging:

```
mysql> SHOW VARIABLES LIKE 'audit_log%';
+--------------------------+--------------+
| Variable_name            | Value        |
+--------------------------+--------------+
| audit_log_buffer_size    | 1048576      |
| audit_log_file           | audit.log    |
| audit_log_flush          | OFF          |
| audit_log_policy         | ALL          |
| audit_log_rotate_on_size | 0            |
| audit_log_strategy       | ASYNCHRONOUS |
+--------------------------+--------------+
```

You can set any of these variables at server startup, and some of them at runtime.

- `audit_log_buffer_size`

System Variable Name	`audit_log_buffer_size`
Variable Scope	Global

Dynamic Variable	No	
	Permitted Values	
	Platform Bit Size	`32`
	Type	`numeric`
	Default	`1048576`
	Range	`4096 .. 4294967295`
	Permitted Values	
	Platform Bit Size	`64`
	Type	`numeric`
	Default	`1048576`
	Range	`4096 .. 18446744073709547520`

When the audit log plugin writes events to the log asynchronously, it uses a buffer to store event contents prior to writing them. This variable controls the size of that buffer, in bytes. The server adjusts the value to a multiple of 4096. The plugin uses a single buffer, which it allocates when it initializes and removes when it terminates. The plugin allocates this buffer only if logging is asynchronous.

This variable is available only if the `audit_log` plugin is enabled.

- `audit_log_file`

System Variable Name	`audit_log_file`	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	`file name`
	Default	`audit.log`

The name of the file to which the audit log plugin writes events. The default value is `audit.log`. If the file name is a relative path, the server interprets it relative to the data directory. For security reasons, the audit log file should be written to a directory accessible only to the MySQL server and users with a legitimate reason to view the log.

This variable is available only if the `audit_log` plugin is enabled.

- `audit_log_flush`

System Variable Name	`audit_log_flush`	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	`boolean`
	Default	`OFF`

When this variable is set to enabled (1 or ON), the audit log plugin closes and reopens its log file to flush it. (The value remains OFF so that you need not disable it explicitly before enabling it again to perform another flush.) Enabling this variable has no effect unless audit_log_rotate_on_size is 0.

This variable is available only if the audit_log plugin is enabled.

- audit_log_format

Introduced	5.7.3	
System Variable Name	audit_log_format	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values (>= 5.7.3)	
	Type	enumeration
	Default	NEW
	Valid Values	OLD
		NEW

The audit log file format. Permitted values are OLD and NEW (default NEW). For details about each format, see Section 6.3.13.3, “The Audit Log File”.

If you change the value of audit_log_format, use this procedure to avoid writing log entries in one format to an existing log file that contains entries in a different format:

1. Stop the server.
2. Rename the current audit log file manually.
3. Restart the server with the new value of audit_log_format. The audit log plugin will create a new log file, which will contain log entries in the selected format.

This variable was added in MySQL 5.7.3. It is available only if the audit_log plugin is enabled.

- audit_log_policy

System Variable Name	audit_log_policy	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	enumeration
	Default	ALL
	Valid Values	ALL
		NONE
		LOGINS
		QUERIES

The policy controlling the information written by the audit log plugin to its log file. The following table shows the permitted values.

Value	Description
ALL	Log all events
NONE	Log nothing (disable the audit stream)
LOGINS	Log only login events
QUERIES	Log only query events

This variable is available only if the audit_log plugin is enabled.

- audit_log_rotate_on_size

System Variable Name	audit_log_rotate_on_size	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	numeric
	Default	0

If the audit_log_rotate_on_size value is greater than 0, the audit log plugin closes and reopens its log file if a write to the file causes its size to exceed this value. The original file is renamed to have a timestamp extension.

If the audit_log_rotate_on_size value is 0, the plugin does not close and reopen its log based on size. Instead, use audit_log_flush to close and reopen the log on demand. In this case, rename the file externally to the server before flushing it.

For more information about audit log file rotation and timestamp interpretation, see Section 6.3.13.4, “Audit Log Plugin Logging Control”.

If you set this variable to a value that is not a multiple of 4096, it is truncated to the nearest multiple. (Thus, setting it to a value less than 4096 has the effect of setting it to 0 and no rotation occurs.)

This variable is available only if the audit_log plugin is enabled.

- audit_log_strategy

System Variable Name	audit_log_strategy	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	enumeration
	Default	ASYNCHRONOUS
	Valid Values	ASYNCHRONOUS
		PERFORMANCE
		SEMISYNCHRONOUS
		SYNCHRONOUS

The logging method used by the audit log plugin. The following table describes the permitted values.

Table 6.15 Audit Log Strategies

Value	Meaning
ASYNCHRONOUS	Log asynchronously, wait for space in output buffer
PERFORMANCE	Log asynchronously, drop request if insufficient space in output buffer
SEMISYNCHRONOUS	Log synchronously, permit caching by operating system
SYNCHRONOUS	Log synchronously, call sync() after each request

This variable is available only if the audit_log plugin is enabled.

6.3.13.6 Audit Log Plugin Restrictions

The audit log plugin is subject to these restrictions:

- Only top-level statements are logged, not statements within stored programs such as triggers or stored procedures.
- Contents of files referenced by statements such as LOAD DATA INFILE are not logged.

6.3.14 SQL-Based MySQL Account Activity Auditing

Applications can use the following guidelines to perform SQL-based auditing that ties database activity to MySQL accounts.

MySQL accounts correspond to rows in the mysql.user table. When a client connects successfully, the server authenticates the client to a particular row in this table. The User and Host column values in this row uniquely identify the account and correspond to the '*user_name*'@'*host_name*' format in which account names are written in SQL statements.

The account used to authenticate a client determines which privileges the client has. Normally, the CURRENT_USER() function can be invoked to determine which account this is for the client user. Its value is constructed from the User and Host columns of the user table row for the account.

However, there are circumstances under which the CURRENT_USER() value corresponds not to the client user but to a different account. This occurs in contexts when privilege checking is not based the client's account:

- Stored routines (procedures and functions) defined with the SQL SECURITY DEFINER characteristic
- Views defined with the SQL SECURITY DEFINER characteristic
- Triggers and events

In those contexts, privilege checking is done against the DEFINER account and CURRENT_USER() refers to that account, not to the account for the client who invoked the stored routine or view or who caused the trigger to activate. To determine the invoking user, you can call the USER() function, which returns a value indicating the actual user name provided by the client and the host from which the client connected. However, this value does not necessarily correspond directly to an account in the user table, because the USER() value never contains wildcards, whereas account values (as returned by CURRENT_USER()) may contain user name and host name wildcards.

For example, a blank user name matches any user, so an account of ''@'localhost' enables clients to connect as an anonymous user from the local host with any user name. If this case, if a client connects as user1 from the local host, USER() and CURRENT_USER() return different values:

```
mysql> SELECT USER(), CURRENT_USER();
+-----------------+----------------+
| USER()          | CURRENT_USER() |
+-----------------+----------------+
| user1@localhost | @localhost     |
+-----------------+----------------+
```

The host name part of an account can contain wildcards, too. If the host name contains a `'%'` or `'_'` pattern character or uses netmask notation, the account can be used for clients connecting from multiple hosts and the `CURRENT_USER()` value will not indicate which one. For example, the account `'user2'@'%.example.com'` can be used by `user2` to connect from any host in the `example.com` domain. If `user2` connects from `remote.example.com`, `USER()` and `CURRENT_USER()` return different values:

```
mysql> SELECT USER(), CURRENT_USER();
+--------------------------+---------------------+
| USER()                   | CURRENT_USER()      |
+--------------------------+---------------------+
| user2@remote.example.com | user2@%.example.com |
+--------------------------+---------------------+
```

If an application must invoke `USER()` for user auditing (for example, if it does auditing from within triggers) but must also be able to associate the `USER()` value with an account in the `user` table, it is necessary to avoid accounts that contain wildcards in the `User` or `Host` column. Specifically, do not permit `User` to be empty (which creates an anonymous-user account), and do not permit pattern characters or netmask notation in `Host` values. All accounts must have a nonempty `User` value and literal `Host` value.

With respect to the previous examples, the `''@'localhost'` and `'user2'@'%.example.com'` accounts should be changed not to use wildcards:

```
RENAME USER ''@'localhost' TO 'user1'@'localhost';
RENAME USER 'user2'@'%.example.com' TO 'user2'@'remote.example.com';
```

If `user2` must be able to connect from several hosts in the `example.com` domain, there should be a separate account for each host.

To extract the user name or host name part from a `CURRENT_USER()` or `USER()` value, use the `SUBSTRING_INDEX()` function:

```
mysql> SELECT SUBSTRING_INDEX(CURRENT_USER(),'@',1);
+---------------------------------------+
| SUBSTRING_INDEX(CURRENT_USER(),'@',1) |
+---------------------------------------+
| user1                                 |
+---------------------------------------+

mysql> SELECT SUBSTRING_INDEX(CURRENT_USER(),'@',-1);
+----------------------------------------+
| SUBSTRING_INDEX(CURRENT_USER(),'@',-1) |
+----------------------------------------+
| localhost                              |
+----------------------------------------+
```

Chapter 7 Backup and Recovery

Table of Contents

It is important to back up your databases so that you can recover your data and be up and running again in case problems occur, such as system crashes, hardware failures, or users deleting data by mistake. Backups are also essential as a safeguard before upgrading a MySQL installation, and they can be used to transfer a MySQL installation to another system or to set up replication slave servers.

MySQL offers a variety of backup strategies from which you can choose the methods that best suit the requirements for your installation. This chapter discusses several backup and recovery topics with which you should be familiar:

- Types of backups: Logical versus physical, full versus incremental, and so forth.
- Methods for creating backups.
- Recovery methods, including point-in-time recovery.
- Backup scheduling, compression, and encryption.
- Table maintenance, to enable recovery of corrupt tables.

Additional Resources

Resources related to backup or to maintaining data availability include the following:

- Customers of MySQL Enterprise Edition can use the MySQL Enterprise Backup product for backups. For an overview of the MySQL Enterprise Backup product, see Section 23.2, “MySQL Enterprise Backup”.
- A forum dedicated to backup issues is available at http://forums.mysql.com/list.php?28.

- Details for `mysqldump`, `mysqlhotcopy`, and other MySQL backup programs can be found in Chapter 4, *MySQL Programs*.
- The syntax of the SQL statements described here is given in Chapter 13, *SQL Statement Syntax*.
- For additional information about `InnoDB` backup procedures, see Section 14.2.14, "`InnoDB` Backup and Recovery".
- Replication enables you to maintain identical data on multiple servers. This has several benefits, such as enabling client query load to be distributed over servers, availability of data even if a given server is taken offline or fails, and the ability to make backups with no impact on the master by using a slave server. See Chapter 16, *Replication*.
- MySQL Cluster provides a high-availability, high-redundancy version of MySQL adapted for the distributed computing environment. See MySQL Cluster NDB 7.3, which provides information about MySQL Cluster NDB 7.3 (based on MySQL 5.6 but containing the latest improvements and fixes for the `NDBCLUSTER` storage engine).

  > **Note**
  >
  > The `NDBCLUSTER` storage engine is currently not supported in MySQL 5.7.

- Distributed Replicated Block Device (DRBD) is another high-availability solution. It works by replicating a block device from a primary server to a secondary server at the block level. See Chapter 15, *High Availability and Scalability*

7.1 Backup and Recovery Types

This section describes the characteristics of different types of backups.

Physical (Raw) Versus Logical Backups

Physical backups consist of raw copies of the directories and files that store database contents. This type of backup is suitable for large, important databases that need to be recovered quickly when problems occur.

Logical backups save information represented as logical database structure (`CREATE DATABASE`, `CREATE TABLE` statements) and content (`INSERT` statements or delimited-text files). This type of backup is suitable for smaller amounts of data where you might edit the data values or table structure, or recreate the data on a different machine architecture.

Physical backup methods have these characteristics:

- The backup consists of exact copies of database directories and files. Typically this is a copy of all or part of the MySQL data directory.
- Physical backup methods are faster than logical because they involve only file copying without conversion.
- Output is more compact than for logical backup.
- Because backup speed and compactness are important for busy, important databases, the MySQL Enterprise Backup product performs physical backups. For an overview of the MySQL Enterprise Backup product, see Section 23.2, "MySQL Enterprise Backup".
- Backup and restore granularity ranges from the level of the entire data directory down to the level of individual files. This may or may not provide for table-level granularity, depending on storage engine. For

example, InnoDB tables can each be in a separate file, or share file storage with other InnoDB tables; each MyISAM table corresponds uniquely to a set of files.

- In addition to databases, the backup can include any related files such as log or configuration files.
- Data from MEMORY tables is tricky to back up this way because their contents are not stored on disk. (The MySQL Enterprise Backup product has a feature where you can retrieve data from MEMORY tables during a backup.)
- Backups are portable only to other machines that have identical or similar hardware characteristics.
- Backups can be performed while the MySQL server is not running. If the server is running, it is necessary to perform appropriate locking so that the server does not change database contents during the backup. MySQL Enterprise Backup does this locking automatically for tables that require it.
- Physical backup tools include the mysqlbackup of MySQL Enterprise Backup for InnoDB or any other tables, file system-level commands (such as cp, scp, tar, rsync), or mysqlhotcopy for MyISAM tables.
- For restore:
  - MySQL Enterprise Backup restores InnoDB and other tables that it backed up.
  - ndb_restore restores NDB tables.
  - Files copied at the file system level or with mysqlhotcopy can be copied back to their original locations with file system commands.

Logical backup methods have these characteristics:

- The backup is done by querying the MySQL server to obtain database structure and content information.
- Backup is slower than physical methods because the server must access database information and convert it to logical format. If the output is written on the client side, the server must also send it to the backup program.
- Output is larger than for physical backup, particularly when saved in text format.
- Backup and restore granularity is available at the server level (all databases), database level (all tables in a particular database), or table level. This is true regardless of storage engine.
- The backup does not include log or configuration files, or other database-related files that are not part of databases.
- Backups stored in logical format are machine independent and highly portable.
- Logical backups are performed with the MySQL server running. The server is not taken offline.
- Logical backup tools include the mysqldump program and the SELECT ... INTO OUTFILE statement. These work for any storage engine, even MEMORY.
- To restore logical backups, SQL-format dump files can be processed using the mysql client. To load delimited-text files, use the LOAD DATA INFILE statement or the mysqlimport client.

Online Versus Offline Backups

Online backups take place while the MySQL server is running so that the database information can be obtained from the server. Offline backups take place while the server is stopped. This distinction can also be described as “hot” versus “cold” backups; a “warm” backup is one where the server remains running but locked against modifying data while you access database files externally.

Online backup methods have these characteristics:

- The backup is less intrusive to other clients, which can connect to the MySQL server during the backup and may be able to access data depending on what operations they need to perform.
- Care must be taken to impose appropriate locking so that data modifications do not take place that would compromise backup integrity. The MySQL Enterprise Backup product does such locking automatically.

Offline backup methods have these characteristics:

- Clients can be affected adversely because the server is unavailable during backup. For that reason, such backups are often taken from a replication slave server that can be taken offline without harming availability.
- The backup procedure is simpler because there is no possibility of interference from client activity.

A similar distinction between online and offline applies for recovery operations, and similar characteristics apply. However, it is more likely that clients will be affected for online recovery than for online backup because recovery requires stronger locking. During backup, clients might be able to read data while it is being backed up. Recovery modifies data and does not just read it, so clients must be prevented from accessing data while it is being restored.

Local Versus Remote Backups

A local backup is performed on the same host where the MySQL server runs, whereas a remote backup is done from a different host. For some types of backups, the backup can be initiated from a remote host even if the output is written locally on the server. host.

- `mysqldump` can connect to local or remote servers. For SQL output (`CREATE` and `INSERT` statements), local or remote dumps can be done and generate output on the client. For delimited-text output (with the `--tab` option), data files are created on the server host.
- `mysqlhotcopy` performs only local backups: It connects to the server to lock it against data modifications and then copies local table files.
- `SELECT ... INTO OUTFILE` can be initiated from a local or remote client host, but the output file is created on the server host.
- Physical backup methods typically are initiated locally on the MySQL server host so that the server can be taken offline, although the destination for copied files might be remote.

Snapshot Backups

Some file system implementations enable “snapshots” to be taken. These provide logical copies of the file system at a given point in time, without requiring a physical copy of the entire file system. (For example, the implementation may use copy-on-write techniques so that only parts of the file system modified after the snapshot time need be copied.) MySQL itself does not provide the capability for taking file system snapshots. It is available through third-party solutions such as Veritas, LVM, or ZFS.

Full Versus Incremental Backups

A full backup includes all data managed by a MySQL server at a given point in time. An incremental backup consists of the changes made to the data during a given time span (from one point in time to another). MySQL has different ways to perform full backups, such as those described earlier in this section. Incremental backups are made possible by enabling the server's binary log, which the server uses to record data changes.

Full Versus Point-in-Time (Incremental) Recovery

A full recovery restores all data from a full backup. This restores the server instance to the state that it had when the backup was made. If that state is not sufficiently current, a full recovery can be followed by recovery of incremental backups made since the full backup, to bring the server to a more up-to-date state.

Incremental recovery is recovery of changes made during a given time span. This is also called point-in-time recovery because it makes a server's state current up to a given time. Point-in-time recovery is based on the binary log and typically follows a full recovery from the backup files that restores the server to its state when the backup was made. Then the data changes written in the binary log files are applied as incremental recovery to redo data modifications and bring the server up to the desired point in time.

Table Maintenance

Data integrity can be compromised if tables become corrupt. For `InnoDB` tables, this is not a typical issue. For programs to check `MyISAM` tables and repair them if problems are found, see Section 7.6, “`MyISAM` Table Maintenance and Crash Recovery”.

Backup Scheduling, Compression, and Encryption

Backup scheduling is valuable for automating backup procedures. Compression of backup output reduces space requirements, and encryption of the output provides better security against unauthorized access of backed-up data. MySQL itself does not provide these capabilities. The MySQL Enterprise Backup product can compress `InnoDB` backups, and compression or encryption of backup output can be achieved using file system utilities. Other third-party solutions may be available.

7.2 Database Backup Methods

This section summarizes some general methods for making backups.

Making a Hot Backup with MySQL Enterprise Backup

Customers of MySQL Enterprise Edition can use the MySQL Enterprise Backup product to do physical backups of entire instances or selected databases, tables, or both. This product includes features for incremental and compressed backups. Backing up the physical database files makes restore much faster than logical techniques such as the `mysqldump` command. `InnoDB` tables are copied using a hot backup mechanism. (Ideally, the `InnoDB` tables should represent a substantial majority of the data.) Tables from other storage engines are copied using a warm backup mechanism. For an overview of the MySQL Enterprise Backup product, see Section 23.2, “MySQL Enterprise Backup”.

Making Backups with `mysqldump` or `mysqlhotcopy`

The `mysqldump` program and the `mysqlhotcopy` script can make backups. `mysqldump` is more general because it can back up all kinds of tables. `mysqlhotcopy` works only with some storage engines. (See Section 7.4, “Using `mysqldump` for Backups”, and Section 4.6.9, “`mysqlhotcopy` — A Database Backup Program”.)

For `InnoDB` tables, it is possible to perform an online backup that takes no locks on tables using the `--single-transaction` option to `mysqldump`. See Section 7.3.1, “Establishing a Backup Policy”.

Making Backups by Copying Table Files

For storage engines that represent each table using its own files, tables can be backed up by copying those files. For example, MyISAM tables are stored as files, so it is easy to do a backup by copying files (*.frm, *.MYD, and *.MYI files). To get a consistent backup, stop the server or lock and flush the relevant tables:

```
FLUSH TABLES tbl_list WITH READ LOCK;
```

You need only a read lock; this enables other clients to continue to query the tables while you are making a copy of the files in the database directory. The flush is needed to ensure that the all active index pages are written to disk before you start the backup. See Section 13.3.5, "LOCK TABLES and UNLOCK TABLES Syntax", and Section 13.7.6.3, "FLUSH Syntax".

You can also create a binary backup simply by copying all table files, as long as the server isn't updating anything. The mysqlhotcopy script uses this method. (But note that table file copying methods do not work if your database contains InnoDB tables. mysqlhotcopy does not work for InnoDB tables because InnoDB does not necessarily store table contents in database directories. Also, even if the server is not actively updating data, InnoDB may still have modified data cached in memory and not flushed to disk.)

Making Delimited-Text File Backups

To create a text file containing a table's data, you can use SELECT * INTO OUTFILE '*file_name*' FROM *tbl_name*. The file is created on the MySQL server host, not the client host. For this statement, the output file cannot already exist because permitting files to be overwritten constitutes a security risk. See Section 13.2.9, "SELECT Syntax". This method works for any kind of data file, but saves only table data, not the table structure.

Another way to create text data files (along with files containing CREATE TABLE statements for the backed up tables) is to use mysqldump with the --tab option. See Section 7.4.3, "Dumping Data in Delimited-Text Format with mysqldump".

To reload a delimited-text data file, use LOAD DATA INFILE or mysqlimport.

Making Incremental Backups by Enabling the Binary Log

MySQL supports incremental backups: You must start the server with the --log-bin option to enable binary logging; see Section 5.2.4, "The Binary Log". The binary log files provide you with the information you need to replicate changes to the database that are made subsequent to the point at which you performed a backup. At the moment you want to make an incremental backup (containing all changes that happened since the last full or incremental backup), you should rotate the binary log by using FLUSH LOGS. This done, you need to copy to the backup location all binary logs which range from the one of the moment of the last full or incremental backup to the last but one. These binary logs are the incremental backup; at restore time, you apply them as explained in Section 7.5, "Point-in-Time (Incremental) Recovery Using the Binary Log". The next time you do a full backup, you should also rotate the binary log using FLUSH LOGS, mysqldump --flush-logs, or mysqlhotcopy --flushlog. See Section 4.5.4, "mysqldump — A Database Backup Program", and Section 4.6.9, "mysqlhotcopy — A Database Backup Program".

Making Backups Using Replication Slaves

If you have performance problems with your master server while making backups, one strategy that can help is to set up replication and perform backups on the slave rather than on the master. See Section 16.3.1, "Using Replication for Backups".

If you are backing up a slave replication server, you should back up its master info and relay log info repositories (see Section 16.2.2, “Replication Relay and Status Logs”) when you back up the slave's databases, regardless of the backup method you choose. These information files are always needed to resume replication after you restore the slave's data. If your slave is replicating LOAD DATA INFILE statements, you should also back up any SQL_LOAD-* files that exist in the directory that the slave uses for this purpose. The slave needs these files to resume replication of any interrupted LOAD DATA INFILE operations. The location of this directory is the value of the --slave-load-tmpdir option. If the server was not started with that option, the directory location is the value of the tmpdir system variable.

Recovering Corrupt Tables

If you have to restore MyISAM tables that have become corrupt, try to recover them using REPAIR TABLE or myisamchk -r first. That should work in 99.9% of all cases. If myisamchk fails, see Section 7.6, “MyISAM Table Maintenance and Crash Recovery”.

Making Backups Using a File System Snapshot

If you are using a Veritas file system, you can make a backup like this:

1. From a client program, execute FLUSH TABLES WITH READ LOCK.
2. From another shell, execute mount vxfs snapshot.
3. From the first client, execute UNLOCK TABLES.
4. Copy files from the snapshot.
5. Unmount the snapshot.

Similar snapshot capabilities may be available in other file systems, such as LVM or ZFS.

7.3 Example Backup and Recovery Strategy

This section discusses a procedure for performing backups that enables you to recover data after several types of crashes:

- Operating system crash
- Power failure
- File system crash
- Hardware problem (hard drive, motherboard, and so forth)

The example commands do not include options such as --user and --password for the mysqldump and mysql client programs. You should include such options as necessary to enable client programs to connect to the MySQL server.

Assume that data is stored in the InnoDB storage engine, which has support for transactions and automatic crash recovery. Assume also that the MySQL server is under load at the time of the crash. If it were not, no recovery would ever be needed.

For cases of operating system crashes or power failures, we can assume that MySQL's disk data is available after a restart. The InnoDB data files might not contain consistent data due to the crash, but InnoDB reads its logs and finds in them the list of pending committed and noncommitted transactions that have not been flushed to the data files. InnoDB automatically rolls back those transactions that were not committed, and flushes to its data files those that were committed. Information about this recovery process is conveyed to the user through the MySQL error log. The following is an example log excerpt:

```
InnoDB: Database was not shut down normally.
InnoDB: Starting recovery from log files...
InnoDB: Starting log scan based on checkpoint at
InnoDB: log sequence number 0 13674004
InnoDB: Doing recovery: scanned up to log sequence number 0 13739520
InnoDB: Doing recovery: scanned up to log sequence number 0 13805056
InnoDB: Doing recovery: scanned up to log sequence number 0 13870592
InnoDB: Doing recovery: scanned up to log sequence number 0 13936128
...
InnoDB: Doing recovery: scanned up to log sequence number 0 20555264
InnoDB: Doing recovery: scanned up to log sequence number 0 20620800
InnoDB: Doing recovery: scanned up to log sequence number 0 20664692
InnoDB: 1 uncommitted transaction(s) which must be rolled back
InnoDB: Starting rollback of uncommitted transactions
InnoDB: Rolling back trx no 16745
InnoDB: Rolling back of trx no 16745 completed
InnoDB: Rollback of uncommitted transactions completed
InnoDB: Starting an apply batch of log records to the database...
InnoDB: Apply batch completed
InnoDB: Started
mysqld: ready for connections
```

For the cases of file system crashes or hardware problems, we can assume that the MySQL disk data is *not* available after a restart. This means that MySQL fails to start successfully because some blocks of disk data are no longer readable. In this case, it is necessary to reformat the disk, install a new one, or otherwise correct the underlying problem. Then it is necessary to recover our MySQL data from backups, which means that backups must already have been made. To make sure that is the case, design and implement a backup policy.

7.3.1 Establishing a Backup Policy

To be useful, backups must be scheduled regularly. A full backup (a snapshot of the data at a point in time) can be done in MySQL with several tools. For example, MySQL Enterprise Backup can perform a physical backup of an entire instance, with optimizations to minimize overhead and avoid disruption when backing up InnoDB data files; mysqldump provides online logical backup. This discussion uses mysqldump.

Assume that we make a full backup of all our InnoDB tables in all databases using the following command on Sunday at 1 p.m., when load is low:

```
shell> mysqldump --single-transaction --all-databases > backup_sunday_1_PM.sql
```

The resulting .sql file produced by mysqldump contains a set of SQL INSERT statements that can be used to reload the dumped tables at a later time.

This backup operation acquires a global read lock on all tables at the beginning of the dump (using FLUSH TABLES WITH READ LOCK). As soon as this lock has been acquired, the binary log coordinates are read and the lock is released. If long updating statements are running when the FLUSH statement is issued, the backup operation may stall until those statements finish. After that, the dump becomes lock-free and does not disturb reads and writes on the tables.

It was assumed earlier that the tables to back up are InnoDB tables, so --single-transaction uses a consistent read and guarantees that data seen by mysqldump does not change. (Changes made by other clients to InnoDB tables are not seen by the mysqldump process.) If the backup operation includes nontransactional tables, consistency requires that they do not change during the backup. For example, for the MyISAM tables in the mysql database, there must be no administrative changes to MySQL accounts during the backup.

Full backups are necessary, but it is not always convenient to create them. They produce large backup files and take time to generate. They are not optimal in the sense that each successive full backup includes all data, even that part that has not changed since the previous full backup. It is more efficient to make an initial full backup, and then to make incremental backups. The incremental backups are smaller and take less time to produce. The tradeoff is that, at recovery time, you cannot restore your data just by reloading the full backup. You must also process the incremental backups to recover the incremental changes.

To make incremental backups, we need to save the incremental changes. In MySQL, these changes are represented in the binary log, so the MySQL server should always be started with the `--log-bin` option to enable that log. With binary logging enabled, the server writes each data change into a file while it updates data. Looking at the data directory of a MySQL server that was started with the `--log-bin` option and that has been running for some days, we find these MySQL binary log files:

```
-rw-rw---- 1 guilhem  guilhem   1277324 Nov 10 23:59 gbichot2-bin.000001
-rw-rw---- 1 guilhem  guilhem         4 Nov 10 23:59 gbichot2-bin.000002
-rw-rw---- 1 guilhem  guilhem        79 Nov 11 11:06 gbichot2-bin.000003
-rw-rw---- 1 guilhem  guilhem       508 Nov 11 11:08 gbichot2-bin.000004
-rw-rw---- 1 guilhem  guilhem 220047446 Nov 12 16:47 gbichot2-bin.000005
-rw-rw---- 1 guilhem  guilhem    998412 Nov 14 10:08 gbichot2-bin.000006
-rw-rw---- 1 guilhem  guilhem       361 Nov 14 10:07 gbichot2-bin.index
```

Each time it restarts, the MySQL server creates a new binary log file using the next number in the sequence. While the server is running, you can also tell it to close the current binary log file and begin a new one manually by issuing a `FLUSH LOGS` SQL statement or with a `mysqladmin flush-logs` command. `mysqldump` also has an option to flush the logs. The `.index` file in the data directory contains the list of all MySQL binary logs in the directory.

The MySQL binary logs are important for recovery because they form the set of incremental backups. If you make sure to flush the logs when you make your full backup, the binary log files created afterward contain all the data changes made since the backup. Let's modify the previous `mysqldump` command a bit so that it flushes the MySQL binary logs at the moment of the full backup, and so that the dump file contains the name of the new current binary log:

```
shell> mysqldump --single-transaction --flush-logs --master-data=2 \
         --all-databases > backup_sunday_1_PM.sql
```

After executing this command, the data directory contains a new binary log file, `gbichot2-bin.000007`, because the `--flush-logs` option causes the server to flush its logs. The `--master-data` option causes `mysqldump` to write binary log information to its output, so the resulting `.sql` dump file includes these lines:

```
-- Position to start replication or point-in-time recovery from
-- CHANGE MASTER TO MASTER_LOG_FILE='gbichot2-bin.000007',MASTER_LOG_POS=4;
```

Because the `mysqldump` command made a full backup, those lines mean two things:

- The dump file contains all changes made before any changes written to the `gbichot2-bin.000007` binary log file or newer.

- All data changes logged after the backup are not present in the dump file, but are present in the `gbichot2-bin.000007` binary log file or newer.

On Monday at 1 p.m., we can create an incremental backup by flushing the logs to begin a new binary log file. For example, executing a `mysqladmin flush-logs` command creates `gbichot2-bin.000008`. All changes between the Sunday 1 p.m. full backup and Monday 1 p.m. will be in the `gbichot2-bin.000007` file. This incremental backup is important, so it is a good idea to copy it to a safe place.

(For example, back it up on tape or DVD, or copy it to another machine.) On Tuesday at 1 p.m., execute another mysqladmin flush-logs command. All changes between Monday 1 p.m. and Tuesday 1 p.m. will be in the gbichot2-bin.000008 file (which also should be copied somewhere safe).

The MySQL binary logs take up disk space. To free up space, purge them from time to time. One way to do this is by deleting the binary logs that are no longer needed, such as when we make a full backup:

```
shell> mysqldump --single-transaction --flush-logs --master-data=2 \
         --all-databases --delete-master-logs > backup_sunday_1_PM.sql
```

Note

Deleting the MySQL binary logs with mysqldump --delete-master-logs can be dangerous if your server is a replication master server, because slave servers might not yet fully have processed the contents of the binary log. The description for the PURGE BINARY LOGS statement explains what should be verified before deleting the MySQL binary logs. See Section 13.4.1.1, “PURGE BINARY LOGS Syntax”.

7.3.2 Using Backups for Recovery

Now, suppose that we have a catastrophic crash on Wednesday at 8 a.m. that requires recovery from backups. To recover, first we restore the last full backup we have (the one from Sunday 1 p.m.). The full backup file is just a set of SQL statements, so restoring it is very easy:

```
shell> mysql < backup_sunday_1_PM.sql
```

At this point, the data is restored to its state as of Sunday 1 p.m.. To restore the changes made since then, we must use the incremental backups; that is, the gbichot2-bin.000007 and gbichot2-bin.000008 binary log files. Fetch the files if necessary from where they were backed up, and then process their contents like this:

```
shell> mysqlbinlog gbichot2-bin.000007 gbichot2-bin.000008 | mysql
```

We now have recovered the data to its state as of Tuesday 1 p.m., but still are missing the changes from that date to the date of the crash. To not lose them, we would have needed to have the MySQL server store its MySQL binary logs into a safe location (RAID disks, SAN, ...) different from the place where it stores its data files, so that these logs were not on the destroyed disk. (That is, we can start the server with a --log-bin option that specifies a location on a different physical device from the one on which the data directory resides. That way, the logs are safe even if the device containing the directory is lost.) If we had done this, we would have the gbichot2-bin.000009 file (and any subsequent files) at hand, and we could apply them using mysqlbinlog and mysql to restore the most recent data changes with no loss up to the moment of the crash:

```
shell> mysqlbinlog gbichot2-bin.000009 ... | mysql
```

For more information about using mysqlbinlog to process binary log files, see Section 7.5, “Point-in-Time (Incremental) Recovery Using the Binary Log”.

7.3.3 Backup Strategy Summary

In case of an operating system crash or power failure, InnoDB itself does all the job of recovering data. But to make sure that you can sleep well, observe the following guidelines:

- Always run the MySQL server with the --log-bin option, or even --log-bin=*log_name*, where the log file name is located on some safe media different from the drive on which the data directory is located. If you have such safe media, this technique can also be good for disk load balancing (which results in a performance improvement).
- Make periodic full backups, using the mysqldump command shown earlier in Section 7.3.1, “Establishing a Backup Policy”, that makes an online, nonblocking backup.
- Make periodic incremental backups by flushing the logs with FLUSH LOGS or mysqladmin flush-logs.

7.4 Using mysqldump for Backups

This section describes how to use mysqldump to produce dump files, and how to reload dump files. A dump file can be used in several ways:

- As a backup to enable data recovery in case of data loss.
- As a source of data for setting up replication slaves.
- As a source of data for experimentation:
  - To make a copy of a database that you can use without changing the original data.
  - To test potential upgrade incompatibilities.

mysqldump produces two types of output, depending on whether the --tab option is given:

- Without --tab, mysqldump writes SQL statements to the standard output. This output consists of CREATE statements to create dumped objects (databases, tables, stored routines, and so forth), and INSERT statements to load data into tables. The output can be saved in a file and reloaded later using mysql to recreate the dumped objects. Options are available to modify the format of the SQL statements, and to control which objects are dumped.
- With --tab, mysqldump produces two output files for each dumped table. The server writes one file as tab-delimited text, one line per table row. This file is named *tbl_name*.txt in the output directory. The server also sends a CREATE TABLE statement for the table to mysqldump, which writes it as a file named *tbl_name*.sql in the output directory.

7.4.1 Dumping Data in SQL Format with mysqldump

This section describes how to use mysqldump to create SQL-format dump files. For information about reloading such dump files, see Section 7.4.2, “Reloading SQL-Format Backups”.

By default, mysqldump writes information as SQL statements to the standard output. You can save the output in a file:

```
shell> mysqldump [arguments] > file_name
```

To dump all databases, invoke mysqldump with the --all-databases option:

```
shell> mysqldump --all-databases > dump.sql
```

To dump only specific databases, name them on the command line and use the --databases option:

```
shell> mysqldump --databases db1 db2 db3 > dump.sql
```

The --databases option causes all names on the command line to be treated as database names. Without this option, mysqldump treats the first name as a database name and those following as table names.

With --all-databases or --databases, mysqldump writes CREATE DATABASE and USE statements prior to the dump output for each database. This ensures that when the dump file is reloaded, it creates each database if it does not exist and makes it the default database so database contents are loaded into the same database from which they came. If you want to cause the dump file to force a drop of each database before recreating it, use the --add-drop-database option as well. In this case, mysqldump writes a DROP DATABASE statement preceding each CREATE DATABASE statement.

To dump a single database, name it on the command line:

```
shell> mysqldump --databases test > dump.sql
```

In the single-database case, it is permissible to omit the --databases option:

```
shell> mysqldump test > dump.sql
```

The difference between the two preceding commands is that without --databases, the dump output contains no CREATE DATABASE or USE statements. This has several implications:

- When you reload the dump file, you must specify a default database name so that the server knows which database to reload.
- For reloading, you can specify a database name different from the original name, which enables you to reload the data into a different database.
- If the database to be reloaded does not exist, you must create it first.
- Because the output will contain no CREATE DATABASE statement, the --add-drop-database option has no effect. If you use it, it produces no DROP DATABASE statement.

To dump only specific tables from a database, name them on the command line following the database name:

```
shell> mysqldump test t1 t3 t7 > dump.sql
```

7.4.2 Reloading SQL-Format Backups

To reload a dump file written by mysqldump that consists of SQL statements, use it as input to the mysql client. If the dump file was created by mysqldump with the --all-databases or --databases option, it contains CREATE DATABASE and USE statements and it is not necessary to specify a default database into which to load the data:

```
shell> mysql < dump.sql
```

Alternatively, from within mysql, use a source command:

```
mysql> source dump.sql
```

If the file is a single-database dump not containing CREATE DATABASE and USE statements, create the database first (if necessary):

```
shell> mysqladmin create db1
```

Then specify the database name when you load the dump file:

```
shell> mysql db1 < dump.sql
```

Alternatively, from within mysql, create the database, select it as the default database, and load the dump file:

```
mysql> CREATE DATABASE IF NOT EXISTS db1;
mysql> USE db1;
mysql> source dump.sql
```

7.4.3 Dumping Data in Delimited-Text Format with mysqldump

This section describes how to use mysqldump to create delimited-text dump files. For information about reloading such dump files, see Section 7.4.4, “Reloading Delimited-Text Format Backups”.

If you invoke mysqldump with the --tab=*dir_name* option, it uses *dir_name* as the output directory and dumps tables individually in that directory using two files for each table. The table name is the basename for these files. For a table named t1, the files are named t1.sql and t1.txt. The .sql file contains a CREATE TABLE statement for the table. The .txt file contains the table data, one line per table row.

The following command dumps the contents of the db1 database to files in the /tmp database:

```
shell> mysqldump --tab=/tmp db1
```

The .txt files containing table data are written by the server, so they are owned by the system account used for running the server. The server uses SELECT ... INTO OUTFILE to write the files, so you must have the FILE privilege to perform this operation, and an error occurs if a given .txt file already exists.

The server sends the CREATE definitions for dumped tables to mysqldump, which writes them to .sql files. These files therefore are owned by the user who executes mysqldump.

It is best that --tab be used only for dumping a local server. If you use it with a remote server, the --tab directory must exist on both the local and remote hosts, and the .txt files will be written by the server in the remote directory (on the server host), whereas the .sql files will be written by mysqldump in the local directory (on the client host).

For mysqldump --tab, the server by default writes table data to .txt files one line per row with tabs between column values, no quotation marks around column values, and newline as the line terminator. (These are the same defaults as for SELECT ... INTO OUTFILE.)

To enable data files to be written using a different format, mysqldump supports these options:

- --fields-terminated-by=*str*

  The string for separating column values (default: tab).

- --fields-enclosed-by=*char*

  The character within which to enclose column values (default: no character).

- --fields-optionally-enclosed-by=*char*

  The character within which to enclose non-numeric column values (default: no character).

- `--fields-escaped-by=char`

  The character for escaping special characters (default: no escaping).

- `--lines-terminated-by=str`

  The line-termination string (default: newline).

Depending on the value you specify for any of these options, it might be necessary on the command line to quote or escape the value appropriately for your command interpreter. Alternatively, specify the value using hex notation. Suppose that you want `mysqldump` to quote column values within double quotation marks. To do so, specify double quote as the value for the `--fields-enclosed-by` option. But this character is often special to command interpreters and must be treated specially. For example, on Unix, you can quote the double quote like this:

```
--fields-enclosed-by='"'
```

On any platform, you can specify the value in hex:

```
--fields-enclosed-by=0x22
```

It is common to use several of the data-formatting options together. For example, to dump tables in comma-separated values format with lines terminated by carriage-return/newline pairs (`\r\n`), use this command (enter it on a single line):

```
shell> mysqldump --tab=/tmp --fields-terminated-by=,
         --fields-enclosed-by='"' --lines-terminated-by=0x0d0a db1
```

Should you use any of the data-formatting options to dump table data, you will need to specify the same format when you reload data files later, to ensure proper interpretation of the file contents.

7.4.4 Reloading Delimited-Text Format Backups

For backups produced with `mysqldump --tab`, each table is represented in the output directory by an `.sql` file containing the `CREATE TABLE` statement for the table, and a `.txt` file containing the table data. To reload a table, first change location into the output directory. Then process the `.sql` file with `mysql` to create an empty table and process the `.txt` file to load the data into the table:

```
shell> mysql db1 < t1.sql
shell> mysqlimport db1 t1.txt
```

An alternative to using `mysqlimport` to load the data file is to use the `LOAD DATA INFILE` statement from within the `mysql` client:

```
mysql> USE db1;
mysql> LOAD DATA INFILE 't1.txt' INTO TABLE t1;
```

If you used any data-formatting options with `mysqldump` when you initially dumped the table, you must use the same options with `mysqlimport` or `LOAD DATA INFILE` to ensure proper interpretation of the data file contents:

```
shell> mysqlimport --fields-terminated-by=,
         --fields-enclosed-by='"' --lines-terminated-by=0x0d0a db1 t1.txt
```

Or:

```
mysql> USE db1;
mysql> LOAD DATA INFILE 't1.txt' INTO TABLE t1
    -> FIELDS TERMINATED BY ',' FIELDS ENCLOSED BY '"'
    -> LINES TERMINATED BY '\r\n';
```

7.4.5 mysqldump Tips

This section surveys techniques that enable you to use mysqldump to solve specific problems:

- How to make a copy a database
- How to copy a database from one server to another
- How to dump stored programs (stored procedures and functions, triggers, and events)
- How to dump definitions and data separately

7.4.5.1 Making a Copy of a Database

```
shell> mysqldump db1 > dump.sql
shell> mysqladmin create db2
shell> mysql db2 < dump.sql
```

Do not use --databases on the mysqldump command line because that causes USE db1 to be included in the dump file, which overrides the effect of naming db2 on the mysql command line.

7.4.5.2 Copy a Database from one Server to Another

On Server 1:

```
shell> mysqldump --databases db1 > dump.sql
```

Copy the dump file from Server 1 to Server 2.

On Server 2:

```
shell> mysql < dump.sql
```

Use of --databases with the mysqldump command line causes the dump file to include CREATE DATABASE and USE statements that create the database if it does exist and make it the default database for the reloaded data.

Alternatively, you can omit --databases from the mysqldump command. Then you will need to create the database on Server 2 (if necessary) and specify it as the default database when you reload the dump file.

On Server 1:

```
shell> mysqldump db1 > dump.sql
```

On Server 2:

```
shell> mysqladmin create db1
shell> mysql db1 < dump.sql
```

You can specify a different database name in this case, so omitting --databases from the mysqldump command enables you to dump data from one database and load it into another.

7.4.5.3 Dumping Stored Programs

Several options control how mysqldump handles stored programs (stored procedures and functions, triggers, and events):

- --events: Dump Event Scheduler events
- --routines: Dump stored procedures and functions
- --triggers: Dump triggers for tables

The --triggers option is enabled by default so that when tables are dumped, they are accompanied by any triggers they have. The other options are disabled by default and must be specified explicitly to dump the corresponding objects. To disable any of these options explicitly, use its skip form: --skip-events, --skip-routines, or --skip-triggers.

7.4.5.4 Dumping Table Definitions and Content Separately

The --no-data option tells mysqldump not to dump table data, resulting in the dump file containing only statements to create the tables. Conversely, the --no-create-info option tells mysqldump to suppress CREATE statements from the output, so that the dump file contains only table data.

For example, to dump table definitions and data separately for the test database, use these commands:

```
shell> mysqldump --no-data test > dump-defs.sql
shell> mysqldump --no-create-info test > dump-data.sql
```

For a definition-only dump, add the --routines and --events options to also include stored routine and event definitions:

```
shell> mysqldump --no-data --routines --events test > dump-defs.sql
```

7.4.5.5 Using mysqldump to Test for Upgrade Incompatibilities

When contemplating a MySQL upgrade, it is prudent to install the newer version separately from your current production version. Then you can dump the database and database object definitions from the production server and load them into the new server to verify that they are handled properly. (This is also useful for testing downgrades.)

On the production server:

```
shell> mysqldump --all-databases --no-data --routines --events > dump-defs.sql
```

On the upgraded server:

```
shell> mysql < dump-defs.sql
```

Because the dump file does not contain table data, it can be processed quickly. This enables you to spot potential incompatibilities without waiting for lengthy data-loading operations. Look for warnings or errors while the dump file is being processed.

After you have verified that the definitions are handled properly, dump the data and try to load it into the upgraded server.

On the production server:

```
shell> mysqldump --all-databases --no-create-info > dump-data.sql
```

On the upgraded server:

```
shell> mysql < dump-data.sql
```

Now check the table contents and run some test queries.

7.5 Point-in-Time (Incremental) Recovery Using the Binary Log

Point-in-time recovery refers to recovery of data changes made since a given point in time. Typically, this type of recovery is performed after restoring a full backup that brings the server to its state as of the time the backup was made. (The full backup can be made in several ways, such as those listed in Section 7.2, “Database Backup Methods”.) Point-in-time recovery then brings the server up to date incrementally from the time of the full backup to a more recent time.

Point-in-time recovery is based on these principles:

- The source of information for point-in-time recovery is the set of incremental backups represented by the binary log files generated subsequent to the full backup operation. Therefore, the server must be started with the `--log-bin` option to enable binary logging (see Section 5.2.4, “The Binary Log”).

  To restore data from the binary log, you must know the name and location of the current binary log files. By default, the server creates binary log files in the data directory, but a path name can be specified with the `--log-bin` option to place the files in a different location. Section 5.2.4, “The Binary Log”.

  To see a listing of all binary log files, use this statement:

  ```
  mysql> SHOW BINARY LOGS;
  ```

  To determine the name of the current binary log file, issue the following statement:

  ```
  mysql> SHOW MASTER STATUS;
  ```

- The `mysqlbinlog` utility converts the events in the binary log files from binary format to text so that they can be executed or viewed. `mysqlbinlog` has options for selecting sections of the binary log based on event times or position of events within the log. See Section 4.6.7, “`mysqlbinlog` — Utility for Processing Binary Log Files”.

- Executing events from the binary log causes the data modifications they represent to be redone. This enables recovery of data changes for a given span of time. To execute events from the binary log, process `mysqlbinlog` output using the `mysql` client:

  ```
  shell> mysqlbinlog binlog_files | mysql -u root -p
  ```

- Viewing log contents can be useful when you need to determine event times or positions to select partial log contents prior to executing events. To view events from the log, send `mysqlbinlog` output into a paging program:

```
shell> mysqlbinlog binlog_files | more
```

Alternatively, save the output in a file and view the file in a text editor:

```
shell> mysqlbinlog binlog_files > tmpfile
shell> ... edit tmpfile ...
```

- Saving the output in a file is useful as a preliminary to executing the log contents with certain events removed, such as an accidental DROP DATABASE. You can delete from the file any statements not to be executed before executing its contents. After editing the file, execute the contents as follows:

```
shell> mysql -u root -p < tmpfile
```

If you have more than one binary log to execute on the MySQL server, the safe method is to process them all using a single connection to the server. Here is an example that demonstrates what may be *unsafe*:

```
shell> mysqlbinlog binlog.000001 | mysql -u root -p # DANGER!!
shell> mysqlbinlog binlog.000002 | mysql -u root -p # DANGER!!
```

Processing binary logs this way using different connections to the server causes problems if the first log file contains a CREATE TEMPORARY TABLE statement and the second log contains a statement that uses the temporary table. When the first mysql process terminates, the server drops the temporary table. When the second mysql process attempts to use the table, the server reports “unknown table.”

To avoid problems like this, use a *single* connection to execute the contents of all binary logs that you want to process. Here is one way to do so:

```
shell> mysqlbinlog binlog.000001 binlog.000002 | mysql -u root -p
```

Another approach is to write all the logs to a single file and then process the file:

```
shell> mysqlbinlog binlog.000001 >  /tmp/statements.sql
shell> mysqlbinlog binlog.000002 >> /tmp/statements.sql
shell> mysql -u root -p -e "source /tmp/statements.sql"
```

When writing to a dump file while reading back from a binary log containing GTIDs (see Section 16.1.3, “Replication with Global Transaction Identifiers”), use the --skip-gtids option with mysqlbinlog, like this:

```
shell> mysqlbinlog --skip-gtids binlog.000001 >  /tmp/dump.sql
shell> mysqlbinlog --skip-gtids binlog.000002 >> /tmp/dump.sql
shell> mysql -u root -p -e "source /tmp/dump.sql"
```

7.5.1 Point-in-Time Recovery Using Event Times

To indicate the start and end times for recovery, specify the --start-datetime and --stop-datetime options for mysqlbinlog, in DATETIME format. As an example, suppose that exactly at 10:00 a.m. on April 20, 2005 an SQL statement was executed that deleted a large table. To restore the table and data, you could restore the previous night's backup, and then execute the following command:

```
shell> mysqlbinlog --stop-datetime="2005-04-20 9:59:59" \
         /var/log/mysql/bin.123456 | mysql -u root -p
```

This command recovers all of the data up until the date and time given by the `--stop-datetime` option. If you did not detect the erroneous SQL statement that was entered until hours later, you will probably also want to recover the activity that occurred afterward. Based on this, you could run `mysqlbinlog` again with a start date and time, like so:

```
shell> mysqlbinlog --start-datetime="2005-04-20 10:01:00" \
         /var/log/mysql/bin.123456 | mysql -u root -p
```

In this command, the SQL statements logged from 10:01 a.m. on will be re-executed. The combination of restoring of the previous night's dump file and the two `mysqlbinlog` commands restores everything up until one second before 10:00 a.m. and everything from 10:01 a.m. on.

To use this method of point-in-time recovery, you should examine the log to be sure of the exact times to specify for the commands. To display the log file contents without executing them, use this command:

```
shell> mysqlbinlog /var/log/mysql/bin.123456 > /tmp/mysql_restore.sql
```

Then open the `/tmp/mysql_restore.sql` file with a text editor to examine it.

Excluding specific changes by specifying times for `mysqlbinlog` does not work well if multiple statements executed at the same time as the one to be excluded.

7.5.2 Point-in-Time Recovery Using Event Positions

Instead of specifying dates and times, the `--start-position` and `--stop-position` options for `mysqlbinlog` can be used for specifying log positions. They work the same as the start and stop date options, except that you specify log position numbers rather than dates. Using positions may enable you to be more precise about which part of the log to recover, especially if many transactions occurred around the same time as a damaging SQL statement. To determine the position numbers, run `mysqlbinlog` for a range of times near the time when the unwanted transaction was executed, but redirect the results to a text file for examination. This can be done like so:

```
shell> mysqlbinlog --start-datetime="2005-04-20 9:55:00" \
         --stop-datetime="2005-04-20 10:05:00" \
         /var/log/mysql/bin.123456 > /tmp/mysql_restore.sql
```

This command creates a small text file in the `/tmp` directory that contains the SQL statements around the time that the deleterious SQL statement was executed. Open this file with a text editor and look for the statement that you do not want to repeat. Determine the positions in the binary log for stopping and resuming the recovery and make note of them. Positions are labeled as `log_pos` followed by a number. After restoring the previous backup file, use the position numbers to process the binary log file. For example, you would use commands something like these:

```
shell> mysqlbinlog --stop-position=368312 /var/log/mysql/bin.123456 \
         | mysql -u root -p

shell> mysqlbinlog --start-position=368315 /var/log/mysql/bin.123456 \
         | mysql -u root -p
```

The first command recovers all the transactions up until the stop position given. The second command recovers all transactions from the starting position given until the end of the binary log. Because the output of `mysqlbinlog` includes `SET TIMESTAMP` statements before each SQL statement recorded, the recovered data and related MySQL logs will reflect the original times at which the transactions were executed.

7.6 MyISAM Table Maintenance and Crash Recovery

This section discusses how to use myisamchk to check or repair MyISAM tables (tables that have .MYD and .MYI files for storing data and indexes). For general myisamchk background, see Section 4.6.3, “myisamchk — MyISAM Table-Maintenance Utility”. Other table-repair information can be found at Section 2.10.4, “Rebuilding or Repairing Tables or Indexes”.

You can use myisamchk to check, repair, or optimize database tables. The following sections describe how to perform these operations and how to set up a table maintenance schedule. For information about using myisamchk to get information about your tables, see Section 4.6.3.5, “Obtaining Table Information with myisamchk”.

Even though table repair with myisamchk is quite secure, it is always a good idea to make a backup *before* doing a repair or any maintenance operation that could make a lot of changes to a table.

myisamchk operations that affect indexes can cause MyISAM FULLTEXT indexes to be rebuilt with full-text parameters that are incompatible with the values used by the MySQL server. To avoid this problem, follow the guidelines in Section 4.6.3.1, “myisamchk General Options”.

MyISAM table maintenance can also be done using the SQL statements that perform operations similar to what myisamchk can do:

- To check MyISAM tables, use CHECK TABLE.
- To repair MyISAM tables, use REPAIR TABLE.
- To optimize MyISAM tables, use OPTIMIZE TABLE.
- To analyze MyISAM tables, use ANALYZE TABLE.

For additional information about these statements, see Section 13.7.2, “Table Maintenance Statements”.

These statements can be used directly or by means of the mysqlcheck client program. One advantage of these statements over myisamchk is that the server does all the work. With myisamchk, you must make sure that the server does not use the tables at the same time so that there is no unwanted interaction between myisamchk and the server.

7.6.1 Using myisamchk for Crash Recovery

This section describes how to check for and deal with data corruption in MySQL databases. If your tables become corrupted frequently, you should try to find the reason why. See Section C.5.4.2, “What to Do If MySQL Keeps Crashing”.

For an explanation of how MyISAM tables can become corrupted, see Section 14.3.4, “MyISAM Table Problems”.

If you run mysqld with external locking disabled (which is the default), you cannot reliably use myisamchk to check a table when mysqld is using the same table. If you can be certain that no one will access the tables through mysqld while you run myisamchk, you only have to execute mysqladmin flush-tables before you start checking the tables. If you cannot guarantee this, you must stop mysqld while you check the tables. If you run myisamchk to check tables that mysqld is updating at the same time, you may get a warning that a table is corrupt even when it is not.

If the server is run with external locking enabled, you can use myisamchk to check tables at any time. In this case, if the server tries to update a table that myisamchk is using, the server will wait for myisamchk to finish before it continues.

If you use myisamchk to repair or optimize tables, you *must* always ensure that the mysqld server is not using the table (this also applies if external locking is disabled). If you do not stop mysqld, you should at least do a mysqladmin flush-tables before you run myisamchk. Your tables *may become corrupted* if the server and myisamchk access the tables simultaneously.

When performing crash recovery, it is important to understand that each MyISAM table *tbl_name* in a database corresponds to the three files in the database directory shown in the following table.

File	Purpose
tbl_name.frm	Definition (format) file
tbl_name.MYD	Data file
tbl_name.MYI	Index file

Each of these three file types is subject to corruption in various ways, but problems occur most often in data files and index files.

myisamchk works by creating a copy of the .MYD data file row by row. It ends the repair stage by removing the old .MYD file and renaming the new file to the original file name. If you use --quick, myisamchk does not create a temporary .MYD file, but instead assumes that the .MYD file is correct and generates only a new index file without touching the .MYD file. This is safe, because myisamchk automatically detects whether the .MYD file is corrupt and aborts the repair if it is. You can also specify the --quick option twice to myisamchk. In this case, myisamchk does not abort on some errors (such as duplicate-key errors) but instead tries to resolve them by modifying the .MYD file. Normally the use of two --quick options is useful only if you have too little free disk space to perform a normal repair. In this case, you should at least make a backup of the table before running myisamchk.

7.6.2 How to Check MyISAM Tables for Errors

To check a MyISAM table, use the following commands:

- myisamchk *tbl_name*

  This finds 99.99% of all errors. What it cannot find is corruption that involves *only* the data file (which is very unusual). If you want to check a table, you should normally run myisamchk without options or with the -s (silent) option.

- myisamchk -m *tbl_name*

  This finds 99.999% of all errors. It first checks all index entries for errors and then reads through all rows. It calculates a checksum for all key values in the rows and verifies that the checksum matches the checksum for the keys in the index tree.

- myisamchk -e *tbl_name*

  This does a complete and thorough check of all data (-e means “extended check”). It does a check-read of every key for each row to verify that they indeed point to the correct row. This may take a long time for a large table that has many indexes. Normally, myisamchk stops after the first error it finds. If you want to obtain more information, you can add the -v (verbose) option. This causes myisamchk to keep going, up through a maximum of 20 errors.

- myisamchk -e -i *tbl_name*

  This is like the previous command, but the -i option tells myisamchk to print additional statistical information.

In most cases, a simple myisamchk command with no arguments other than the table name is sufficient to check a table.

7.6.3 How to Repair MyISAM Tables

The discussion in this section describes how to use myisamchk on MyISAM tables (extensions .MYI and .MYD).

You can also use the CHECK TABLE and REPAIR TABLE statements to check and repair MyISAM tables. See Section 13.7.2.2, “CHECK TABLE Syntax”, and Section 13.7.2.5, “REPAIR TABLE Syntax”.

Symptoms of corrupted tables include queries that abort unexpectedly and observable errors such as these:

- *tbl_name*.frm is locked against change
- Can't find file *tbl_name*.MYI (Errcode: *nnn*)
- Unexpected end of file
- Record file is crashed
- Got error *nnn* from table handler

To get more information about the error, run perror *nnn*, where *nnn* is the error number. The following example shows how to use perror to find the meanings for the most common error numbers that indicate a problem with a table:

```
shell> perror 126 127 132 134 135 136 141 144 145
MySQL error code 126 = Index file is crashed
MySQL error code 127 = Record-file is crashed
MySQL error code 132 = Old database file
MySQL error code 134 = Record was already deleted (or record file crashed)
MySQL error code 135 = No more room in record file
MySQL error code 136 = No more room in index file
MySQL error code 141 = Duplicate unique key or constraint on write or update
MySQL error code 144 = Table is crashed and last repair failed
MySQL error code 145 = Table was marked as crashed and should be repaired
```

Note that error 135 (no more room in record file) and error 136 (no more room in index file) are not errors that can be fixed by a simple repair. In this case, you must use ALTER TABLE to increase the MAX_ROWS and AVG_ROW_LENGTH table option values:

```
ALTER TABLE tbl_name MAX_ROWS=xxx AVG_ROW_LENGTH=yyy;
```

If you do not know the current table option values, use SHOW CREATE TABLE.

For the other errors, you must repair your tables. myisamchk can usually detect and fix most problems that occur.

The repair process involves up to four stages, described here. Before you begin, you should change location to the database directory and check the permissions of the table files. On Unix, make sure that they are readable by the user that mysqld runs as (and to you, because you need to access the files you are checking). If it turns out you need to modify files, they must also be writable by you.

This section is for the cases where a table check fails (such as those described in Section 7.6.2, “How to Check MyISAM Tables for Errors”), or you want to use the extended features that myisamchk provides.

The myisamchk options used for table maintenance with are described in Section 4.6.3, "myisamchk — MyISAM Table-Maintenance Utility". myisamchk also has variables that you can set to control memory allocation that may improve performance. See Section 4.6.3.6, "myisamchk Memory Usage".

If you are going to repair a table from the command line, you must first stop the mysqld server. Note that when you do mysqladmin shutdown on a remote server, the mysqld server is still available for a while after mysqladmin returns, until all statement-processing has stopped and all index changes have been flushed to disk.

Stage 1: Checking your tables

Run myisamchk *.MYI or myisamchk -e *.MYI if you have more time. Use the -s (silent) option to suppress unnecessary information.

If the mysqld server is stopped, you should use the --update-state option to tell myisamchk to mark the table as "checked."

You have to repair only those tables for which myisamchk announces an error. For such tables, proceed to Stage 2.

If you get unexpected errors when checking (such as out of memory errors), or if myisamchk crashes, go to Stage 3.

Stage 2: Easy safe repair

First, try myisamchk -r -q *tbl_name* (-r -q means "quick recovery mode"). This attempts to repair the index file without touching the data file. If the data file contains everything that it should and the delete links point at the correct locations within the data file, this should work, and the table is fixed. Start repairing the next table. Otherwise, use the following procedure:

1. Make a backup of the data file before continuing.

2. Use myisamchk -r *tbl_name* (-r means "recovery mode"). This removes incorrect rows and deleted rows from the data file and reconstructs the index file.

3. If the preceding step fails, use myisamchk --safe-recover *tbl_name*. Safe recovery mode uses an old recovery method that handles a few cases that regular recovery mode does not (but is slower).

> **Note**
>
> If you want a repair operation to go much faster, you should set the values of the sort_buffer_size and key_buffer_size variables each to about 25% of your available memory when running myisamchk.

If you get unexpected errors when repairing (such as out of memory errors), or if myisamchk crashes, go to Stage 3.

Stage 3: Difficult repair

You should reach this stage only if the first 16KB block in the index file is destroyed or contains incorrect information, or if the index file is missing. In this case, it is necessary to create a new index file. Do so as follows:

1. Move the data file to a safe place.

2. Use the table description file to create new (empty) data and index files:

```
shell> mysql db_name
mysql> SET autocommit=1;
mysql> TRUNCATE TABLE tbl_name;
mysql> quit
```

3. Copy the old data file back onto the newly created data file. (Do not just move the old file back onto the new file. You want to retain a copy in case something goes wrong.)

⚠ **Important**

If you are using replication, you should stop it prior to performing the above procedure, since it involves file system operations, and these are not logged by MySQL.

Go back to Stage 2. `myisamchk -r -q` should work. (This should not be an endless loop.)

You can also use the `REPAIR TABLE tbl_name USE_FRM` SQL statement, which performs the whole procedure automatically. There is also no possibility of unwanted interaction between a utility and the server, because the server does all the work when you use `REPAIR TABLE`. See Section 13.7.2.5, “`REPAIR TABLE` Syntax”.

Stage 4: Very difficult repair

You should reach this stage only if the `.frm` description file has also crashed. That should never happen, because the description file is not changed after the table is created:

1. Restore the description file from a backup and go back to Stage 3. You can also restore the index file and go back to Stage 2. In the latter case, you should start with `myisamchk -r`.

2. If you do not have a backup but know exactly how the table was created, create a copy of the table in another database. Remove the new data file, and then move the `.frm` description and `.MYI` index files from the other database to your crashed database. This gives you new description and index files, but leaves the `.MYD` data file alone. Go back to Stage 2 and attempt to reconstruct the index file.

7.6.4 `MyISAM` Table Optimization

To coalesce fragmented rows and eliminate wasted space that results from deleting or updating rows, run `myisamchk` in recovery mode:

```
shell> myisamchk -r tbl_name
```

You can optimize a table in the same way by using the `OPTIMIZE TABLE` SQL statement. `OPTIMIZE TABLE` does a table repair and a key analysis, and also sorts the index tree so that key lookups are faster. There is also no possibility of unwanted interaction between a utility and the server, because the server does all the work when you use `OPTIMIZE TABLE`. See Section 13.7.2.4, “`OPTIMIZE TABLE` Syntax”.

`myisamchk` has a number of other options that you can use to improve the performance of a table:

- `--analyze` or `-a`: Perform key distribution analysis. This improves join performance by enabling the join optimizer to better choose the order in which to join the tables and which indexes it should use.

- `--sort-index` or `-S`: Sort the index blocks. This optimizes seeks and makes table scans that use indexes faster.

- `--sort-records=index_num` or `-R index_num`: Sort data rows according to a given index. This makes your data much more localized and may speed up range-based `SELECT` and `ORDER BY` operations that use this index.

For a full description of all available options, see Section 4.6.3, “myisamchk — MyISAM Table-Maintenance Utility”.

7.6.5 Setting Up a MyISAM Table Maintenance Schedule

It is a good idea to perform table checks on a regular basis rather than waiting for problems to occur. One way to check and repair MyISAM tables is with the CHECK TABLE and REPAIR TABLE statements. See Section 13.7.2, “Table Maintenance Statements”.

Another way to check tables is to use myisamchk. For maintenance purposes, you can use myisamchk -s. The -s option (short for --silent) causes myisamchk to run in silent mode, printing messages only when errors occur.

It is also a good idea to enable automatic MyISAM table checking. For example, whenever the machine has done a restart in the middle of an update, you usually need to check each table that could have been affected before it is used further. (These are “expected crashed tables.”) To cause the server to check MyISAM tables automatically, start it with the --myisam-recover-options option. See Section 5.1.3, “Server Command Options”.

You should also check your tables regularly during normal system operation. For example, you can run a cron job to check important tables once a week, using a line like this in a crontab file:

```
35 0 * * 0 /path/to/myisamchk --fast --silent /path/to/datadir/*/*.MYI
```

This prints out information about crashed tables so that you can examine and repair them as necessary.

To start with, execute myisamchk -s each night on all tables that have been updated during the last 24 hours. As you see that problems occur infrequently, you can back off the checking frequency to once a week or so.

Normally, MySQL tables need little maintenance. If you are performing many updates to MyISAM tables with dynamic-sized rows (tables with VARCHAR, BLOB, or TEXT columns) or have tables with many deleted rows you may want to defragment/reclaim space from the tables from time to time. You can do this by using OPTIMIZE TABLE on the tables in question. Alternatively, if you can stop the mysqld server for a while, change location into the data directory and use this command while the server is stopped:

```
shell> myisamchk -r -s --sort-index --myisam_sort_buffer_size=16M */*.MYI
```

Chapter 8 Optimization

Table of Contents

This chapter explains how to optimize MySQL performance and provides examples. Optimization involves configuring, tuning, and measuring performance, at several levels. Depending on your job role (developer, DBA, or a combination of both), you might optimize at the level of individual SQL statements, entire applications, a single database server, or multiple networked database servers. Sometimes you can be proactive and plan in advance for performance, while other times you might troubleshoot a configuration or code issue after a problem occurs. Optimizing CPU and memory usage can also improve scalability, allowing the database to handle more load without slowing down.

8.1 Optimization Overview

Database performance depends on several factors at the database level, such as tables, queries, and configuration settings. These software constructs result in CPU and I/O operations at the hardware level, which you must minimize and make as efficient as possible. As you work on database performance, you start by learning the high-level rules and guidelines for the software side, and measuring performance using wall-clock time. As you become an expert, you learn more about what happens internally, and start measuring things such as CPU cycles and I/O operations.

Typical users aim to get the best database performance out of their existing software and hardware configurations. Advanced users look for opportunities to improve the MySQL software itself, or develop their own storage engines and hardware appliances to expand the MySQL ecosystem.

Optimizing at the Database Level

The most important factor in making a database application fast is its basic design:

- Are the tables structured properly? In particular, do the columns have the right data types, and does each table have the appropriate columns for the type of work? For example, applications that perform frequent updates often have many tables with few columns, while applications that analyze large amounts of data often have few tables with many columns.

- Are the right indexes in place to make queries efficient?

- Are you using the appropriate storage engine for each table, and taking advantage of the strengths and features of each storage engine you use? In particular, the choice of a transactional storage engine such as InnoDB or a nontransactional one such as MyISAM can be very important for performance and scalability.

> **Note**
>
> In MySQL 5.5 and higher, InnoDB is the default storage engine for new tables. In practice, the advanced InnoDB performance features mean that InnoDB tables often outperform the simpler MyISAM tables, especially for a busy database.

- Does each table use an appropriate row format? This choice also depends on the storage engine used for the table. In particular, compressed tables use less disk space and so require less disk I/O to read and write the data. Compression is available for all kinds of workloads with InnoDB tables, and for read-only MyISAM tables.

- Does the application use an appropriate locking strategy? For example, by allowing shared access when possible so that database operations can run concurrently, and requesting exclusive access when appropriate so that critical operations get top priority. Again, the choice of storage engine is significant. The InnoDB storage engine handles most locking issues without involvement from you, allowing for better concurrency in the database and reducing the amount of experimentation and tuning for your code.

- Are all memory areas used for caching sized correctly? That is, large enough to hold frequently accessed data, but not so large that they overload physical memory and cause paging. The main memory areas to configure are the InnoDB buffer pool, the MyISAM key cache, and the MySQL query cache.

Optimizing at the Hardware Level

Any database application eventually hits hardware limits as the database becomes more and more busy. A DBA must evaluate whether it is possible to tune the application or reconfigure the server to avoid these bottlenecks, or whether more hardware resources are required. System bottlenecks typically arise from these sources:

- Disk seeks. It takes time for the disk to find a piece of data. With modern disks, the mean time for this is usually lower than 10ms, so we can in theory do about 100 seeks a second. This time improves slowly with new disks and is very hard to optimize for a single table. The way to optimize seek time is to distribute the data onto more than one disk.

- Disk reading and writing. When the disk is at the correct position, we need to read or write the data. With modern disks, one disk delivers at least 10–20MB/s throughput. This is easier to optimize than seeks because you can read in parallel from multiple disks.

- CPU cycles. When the data is in main memory, we must process it to get our result. Having small tables compared to the amount of memory is the most common limiting factor. But with small tables, speed is usually not the problem.

- Memory bandwidth. When the CPU needs more data than can fit in the CPU cache, main memory bandwidth becomes a bottleneck. This is an uncommon bottleneck for most systems, but one to be aware of.

Balancing Portability and Performance

To use performance-oriented SQL extensions in a portable MySQL program, you can wrap MySQL-specific keywords in a statement within `/*! */` comment delimiters. Other SQL servers ignore the commented keywords. For information about writing comments, see Section 9.6, “Comment Syntax”.

8.2 Optimizing SQL Statements

The core logic of a database application is performed through SQL statements, whether issued directly through an interpreter or submitted behind the scenes through an API. The tuning guidelines in this section help to speed up all kinds of MySQL applications. The guidelines cover SQL operations that read and write data, the behind-the-scenes overhead for SQL operations in general, and operations used in specific scenarios such as database monitoring.

8.2.1 Optimizing SELECT Statements

Queries, in the form of SELECT statements, perform all the lookup operations in the database. Tuning these statements is a top priority, whether to achieve sub-second response times for dynamic web pages, or to chop hours off the time to generate huge overnight reports.

8.2.1.1 Speed of SELECT Statements

The main considerations for optimizing queries are:

- To make a slow SELECT ... WHERE query faster, the first thing to check is whether you can add an index. Set up indexes on columns used in the WHERE clause, to speed up evaluation, filtering, and the final retrieval of results. To avoid wasted disk space, construct a small set of indexes that speed up many related queries used in your application.

  Indexes are especially important for queries that reference different tables, using features such as joins and foreign keys. You can use the EXPLAIN statement to determine which indexes are used for a SELECT. See Section 8.3.1, “How MySQL Uses Indexes” and Section 8.8.1, “Optimizing Queries with EXPLAIN”.

- Isolate and tune any part of the query, such as a function call, that takes excessive time. Depending on how the query is structured, a function could be called once for every row in the result set, or even once for every row in the table, greatly magnifying any inefficiency.

- Minimize the number of full table scans in your queries, particularly for big tables.

- Keep table statistics up to date by using the ANALYZE TABLE statement periodically, so the optimizer has the information needed to construct an efficient execution plan.

- Learn the tuning techniques, indexing techniques, and configuration parameters that are specific to the storage engine for each table. Both InnoDB and MyISAM have sets of guidelines for enabling and sustaining high performance in queries. For details, see Section 8.5.5, “Optimizing InnoDB Queries” and Section 8.6.1, “Optimizing MyISAM Queries”.

- You can optimize single-query transactions for InnoDB tables, using the technique in Optimizations for Read-Only Transactions.

- Avoid transforming the query in ways that make it hard to understand, especially if the optimizer does some of the same transformations automatically.

- If a performance issue is not easily solved by one of the basic guidelines, investigate the internal details of the specific query by reading the EXPLAIN plan and adjusting your indexes, WHERE clauses, join clauses, and so on. (When you reach a certain level of expertise, reading the EXPLAIN plan might be your first step for every query.)

- Adjust the size and properties of the memory areas that MySQL uses for caching. With efficient use of the InnoDB buffer pool, MyISAM key cache, and the MySQL query cache, repeated queries run faster because the results are retrieved from memory the second and subsequent times.

- Even for a query that runs fast using the cache memory areas, you might still optimize further so that they require less cache memory, making your application more scalable. Scalability means that your application can handle more simultaneous users, larger requests, and so on without experiencing a big drop in performance.
- Deal with locking issues, where the speed of your query might be affected by other sessions accessing the tables at the same time.

8.2.1.2 How MySQL Optimizes WHERE Clauses

This section discusses optimizations that can be made for processing WHERE clauses. The examples use SELECT statements, but the same optimizations apply for WHERE clauses in DELETE and UPDATE statements.

Note

Because work on the MySQL optimizer is ongoing, not all of the optimizations that MySQL performs are documented here.

You might be tempted to rewrite your queries to make arithmetic operations faster, while sacrificing readability. Because MySQL does similar optimizations automatically, you can often avoid this work, and leave the query in a more understandable and maintainable form. Some of the optimizations performed by MySQL follow:

- Removal of unnecessary parentheses:

```
   ((a AND b) AND c OR (((a AND b) AND (c AND d))))
-> (a AND b AND c) OR (a AND b AND c AND d)
```

- Constant folding:

```
   (a<b AND b=c) AND a=5
-> b>5 AND b=c AND a=5
```

- Constant condition removal (needed because of constant folding):

```
   (B>=5 AND B=5) OR (B=6 AND 5=5) OR (B=7 AND 5=6)
-> B=5 OR B=6
```

- Constant expressions used by indexes are evaluated only once.
- COUNT(*) on a single table without a WHERE is retrieved directly from the table information for MyISAM and MEMORY tables. This is also done for any NOT NULL expression when used with only one table.
- Early detection of invalid constant expressions. MySQL quickly detects that some SELECT statements are impossible and returns no rows.
- HAVING is merged with WHERE if you do not use GROUP BY or aggregate functions (COUNT(), MIN(), and so on).
- For each table in a join, a simpler WHERE is constructed to get a fast WHERE evaluation for the table and also to skip rows as soon as possible.
- All constant tables are read first before any other tables in the query. A constant table is any of the following:
  - An empty table or a table with one row.

  - A table that is used with a WHERE clause on a PRIMARY KEY or a UNIQUE index, where all index parts are compared to constant expressions and are defined as NOT NULL.

  All of the following tables are used as constant tables:

  ```
  SELECT * FROM t WHERE primary_key=1;
  SELECT * FROM t1,t2
    WHERE t1.primary_key=1 AND t2.primary_key=t1.id;
  ```

- The best join combination for joining the tables is found by trying all possibilities. If all columns in ORDER BY and GROUP BY clauses come from the same table, that table is preferred first when joining.

- If there is an ORDER BY clause and a different GROUP BY clause, or if the ORDER BY or GROUP BY contains columns from tables other than the first table in the join queue, a temporary table is created.

- If you use the SQL_SMALL_RESULT option, MySQL uses an in-memory temporary table.

- Each table index is queried, and the best index is used unless the optimizer believes that it is more efficient to use a table scan. At one time, a scan was used based on whether the best index spanned more than 30% of the table, but a fixed percentage no longer determines the choice between using an index or a scan. The optimizer now is more complex and bases its estimate on additional factors such as table size, number of rows, and I/O block size.

- In some cases, MySQL can read rows from the index without even consulting the data file. If all columns used from the index are numeric, only the index tree is used to resolve the query.

- Before each row is output, those that do not match the HAVING clause are skipped.

Some examples of queries that are very fast:

```
SELECT COUNT(*) FROM tbl_name;

SELECT MIN(key_part1),MAX(key_part1) FROM tbl_name;

SELECT MAX(key_part2) FROM tbl_name
  WHERE key_part1=constant;

SELECT ... FROM tbl_name
  ORDER BY key_part1,key_part2,... LIMIT 10;

SELECT ... FROM tbl_name
  ORDER BY key_part1 DESC, key_part2 DESC, ... LIMIT 10;
```

MySQL resolves the following queries using only the index tree, assuming that the indexed columns are numeric:

```
SELECT key_part1,key_part2 FROM tbl_name WHERE key_part1=val;

SELECT COUNT(*) FROM tbl_name
  WHERE key_part1=val1 AND key_part2=val2;

SELECT key_part2 FROM tbl_name GROUP BY key_part1;
```

The following queries use indexing to retrieve the rows in sorted order without a separate sorting pass:

```
SELECT ... FROM tbl_name
  ORDER BY key_part1,key_part2,... ;
```

```
SELECT ... FROM tbl_name
  ORDER BY key_part1 DESC, key_part2 DESC, ... ;
```

8.2.1.3 Range Optimization

The range access method uses a single index to retrieve a subset of table rows that are contained within one or several index value intervals. It can be used for a single-part or multiple-part index. The following sections give descriptions of conditions under which the optimizer uses range access.

The Range Access Method for Single-Part Indexes

For a single-part index, index value intervals can be conveniently represented by corresponding conditions in the WHERE clause, so we speak of *range conditions* rather than "intervals."

The definition of a range condition for a single-part index is as follows:

- For both BTREE and HASH indexes, comparison of a key part with a constant value is a range condition when using the =, <=>, IN(), IS NULL, or IS NOT NULL operators.
- Additionally, for BTREE indexes, comparison of a key part with a constant value is a range condition when using the >, <, >=, <=, BETWEEN, !=, or <> operators, or LIKE comparisons if the argument to LIKE is a constant string that does not start with a wildcard character.
- For all types of indexes, multiple range conditions combined with OR or AND form a range condition.

"Constant value" in the preceding descriptions means one of the following:

- A constant from the query string
- A column of a const or system table from the same join
- The result of an uncorrelated subquery
- Any expression composed entirely from subexpressions of the preceding types

Here are some examples of queries with range conditions in the WHERE clause:

```
SELECT * FROM t1
  WHERE key_col > 1
  AND key_col < 10;

SELECT * FROM t1
  WHERE key_col = 1
  OR key_col IN (15,18,20);

SELECT * FROM t1
  WHERE key_col LIKE 'ab%'
  OR key_col BETWEEN 'bar' AND 'foo';
```

Note that some nonconstant values may be converted to constants during the constant propagation phase.

MySQL tries to extract range conditions from the WHERE clause for each of the possible indexes. During the extraction process, conditions that cannot be used for constructing the range condition are dropped, conditions that produce overlapping ranges are combined, and conditions that produce empty ranges are removed.

Consider the following statement, where key1 is an indexed column and nonkey is not indexed:

```
SELECT * FROM t1 WHERE
  (key1 < 'abc' AND (key1 LIKE 'abcde%' OR key1 LIKE '%b')) OR
```

```
(key1 < 'bar' AND nonkey = 4) OR
(key1 < 'uux' AND key1 > 'z');
```

The extraction process for key key1 is as follows:

1. Start with original WHERE clause:

   ```
   (key1 < 'abc' AND (key1 LIKE 'abcde%' OR key1 LIKE '%b')) OR
   (key1 < 'bar' AND nonkey = 4) OR
   (key1 < 'uux' AND key1 > 'z')
   ```

2. Remove nonkey = 4 and key1 LIKE '%b' because they cannot be used for a range scan. The correct way to remove them is to replace them with TRUE, so that we do not miss any matching rows when doing the range scan. Having replaced them with TRUE, we get:

   ```
   (key1 < 'abc' AND (key1 LIKE 'abcde%' OR TRUE)) OR
   (key1 < 'bar' AND TRUE) OR
   (key1 < 'uux' AND key1 > 'z')
   ```

3. Collapse conditions that are always true or false:

   - (key1 LIKE 'abcde%' OR TRUE) is always true
   - (key1 < 'uux' AND key1 > 'z') is always false

   Replacing these conditions with constants, we get:

   ```
   (key1 < 'abc' AND TRUE) OR (key1 < 'bar' AND TRUE) OR (FALSE)
   ```

   Removing unnecessary TRUE and FALSE constants, we obtain:

   ```
   (key1 < 'abc') OR (key1 < 'bar')
   ```

4. Combining overlapping intervals into one yields the final condition to be used for the range scan:

   ```
   (key1 < 'bar')
   ```

In general (and as demonstrated by the preceding example), the condition used for a range scan is less restrictive than the WHERE clause. MySQL performs an additional check to filter out rows that satisfy the range condition but not the full WHERE clause.

The range condition extraction algorithm can handle nested AND/OR constructs of arbitrary depth, and its output does not depend on the order in which conditions appear in WHERE clause.

Currently, MySQL does not support merging multiple ranges for the range access method for spatial indexes. To work around this limitation, you can use a UNION with identical SELECT statements, except that you put each spatial predicate in a different SELECT.

The Range Access Method for Multiple-Part Indexes

Range conditions on a multiple-part index are an extension of range conditions for a single-part index. A range condition on a multiple-part index restricts index rows to lie within one or several key tuple intervals. Key tuple intervals are defined over a set of key tuples, using ordering from the index.

For example, consider a multiple-part index defined as key1(*key_part1*, *key_part2*, *key_part3*), and the following set of key tuples listed in key order:

```
key_part1  key_part2  key_part3
  NULL       1          'abc'
  NULL       1          'xyz'
  NULL       2          'foo'
   1         1          'abc'
   1         1          'xyz'
   1         2          'abc'
   2         1          'aaa'
```

The condition `key_part1 = 1` defines this interval:

```
(1,-inf,-inf) <= (key_part1,key_part2,key_part3) < (1,+inf,+inf)
```

The interval covers the 4th, 5th, and 6th tuples in the preceding data set and can be used by the range access method.

By contrast, the condition `key_part3 = 'abc'` does not define a single interval and cannot be used by the range access method.

The following descriptions indicate how range conditions work for multiple-part indexes in greater detail.

- For `HASH` indexes, each interval containing identical values can be used. This means that the interval can be produced only for conditions in the following form:

  ```
      key_part1 cmp const1
  AND key_part2 cmp const2
  AND ...
  AND key_partN cmp constN;
  ```

  Here, *const1*, *const2*, … are constants, *cmp* is one of the `=`, `<=>`, or `IS NULL` comparison operators, and the conditions cover all index parts. (That is, there are *N* conditions, one for each part of an *N*-part index.) For example, the following is a range condition for a three-part `HASH` index:

  ```
  key_part1 = 1 AND key_part2 IS NULL AND key_part3 = 'foo'
  ```

  For the definition of what is considered to be a constant, see The Range Access Method for Single-Part Indexes.

- For a `BTREE` index, an interval might be usable for conditions combined with `AND`, where each condition compares a key part with a constant value using `=`, `<=>`, `IS NULL`, `>`, `<`, `>=`, `<=`, `!=`, `<>`, `BETWEEN`, or `LIKE 'pattern'` (where `'pattern'` does not start with a wildcard). An interval can be used as long as it is possible to determine a single key tuple containing all rows that match the condition (or two intervals if `<>` or `!=` is used).

  The optimizer attempts to use additional key parts to determine the interval as long as the comparison operator is `=`, `<=>`, or `IS NULL`. If the operator is `>`, `<`, `>=`, `<=`, `!=`, `<>`, `BETWEEN`, or `LIKE`, the optimizer uses it but considers no more key parts. For the following expression, the optimizer uses `=` from the first comparison. It also uses `>=` from the second comparison but considers no further key parts and does not use the third comparison for interval construction:

  ```
  key_part1 = 'foo' AND key_part2 >= 10 AND key_part3 > 10
  ```

  The single interval is:

  ```
  ('foo',10,-inf) < (key_part1,key_part2,key_part3) < ('foo',+inf,+inf)
  ```

It is possible that the created interval contains more rows than the initial condition. For example, the preceding interval includes the value ('foo', 11, 0), which does not satisfy the original condition.

- If conditions that cover sets of rows contained within intervals are combined with OR, they form a condition that covers a set of rows contained within the union of their intervals. If the conditions are combined with AND, they form a condition that covers a set of rows contained within the intersection of their intervals. For example, for this condition on a two-part index:

  ```
  (key_part1 = 1 AND key_part2 < 2) OR (key_part1 > 5)
  ```

  The intervals are:

  ```
  (1,-inf) < (key_part1,key_part2) < (1,2)
  (5,-inf) < (key_part1,key_part2)
  ```

  In this example, the interval on the first line uses one key part for the left bound and two key parts for the right bound. The interval on the second line uses only one key part. The key_len column in the EXPLAIN output indicates the maximum length of the key prefix used.

  In some cases, key_len may indicate that a key part was used, but that might be not what you would expect. Suppose that *key_part1* and *key_part2* can be NULL. Then the key_len column displays two key part lengths for the following condition:

  ```
  key_part1 >= 1 AND key_part2 < 2
  ```

  But, in fact, the condition is converted to this:

  ```
  key_part1 >= 1 AND key_part2 IS NOT NULL
  ```

The Range Access Method for Single-Part Indexes, describes how optimizations are performed to combine or eliminate intervals for range conditions on a single-part index. Analogous steps are performed for range conditions on multiple-part indexes.

Equality Range Optimization of Many-Valued Comparisons

Consider these expressions, where *col_name* is an indexed column:

```
col_name IN(val1, ..., valN)
col_name = val1 OR ... OR col_name = valN
```

Each expression is true if *col_name* is equal to any of several values. These comparisons are equality range comparisons (where the “range” is a single value). The optimizer estimates the cost of reading qualifying rows for equality range comparisons as follows:

- If there is a unique index on *col_name*, the row estimate for each range is 1 because at most one row can have the given value.

- Otherwise, the optimizer can estimate the row count for each range using dives into the index or index statistics.

With index dives, the optimizer makes a dive at each end of a range and uses the number of rows in the range as the estimate. For example, the expression *col_name* IN (10, 20, 30) has three equality ranges and the optimizer makes two dives per range to generate a row estimate. Each pair of dives yields an estimate of the number of rows that have the given value.

Index dives provide accurate row estimates, but as the number of comparison values in the expression increases, the optimizer takes longer to generate a row estimate. Use of index statistics is less accurate than index dives but permits faster row estimation for large value lists.

The eq_range_index_dive_limit system variable enables you to configure the number of values at which the optimizer switches from one row estimation strategy to the other. To disable use of statistics and always use index dives, set eq_range_index_dive_limit to 0. To permit use of index dives for comparisons of up to *N* equality ranges, set eq_range_index_dive_limit to *N* + 1.

To update table index statistics for best estimates, use ANALYZE TABLE.

Range Optimization of Row Constructor Expressions

As of MySQL 5.7.3, the optimizer is able to apply the range scan access method to queries of this form:

```
SELECT ... FROM t1 WHERE ( col_1, col_2 ) IN (( 'a', 'b' ), ( 'c', 'd' ));
```

Previously, for range scans to be used it was necessary for the query to be written as:

```
SELECT ... FROM t1 WHERE ( col_1 = 'a' AND col_2 = 'b' )
OR ( col_1 = 'c' AND col_2 = 'd' );
```

For the optimizer to use a range scan, queries must satisfy these conditions:

- Only IN predicates can be used, not NOT IN.
- There may only be column references in the row constructor on the IN predicate's left hand side.
- There must be more than one row constructor on the IN predicate's right hand side.
- Row constructors on the IN predicate's right hand side must contain only runtime constants, which are either literals or local column references that are bound to constants during execution.

Compared to similar queries executed before MySQL 5.7.3, EXPLAIN output for applicable queries changes from full table or index scan to range scan. Changes are also visible by checking the values of the Handler_read_first, Handler_read_key, and Handler_read_next status variables.

8.2.1.4 Index Merge Optimization

The *Index Merge* method is used to retrieve rows with several range scans and to merge their results into one. The merge can produce unions, intersections, or unions-of-intersections of its underlying scans. This access method merges index scans from a single table; it does not merge scans across multiple tables.

In EXPLAIN output, the Index Merge method appears as index_merge in the type column. In this case, the key column contains a list of indexes used, and key_len contains a list of the longest key parts for those indexes.

Examples:

```
SELECT * FROM tbl_name WHERE key1 = 10 OR key2 = 20;

SELECT * FROM tbl_name
  WHERE (key1 = 10 OR key2 = 20) AND non_key=30;

SELECT * FROM t1, t2
```

```
  WHERE (t1.key1 IN (1,2) OR t1.key2 LIKE 'value%')
  AND t2.key1=t1.some_col;

SELECT * FROM t1, t2
  WHERE t1.key1=1
  AND (t2.key1=t1.some_col OR t2.key2=t1.some_col2);
```

The Index Merge method has several access algorithms (seen in the `Extra` field of `EXPLAIN` output):

- `Using intersect(...)`
- `Using union(...)`
- `Using sort_union(...)`

The following sections describe these methods in greater detail.

Note

The Index Merge optimization algorithm has the following known deficiencies:

- If your query has a complex `WHERE` clause with deep `AND`/`OR` nesting and MySQL doesn't choose the optimal plan, try distributing terms using the following identity laws:

  ```
  (x AND y) OR z = (x OR z) AND (y OR z)
  (x OR y) AND z = (x AND z) OR (y AND z)
  ```

- Index Merge is not applicable to full-text indexes. We plan to extend it to cover these in a future MySQL release.

The choice between different possible variants of the Index Merge access method and other access methods is based on cost estimates of various available options.

The Index Merge Intersection Access Algorithm

This access algorithm can be employed when a `WHERE` clause was converted to several range conditions on different keys combined with `AND`, and each condition is one of the following:

- In this form, where the index has exactly *N* parts (that is, all index parts are covered):

  ```
  key_part1=const1 AND key_part2=const2 ... AND key_partN=constN
  ```

- Any range condition over a primary key of an `InnoDB` table.

Examples:

```
SELECT * FROM innodb_table WHERE primary_key < 10 AND key_col1=20;

SELECT * FROM tbl_name
  WHERE (key1_part1=1 AND key1_part2=2) AND key2=2;
```

The Index Merge intersection algorithm performs simultaneous scans on all used indexes and produces the intersection of row sequences that it receives from the merged index scans.

If all columns used in the query are covered by the used indexes, full table rows are not retrieved (`EXPLAIN` output contains `Using index` in `Extra` field in this case). Here is an example of such a query:

```
SELECT COUNT(*) FROM t1 WHERE key1=1 AND key2=1;
```

If the used indexes don't cover all columns used in the query, full rows are retrieved only when the range conditions for all used keys are satisfied.

If one of the merged conditions is a condition over a primary key of an InnoDB table, it is not used for row retrieval, but is used to filter out rows retrieved using other conditions.

The Index Merge Union Access Algorithm

The applicability criteria for this algorithm are similar to those for the Index Merge method intersection algorithm. The algorithm can be employed when the table's WHERE clause was converted to several range conditions on different keys combined with OR, and each condition is one of the following:

- In this form, where the index has exactly *N* parts (that is, all index parts are covered):

  ```
  key_part1=const1 AND key_part2=const2 ... AND key_partN=constN
  ```

- Any range condition over a primary key of an InnoDB table.
- A condition for which the Index Merge method intersection algorithm is applicable.

Examples:

```
SELECT * FROM t1 WHERE key1=1 OR key2=2 OR key3=3;

SELECT * FROM innodb_table WHERE (key1=1 AND key2=2) OR
  (key3='foo' AND key4='bar') AND key5=5;
```

The Index Merge Sort-Union Access Algorithm

This access algorithm is employed when the WHERE clause was converted to several range conditions combined by OR, but for which the Index Merge method union algorithm is not applicable.

Examples:

```
SELECT * FROM tbl_name WHERE key_col1 < 10 OR key_col2 < 20;

SELECT * FROM tbl_name
  WHERE (key_col1 > 10 OR key_col2 = 20) AND nonkey_col=30;
```

The difference between the sort-union algorithm and the union algorithm is that the sort-union algorithm must first fetch row IDs for all rows and sort them before returning any rows.

8.2.1.5 Engine Condition Pushdown Optimization

This optimization improves the efficiency of direct comparisons between a nonindexed column and a constant. In such cases, the condition is “pushed down” to the storage engine for evaluation. This optimization can be used only by the NDB storage engine.

Note

The NDB storage engine is currently not available in MySQL 5.7. If you are interested in using MySQL Cluster, see MySQL Cluster NDB 7.2, which provides

> information about MySQL Cluster NDB 7.2, which is based on MySQL 5.5 but contains the latest improvements and fixes for `NDBCLUSTER`.

For MySQL Cluster, this optimization can eliminate the need to send nonmatching rows over the network between the cluster's data nodes and the MySQL Server that issued the query, and can speed up queries where it is used by a factor of 5 to 10 times over cases where condition pushdown could be but is not used.

Suppose that a MySQL Cluster table is defined as follows:

```
CREATE TABLE t1 (
    a INT,
    b INT,
    KEY(a)
) ENGINE=NDB;
```

Condition pushdown can be used with queries such as the one shown here, which includes a comparison between a nonindexed column and a constant:

```
SELECT a, b FROM t1 WHERE b = 10;
```

The use of condition pushdown can be seen in the output of `EXPLAIN`:

```
mysql> EXPLAIN SELECT a,b FROM t1 WHERE b = 10\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: t1
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 10
        Extra: Using where with pushed condition
```

However, condition pushdown *cannot* be used with either of these two queries:

```
SELECT a,b FROM t1 WHERE a = 10;
SELECT a,b FROM t1 WHERE b + 1 = 10;
```

Condition pushdown is not applicable to the first query because an index exists on column `a`. (An index access method would be more efficient and so would be chosen in preference to condition pushdown.) Condition pushdown cannot be employed for the second query because the comparison involving the nonindexed column `b` is indirect. (However, condition pushdown could be applied if you were to reduce `b + 1 = 10` to `b = 9` in the `WHERE` clause.)

Condition pushdown may also be employed when an indexed column is compared with a constant using a `>` or `<` operator:

```
mysql> EXPLAIN SELECT a, b FROM t1 WHERE a < 2\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: t1
         type: range
possible_keys: a
          key: a
```

```
     key_len: 5
         ref: NULL
        rows: 2
       Extra: Using where with pushed condition
```

Other supported comparisons for condition pushdown include the following:

- *column* [NOT] LIKE *pattern*

  pattern must be a string literal containing the pattern to be matched; for syntax, see Section 12.5.1, “String Comparison Functions”.

- *column* IS [NOT] NULL

- *column* IN (*value_list*)

  Each item in the *value_list* must be a constant, literal value.

- *column* BETWEEN *constant1* AND *constant2*

  constant1 and *constant2* must each be a constant, literal value.

In all of the cases in the preceding list, it is possible for the condition to be converted into the form of one or more direct comparisons between a column and a constant.

Engine condition pushdown is enabled by default. To disable it at server startup, set the optimizer_switch system variable. For example, in a my.cnf file, use these lines:

```
[mysqld]
optimizer_switch=engine_condition_pushdown=off
```

At runtime, enable condition pushdown like this:

```
SET optimizer_switch='engine_condition_pushdown=off';
```

Limitations. Engine condition pushdown is subject to the following limitations:

- Condition pushdown is supported only by the NDB storage engine.
- Columns may be compared with constants only; however, this includes expressions which evaluate to constant values.
- Columns used in comparisons cannot be of any of the BLOB or TEXT types.
- A string value to be compared with a column must use the same collation as the column.
- Joins are not directly supported; conditions involving multiple tables are pushed separately where possible. Use EXPLAIN EXTENDED to determine which conditions are actually pushed down.

8.2.1.6 Index Condition Pushdown Optimization

Index Condition Pushdown (ICP) is an optimization for the case where MySQL retrieves rows from a table using an index. Without ICP, the storage engine traverses the index to locate rows in the base table and returns them to the MySQL server which evaluates the WHERE condition for the rows. With ICP enabled, and if parts of the WHERE condition can be evaluated by using only fields from the index, the MySQL server pushes this part of the WHERE condition down to the storage engine. The storage engine then evaluates the pushed index condition by using the index entry and only if this is satisfied is the row read from the table.

ICP can reduce the number of times the storage engine must access the base table and the number of times the MySQL server must access the storage engine.

Index Condition Pushdown optimization is used for the range, ref, eq_ref, and ref_or_null access methods when there is a need to access full table rows. This strategy can be used for InnoDB and MyISAM tables. Beginning with MySQL 5.7.3, it can also be used with partitioned InnoDB and MyISAM tables (Bug #17306882, Bug #70001). For InnoDB tables, however, ICP is used only for secondary indexes. The goal of ICP is to reduce the number of full-record reads and thereby reduce IO operations. For InnoDB clustered indexes, the complete record is already read into the InnoDB buffer. Using ICP in this case does not reduce IO.

To see how this optimization works, consider first how an index scan proceeds when Index Condition Pushdown is not used:

1. Get the next row, first by reading the index tuple, and then by using the index tuple to locate and read the full table row.

2. Test the part of the WHERE condition that applies to this table. Accept or reject the row based on the test result.

When Index Condition Pushdown is used, the scan proceeds like this instead:

1. Get the next row's index tuple (but not the full table row).

2. Test the part of the WHERE condition that applies to this table and can be checked using only index columns. If the condition is not satisfied, proceed to the index tuple for the next row.

3. If the condition is satisfied, use the index tuple to locate and read the full table row.

4. Test the remaining part of the WHERE condition that applies to this table. Accept or reject the row based on the test result.

When Index Condition Pushdown is used, the Extra column in EXPLAIN output shows Using index condition. It will not show Index only because that does not apply when full table rows must be read.

Suppose that we have a table containing information about people and their addresses and that the table has an index defined as INDEX (zipcode, lastname, firstname). If we know a person's zipcode value but are not sure about the last name, we can search like this:

```
SELECT * FROM people
  WHERE zipcode='95054'
  AND lastname LIKE '%etrunia%'
  AND address LIKE '%Main Street%';
```

MySQL can use the index to scan through people with zipcode='95054'. The second part (lastname LIKE '%etrunia%') cannot be used to limit the number of rows that must be scanned, so without Index Condition Pushdown, this query must retrieve full table rows for all the people who have zipcode='95054'.

With Index Condition Pushdown, MySQL will check the lastname LIKE '%etrunia%' part before reading the full table row. This avoids reading full rows corresponding to all index tuples that do not match the lastname condition.

Index Condition Pushdown is enabled by default; it can be controlled with the optimizer_switch system variable by setting the index_condition_pushdown flag. See Section 8.8.6.2, “Controlling Switchable Optimizations”.

8.2.1.7 Use of Index Extensions

InnoDB automatically extends each secondary index by appending the primary key columns to it. Consider this table definition:

```
CREATE TABLE t1 (
  i1 INT NOT NULL DEFAULT 0,
  i2 INT NOT NULL DEFAULT 0,
  d DATE DEFAULT NULL,
  PRIMARY KEY (i1, i2),
  INDEX k_d (d)
) ENGINE = InnoDB;
```

This table defines the primary key on columns (i1, i2). It also defines a secondary index k_d on column (d), but internally InnoDB extends this index and treats it as columns (d, i1, i2).

In MySQL 5.7, the optimizer takes into account the primary key columns of the extended secondary index when determining how and whether to use that index. This can result in more efficient query execution plans and better performance.

The optimizer can use extended secondary indexes for ref, range, and index_merge index access, for loose index scans, for join and sorting optimization, and for MIN()/MAX() optimization.

The following example shows how execution plans are affected by whether the optimizer uses extended secondary indexes. Suppose that t1 is populated with these rows:

```
INSERT INTO t1 VALUES
(1, 1, '1998-01-01'), (1, 2, '1999-01-01'),
(1, 3, '2000-01-01'), (1, 4, '2001-01-01'),
(1, 5, '2002-01-01'), (2, 1, '1998-01-01'),
(2, 2, '1999-01-01'), (2, 3, '2000-01-01'),
(2, 4, '2001-01-01'), (2, 5, '2002-01-01'),
(3, 1, '1998-01-01'), (3, 2, '1999-01-01'),
(3, 3, '2000-01-01'), (3, 4, '2001-01-01'),
(3, 5, '2002-01-01'), (4, 1, '1998-01-01'),
(4, 2, '1999-01-01'), (4, 3, '2000-01-01'),
(4, 4, '2001-01-01'), (4, 5, '2002-01-01'),
(5, 1, '1998-01-01'), (5, 2, '1999-01-01'),
(5, 3, '2000-01-01'), (5, 4, '2001-01-01'),
(5, 5, '2002-01-01');
```

Now consider this query:

```
EXPLAIN SELECT COUNT(*) FROM t1 WHERE i1 = 3 AND d = '2000-01-01'
```

The optimizer cannot use the primary key in this case because that comprises columns (i1, i2) and the query does not refer to i2. Instead, the optimizer can use the secondary index k_d on (d), and the execution plan depends on whether the extended index is used.

When the optimizer does not consider index extensions, it treats the index k_d as only (d). EXPLAIN for the query produces this result:

```
mysql> EXPLAIN SELECT COUNT(*) FROM t1 WHERE i1 = 3 AND d = '2000-01-01'\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: t1
```

```
         type: ref
possible_keys: PRIMARY,k_d
          key: k_d
      key_len: 4
          ref: const
         rows: 5
        Extra: Using where; Using index
```

When the optimizer takes index extensions into account, it treats `k_d` as `(d, i1, i2)`. In this case, it can use the leftmost index prefix `(d, i1)` to produce a better execution plan:

```
mysql> EXPLAIN SELECT COUNT(*) FROM t1 WHERE i1 = 3 AND d = '2000-01-01'\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: t1
         type: ref
possible_keys: PRIMARY,k_d
          key: k_d
      key_len: 8
          ref: const,const
         rows: 1
        Extra: Using index
```

In both cases, `key` indicates that the optimizer will use secondary index `k_d` but the `EXPLAIN` output shows these improvements from using the extended index:

- `key_len` goes from 4 bytes to 8 bytes, indicating that key lookups use columns `d` and `i1`, not just `d`.
- The `ref` value changes from `const` to `const,const` because the key lookup uses two key parts, not one.
- The `rows` count decreases from 5 to 1, indicating that `InnoDB` should need to examine fewer rows to produce the result.
- The `Extra` value changes from `Using where; Using index` to `Using index`. This means that rows can be read using only the index, without consulting columns in the data row.

Differences in optimizer behavior for use of extended indexes can also be seen with `SHOW STATUS`:

```
FLUSH TABLE t1;
FLUSH STATUS;
SELECT COUNT(*) FROM t1 WHERE i1 = 3 AND d = '2000-01-01';
SHOW STATUS LIKE 'handler_read%'
```

The preceding statements include `FLUSH TABLE` and `FLUSH STATUS` to flush the table cache and clear the status counters.

Without index extensions, `SHOW STATUS` produces this result:

```
+-----------------------+-------+
| Variable_name         | Value |
+-----------------------+-------+
| Handler_read_first    | 0     |
| Handler_read_key      | 1     |
| Handler_read_last     | 0     |
| Handler_read_next     | 5     |
| Handler_read_prev     | 0     |
| Handler_read_rnd      | 0     |
| Handler_read_rnd_next | 0     |
+-----------------------+-------+
```

With index extensions, SHOW STATUS produces this result. The Handler_read_next value decreases from 5 to 1, indicating more efficient use of the index:

```
+-----------------------+-------+
| Variable_name         | Value |
+-----------------------+-------+
| Handler_read_first    | 0     |
| Handler_read_key      | 1     |
| Handler_read_last     | 0     |
| Handler_read_next     | 1     |
| Handler_read_prev     | 0     |
| Handler_read_rnd      | 0     |
| Handler_read_rnd_next | 0     |
+-----------------------+-------+
```

The use_index_extensions flag of the optimizer_switch system variable permits control over whether the optimizer takes the primary key columns into account when determining how to use an InnoDB table's secondary indexes. By default, use_index_extensions is enabled. To check whether disabling use of index extensions will improve performance, use this statement:

```
SET optimizer_switch = 'use_index_extensions=off';
```

Use of index extensions by the optimizer is subject to the usual limits on the number of key parts in an index (16) and the maximum key length (3072 bytes).

8.2.1.8 IS NULL Optimization

MySQL can perform the same optimization on *col_name* IS NULL that it can use for *col_name* = *constant_value*. For example, MySQL can use indexes and ranges to search for NULL with IS NULL.

Examples:

```
SELECT * FROM tbl_name WHERE key_col IS NULL;

SELECT * FROM tbl_name WHERE key_col <=> NULL;

SELECT * FROM tbl_name
  WHERE key_col=const1 OR key_col=const2 OR key_col IS NULL;
```

If a WHERE clause includes a *col_name* IS NULL condition for a column that is declared as NOT NULL, that expression is optimized away. This optimization does not occur in cases when the column might produce NULL anyway; for example, if it comes from a table on the right side of a LEFT JOIN.

MySQL can also optimize the combination *col_name* = *expr* OR *col_name* IS NULL, a form that is common in resolved subqueries. EXPLAIN shows ref_or_null when this optimization is used.

This optimization can handle one IS NULL for any key part.

Some examples of queries that are optimized, assuming that there is an index on columns a and b of table t2:

```
SELECT * FROM t1 WHERE t1.a=expr OR t1.a IS NULL;

SELECT * FROM t1, t2 WHERE t1.a=t2.a OR t2.a IS NULL;
SELECT * FROM t1, t2
  WHERE (t1.a=t2.a OR t2.a IS NULL) AND t2.b=t1.b;

SELECT * FROM t1, t2
  WHERE t1.a=t2.a AND (t2.b=t1.b OR t2.b IS NULL);

SELECT * FROM t1, t2
  WHERE (t1.a=t2.a AND t2.a IS NULL AND ...)
  OR (t1.a=t2.a AND t2.a IS NULL AND ...);
```

`ref_or_null` works by first doing a read on the reference key, and then a separate search for rows with a `NULL` key value.

Note that the optimization can handle only one `IS NULL` level. In the following query, MySQL uses key lookups only on the expression `(t1.a=t2.a AND t2.a IS NULL)` and is not able to use the key part on `b`:

```
SELECT * FROM t1, t2
  WHERE (t1.a=t2.a AND t2.a IS NULL)
  OR (t1.b=t2.b AND t2.b IS NULL);
```

8.2.1.9 LEFT JOIN and RIGHT JOIN Optimization

MySQL implements an *`A`* `LEFT JOIN` *`B`* `join_condition` as follows:

- Table *B* is set to depend on table *A* and all tables on which *A* depends.

- Table *A* is set to depend on all tables (except *B*) that are used in the `LEFT JOIN` condition.

- The `LEFT JOIN` condition is used to decide how to retrieve rows from table *B*. (In other words, any condition in the `WHERE` clause is not used.)

- All standard join optimizations are performed, with the exception that a table is always read after all tables on which it depends. If there is a circular dependence, MySQL issues an error.

- All standard `WHERE` optimizations are performed.

- If there is a row in *A* that matches the `WHERE` clause, but there is no row in *B* that matches the `ON` condition, an extra *B* row is generated with all columns set to `NULL`.

- If you use `LEFT JOIN` to find rows that do not exist in some table and you have the following test: *`col_name`* `IS NULL` in the `WHERE` part, where *`col_name`* is a column that is declared as `NOT NULL`, MySQL stops searching for more rows (for a particular key combination) after it has found one row that matches the `LEFT JOIN` condition.

The implementation of `RIGHT JOIN` is analogous to that of `LEFT JOIN` with the roles of the tables reversed.

The join optimizer calculates the order in which tables should be joined. The table read order forced by `LEFT JOIN` or `STRAIGHT_JOIN` helps the join optimizer do its work much more quickly, because there are fewer table permutations to check. Note that this means that if you do a query of the following type, MySQL does a full scan on `b` because the `LEFT JOIN` forces it to be read before `d`:

```
SELECT *
  FROM a JOIN b LEFT JOIN c ON (c.key=a.key)
  LEFT JOIN d ON (d.key=a.key)
  WHERE b.key=d.key;
```

The fix in this case is reverse the order in which a and b are listed in the FROM clause:

```
SELECT *
  FROM b JOIN a LEFT JOIN c ON (c.key=a.key)
  LEFT JOIN d ON (d.key=a.key)
  WHERE b.key=d.key;
```

For a LEFT JOIN, if the WHERE condition is always false for the generated NULL row, the LEFT JOIN is changed to a normal join. For example, the WHERE clause would be false in the following query if t2.column1 were NULL:

```
SELECT * FROM t1 LEFT JOIN t2 ON (column1) WHERE t2.column2=5;
```

Therefore, it is safe to convert the query to a normal join:

```
SELECT * FROM t1, t2 WHERE t2.column2=5 AND t1.column1=t2.column1;
```

This can be made faster because MySQL can use table t2 before table t1 if doing so would result in a better query plan. To provide a hint about the table join order, use STRAIGHT_JOIN. (See Section 13.2.9, “SELECT Syntax”.)

8.2.1.10 Nested-Loop Join Algorithms

MySQL executes joins between tables using a nested-loop algorithm or variations on it.

Nested-Loop Join Algorithm

A simple nested-loop join (NLJ) algorithm reads rows from the first table in a loop one at a time, passing each row to a nested loop that processes the next table in the join. This process is repeated as many times as there remain tables to be joined.

Assume that a join between three tables t1, t2, and t3 is to be executed using the following join types:

```
Table   Join Type
t1      range
t2      ref
t3      ALL
```

If a simple NLJ algorithm is used, the join is processed like this:

```
for each row in t1 matching range {
  for each row in t2 matching reference key {
    for each row in t3 {
      if row satisfies join conditions,
      send to client
    }
  }
}
```

Because the NLJ algorithm passes rows one at a time from outer loops to inner loops, it typically reads tables processed in the inner loops many times.

Block Nested-Loop Join Algorithm

A Block Nested-Loop (BNL) join algorithm uses buffering of rows read in outer loops to reduce the number of times that tables in inner loops must be read. For example, if 10 rows are read into a buffer and the buffer is passed to the next inner loop, each row read in the inner loop can be compared against all 10 rows in the buffer. The reduces the number of times the inner table must be read by an order of magnitude.

MySQL uses join buffering under these conditions:

- The join_buffer_size system variable determines the size of each join buffer.
- Join buffering can be used when the join is of type ALL or index (in other words, when no possible keys can be used, and a full scan is done, of either the data or index rows, respectively), or range. In MySQL 5.7, use of buffering is extended to be applicable to outer joins, as described in Section 8.2.1.14, "Block Nested-Loop and Batched Key Access Joins".
- One buffer is allocated for each join that can be buffered, so a given query might be processed using multiple join buffers.
- A join buffer is never allocated for the first nonconst table, even if it would be of type ALL or index.
- A join buffer is allocated prior to executing the join and freed after the query is done.
- Only columns of interest to the join are stored in the join buffer, not whole rows.

For the example join described previously for the NLJ algorithm (without buffering), the join is done as follow using join buffering:

```
for each row in t1 matching range {
  for each row in t2 matching reference key {
    store used columns from t1, t2 in join buffer
    if buffer is full {
      for each row in t3 {
        for each t1, t2 combination in join buffer {
          if row satisfies join conditions,
          send to client
        }
      }
      empty buffer
    }
  }
}

if buffer is not empty {
  for each row in t3 {
    for each t1, t2 combination in join buffer {
      if row satisfies join conditions,
      send to client
    }
  }
}
```

If *S* is the size of each stored t1, t2 combination is the join buffer and *C* is the number of combinations in the buffer, the number of times table t3 is scanned is:

```
(S * C)/join_buffer_size + 1
```

The number of t3 scans decreases as the value of join_buffer_size increases, up to the point when join_buffer_size is large enough to hold all previous row combinations. At that point, there is no speed to be gained by making it larger.

8.2.1.11 Nested Join Optimization

The syntax for expressing joins permits nested joins. The following discussion refers to the join syntax described in Section 13.2.9.2, "JOIN Syntax".

The syntax of *table_factor* is extended in comparison with the SQL Standard. The latter accepts only *table_reference*, not a list of them inside a pair of parentheses. This is a conservative extension if we consider each comma in a list of *table_reference* items as equivalent to an inner join. For example:

```
SELECT * FROM t1 LEFT JOIN (t2, t3, t4)
                 ON (t2.a=t1.a AND t3.b=t1.b AND t4.c=t1.c)
```

is equivalent to:

```
SELECT * FROM t1 LEFT JOIN (t2 CROSS JOIN t3 CROSS JOIN t4)
                 ON (t2.a=t1.a AND t3.b=t1.b AND t4.c=t1.c)
```

In MySQL, CROSS JOIN is a syntactic equivalent to INNER JOIN (they can replace each other). In standard SQL, they are not equivalent. INNER JOIN is used with an ON clause; CROSS JOIN is used otherwise.

In general, parentheses can be ignored in join expressions containing only inner join operations. After removing parentheses and grouping operations to the left, the join expression:

```
t1 LEFT JOIN (t2 LEFT JOIN t3 ON t2.b=t3.b OR t2.b IS NULL)
   ON t1.a=t2.a
```

transforms into the expression:

```
(t1 LEFT JOIN t2 ON t1.a=t2.a) LEFT JOIN t3
    ON t2.b=t3.b OR t2.b IS NULL
```

Yet, the two expressions are not equivalent. To see this, suppose that the tables t1, t2, and t3 have the following state:

- Table t1 contains rows (1), (2)
- Table t2 contains row (1,101)
- Table t3 contains row (101)

In this case, the first expression returns a result set including the rows (1,1,101,101), (2,NULL,NULL,NULL), whereas the second expression returns the rows (1,1,101,101), (2,NULL,NULL,101):

```
mysql> SELECT *
    -> FROM t1
    ->      LEFT JOIN
    ->      (t2 LEFT JOIN t3 ON t2.b=t3.b OR t2.b IS NULL)
    ->      ON t1.a=t2.a;
+------+------+------+------+
| a    | a    | b    | b    |
+------+------+------+------+
|    1 |    1 |  101 |  101 |
|    2 | NULL | NULL | NULL |
+------+------+------+------+

mysql> SELECT *
    -> FROM (t1 LEFT JOIN t2 ON t1.a=t2.a)
    ->      LEFT JOIN t3
    ->      ON t2.b=t3.b OR t2.b IS NULL;
```

```
+------+------+------+------+
| a    | a    | b    | b    |
+------+------+------+------+
|    1 |    1 |  101 |  101 |
|    2 | NULL | NULL |  101 |
+------+------+------+------+
```

In the following example, an outer join operation is used together with an inner join operation:

```
t1 LEFT JOIN (t2, t3) ON t1.a=t2.a
```

That expression cannot be transformed into the following expression:

```
t1 LEFT JOIN t2 ON t1.a=t2.a, t3.
```

For the given table states, the two expressions return different sets of rows:

```
mysql> SELECT *
    -> FROM t1 LEFT JOIN (t2, t3) ON t1.a=t2.a;
+------+------+------+------+
| a    | a    | b    | b    |
+------+------+------+------+
|    1 |    1 |  101 |  101 |
|    2 | NULL | NULL | NULL |
+------+------+------+------+

mysql> SELECT *
    -> FROM t1 LEFT JOIN t2 ON t1.a=t2.a, t3;
+------+------+------+------+
| a    | a    | b    | b    |
+------+------+------+------+
|    1 |    1 |  101 |  101 |
|    2 | NULL | NULL |  101 |
+------+------+------+------+
```

Therefore, if we omit parentheses in a join expression with outer join operators, we might change the result set for the original expression.

More exactly, we cannot ignore parentheses in the right operand of the left outer join operation and in the left operand of a right join operation. In other words, we cannot ignore parentheses for the inner table expressions of outer join operations. Parentheses for the other operand (operand for the outer table) can be ignored.

The following expression:

```
(t1,t2) LEFT JOIN t3 ON P(t2.b,t3.b)
```

is equivalent to this expression:

```
t1, t2 LEFT JOIN t3 ON P(t2.b,t3.b)
```

for any tables `t1,t2,t3` and any condition `P` over attributes `t2.b` and `t3.b`.

Whenever the order of execution of the join operations in a join expression (*`join_table`*) is not from left to right, we talk about nested joins. Consider the following queries:

```
SELECT * FROM t1 LEFT JOIN (t2 LEFT JOIN t3 ON t2.b=t3.b) ON t1.a=t2.a
  WHERE t1.a > 1

SELECT * FROM t1 LEFT JOIN (t2, t3) ON t1.a=t2.a
  WHERE (t2.b=t3.b OR t2.b IS NULL) AND t1.a > 1
```

Those queries are considered to contain these nested joins:

```
t2 LEFT JOIN t3 ON t2.b=t3.b
t2, t3
```

The nested join is formed in the first query with a left join operation, whereas in the second query it is formed with an inner join operation.

In the first query, the parentheses can be omitted: The grammatical structure of the join expression will dictate the same order of execution for join operations. For the second query, the parentheses cannot be omitted, although the join expression here can be interpreted unambiguously without them. (In our extended syntax the parentheses in `(t2, t3)` of the second query are required, although theoretically the query could be parsed without them: We still would have unambiguous syntactical structure for the query because `LEFT JOIN` and `ON` would play the role of the left and right delimiters for the expression `(t2,t3)`.)

The preceding examples demonstrate these points:

- For join expressions involving only inner joins (and not outer joins), parentheses can be removed. You can remove parentheses and evaluate left to right (or, in fact, you can evaluate the tables in any order).
- The same is not true, in general, for outer joins or for outer joins mixed with inner joins. Removal of parentheses may change the result.

Queries with nested outer joins are executed in the same pipeline manner as queries with inner joins. More exactly, a variation of the nested-loop join algorithm is exploited. Recall by what algorithmic schema the nested-loop join executes a query. Suppose that we have a join query over 3 tables `T1,T2,T3` of the form:

```
SELECT * FROM T1 INNER JOIN T2 ON P1(T1,T2)
                 INNER JOIN T3 ON P2(T2,T3)
  WHERE P(T1,T2,T3).
```

Here, `P1(T1,T2)` and `P2(T3,T3)` are some join conditions (on expressions), whereas `P(T1,T2,T3)` is a condition over columns of tables `T1,T2,T3`.

The nested-loop join algorithm would execute this query in the following manner:

```
FOR each row t1 in T1 {
  FOR each row t2 in T2 such that P1(t1,t2) {
    FOR each row t3 in T3 such that P2(t2,t3) {
      IF P(t1,t2,t3) {
         t:=t1||t2||t3; OUTPUT t;
      }
    }
  }
}
```

The notation `t1||t2||t3` means “a row constructed by concatenating the columns of rows `t1`, `t2`, and `t3`.” In some of the following examples, `NULL` where a row name appears means that `NULL` is used for each column of that row. For example, t1||t2||NULL means “a row constructed by concatenating the columns of rows t1 and t2, and NULL for each column of t3.”

Now let's consider a query with nested outer joins:

```
SELECT * FROM T1 LEFT JOIN
              (T2 LEFT JOIN T3 ON P2(T2,T3))
              ON P1(T1,T2)
  WHERE P(T1,T2,T3).
```

For this query, we modify the nested-loop pattern to get:

```
FOR each row t1 in T1 {
  BOOL f1:=FALSE;
  FOR each row t2 in T2 such that P1(t1,t2) {
    BOOL f2:=FALSE;
    FOR each row t3 in T3 such that P2(t2,t3) {
      IF P(t1,t2,t3) {
        t:=t1||t2||t3; OUTPUT t;
      }
      f2=TRUE;
      f1=TRUE;
    }
    IF (!f2) {
      IF P(t1,t2,NULL) {
        t:=t1||t2||NULL; OUTPUT t;
      }
      f1=TRUE;
    }
  }
  IF (!f1) {
    IF P(t1,NULL,NULL) {
      t:=t1||NULL||NULL; OUTPUT t;
    }
  }
}
```

In general, for any nested loop for the first inner table in an outer join operation, a flag is introduced that is turned off before the loop and is checked after the loop. The flag is turned on when for the current row from the outer table a match from the table representing the inner operand is found. If at the end of the loop cycle the flag is still off, no match has been found for the current row of the outer table. In this case, the row is complemented by NULL values for the columns of the inner tables. The result row is passed to the final check for the output or into the next nested loop, but only if the row satisfies the join condition of all embedded outer joins.

In our example, the outer join table expressed by the following expression is embedded:

```
(T2 LEFT JOIN T3 ON P2(T2,T3))
```

Note that for the query with inner joins, the optimizer could choose a different order of nested loops, such as this one:

```
FOR each row t3 in T3 {
  FOR each row t2 in T2 such that P2(t2,t3) {
    FOR each row t1 in T1 such that P1(t1,t2) {
      IF P(t1,t2,t3) {
         t:=t1||t2||t3; OUTPUT t;
      }
    }
  }
}
```

For the queries with outer joins, the optimizer can choose only such an order where loops for outer tables precede loops for inner tables. Thus, for our query with outer joins, only one nesting order is possible. For the following query, the optimizer will evaluate two different nestings:

```
SELECT * T1 LEFT JOIN (T2,T3) ON P1(T1,T2) AND P2(T1,T3)
  WHERE P(T1,T2,T3)
```

The nestings are these:

```
FOR each row t1 in T1 {
  BOOL f1:=FALSE;
  FOR each row t2 in T2 such that P1(t1,t2) {
    FOR each row t3 in T3 such that P2(t1,t3) {
      IF P(t1,t2,t3) {
        t:=t1||t2||t3; OUTPUT t;
      }
      f1:=TRUE
    }
  }
  IF (!f1) {
    IF P(t1,NULL,NULL) {
      t:=t1||NULL||NULL; OUTPUT t;
    }
  }
}
```

and:

```
FOR each row t1 in T1 {
  BOOL f1:=FALSE;
  FOR each row t3 in T3 such that P2(t1,t3) {
    FOR each row t2 in T2 such that P1(t1,t2) {
      IF P(t1,t2,t3) {
        t:=t1||t2||t3; OUTPUT t;
      }
      f1:=TRUE
    }
  }
  IF (!f1) {
    IF P(t1,NULL,NULL) {
      t:=t1||NULL||NULL; OUTPUT t;
    }
  }
}
```

In both nestings, T1 must be processed in the outer loop because it is used in an outer join. T2 and T3 are used in an inner join, so that join must be processed in the inner loop. However, because the join is an inner join, T2 and T3 can be processed in either order.

When discussing the nested-loop algorithm for inner joins, we omitted some details whose impact on the performance of query execution may be huge. We did not mention so-called “pushed-down” conditions. Suppose that our WHERE condition P(T1,T2,T3) can be represented by a conjunctive formula:

```
P(T1,T2,T2) = C1(T1) AND C2(T2) AND C3(T3).
```

In this case, MySQL actually uses the following nested-loop schema for the execution of the query with inner joins:

```
FOR each row t1 in T1 such that C1(t1) {
  FOR each row t2 in T2 such that P1(t1,t2) AND C2(t2)  {
    FOR each row t3 in T3 such that P2(t2,t3) AND C3(t3) {
      IF P(t1,t2,t3) {
         t:=t1||t2||t3; OUTPUT t;
      }
    }
  }
}
```

You see that each of the conjuncts C1(T1), C2(T2), C3(T3) are pushed out of the most inner loop to the most outer loop where it can be evaluated. If C1(T1) is a very restrictive condition, this condition pushdown may greatly reduce the number of rows from table T1 passed to the inner loops. As a result, the execution time for the query may improve immensely.

For a query with outer joins, the WHERE condition is to be checked only after it has been found that the current row from the outer table has a match in the inner tables. Thus, the optimization of pushing conditions out of the inner nested loops cannot be applied directly to queries with outer joins. Here we have to introduce conditional pushed-down predicates guarded by the flags that are turned on when a match has been encountered.

For our example with outer joins with:

```
P(T1,T2,T3)=C1(T1) AND C(T2) AND C3(T3)
```

the nested-loop schema using guarded pushed-down conditions looks like this:

```
FOR each row t1 in T1 such that C1(t1) {
  BOOL f1:=FALSE;
  FOR each row t2 in T2
      such that P1(t1,t2) AND (f1?C2(t2):TRUE) {
    BOOL f2:=FALSE;
    FOR each row t3 in T3
        such that P2(t2,t3) AND (f1&&f2?C3(t3):TRUE) {
      IF (f1&&f2?TRUE:(C2(t2) AND C3(t3))) {
        t:=t1||t2||t3; OUTPUT t;
      }
      f2=TRUE;
      f1=TRUE;
    }
    IF (!f2) {
      IF (f1?TRUE:C2(t2) && P(t1,t2,NULL)) {
        t:=t1||t2||NULL; OUTPUT t;
      }
      f1=TRUE;
    }
  }
  IF (!f1 && P(t1,NULL,NULL)) {
      t:=t1||NULL||NULL; OUTPUT t;
  }
}
```

In general, pushed-down predicates can be extracted from join conditions such as P1(T1,T2) and P(T2,T3). In this case, a pushed-down predicate is guarded also by a flag that prevents checking the predicate for the NULL-complemented row generated by the corresponding outer join operation.

Note that access by key from one inner table to another in the same nested join is prohibited if it is induced by a predicate from the WHERE condition. (We could use conditional key access in this case, but this technique is not employed yet in MySQL 5.7.)

8.2.1.12 Outer Join Simplification

Table expressions in the FROM clause of a query are simplified in many cases.

At the parser stage, queries with right outer joins operations are converted to equivalent queries containing only left join operations. In the general case, the conversion is performed according to the following rule:

```
(T1, ...) RIGHT JOIN (T2,...) ON P(T1,...,T2,...) =
(T2, ...) LEFT JOIN (T1,...) ON P(T1,...,T2,...)
```

All inner join expressions of the form T1 INNER JOIN T2 ON P(T1,T2) are replaced by the list T1,T2, P(T1,T2) being joined as a conjunct to the WHERE condition (or to the join condition of the embedding join, if there is any).

When the optimizer evaluates plans for join queries with outer join operation, it takes into consideration only the plans where, for each such operation, the outer tables are accessed before the inner tables. The optimizer options are limited because only such plans enables us to execute queries with outer joins operations by the nested loop schema.

Suppose that we have a query of the form:

```
SELECT * T1 LEFT JOIN T2 ON P1(T1,T2)
  WHERE P(T1,T2) AND R(T2)
```

with R(T2) narrowing greatly the number of matching rows from table T2. If we executed the query as it is, the optimizer would have no other choice besides to access table T1 before table T2 that may lead to a very inefficient execution plan.

Fortunately, MySQL converts such a query into a query without an outer join operation if the WHERE condition is null-rejected. A condition is called null-rejected for an outer join operation if it evaluates to FALSE or to UNKNOWN for any NULL-complemented row built for the operation.

Thus, for this outer join:

```
T1 LEFT JOIN T2 ON T1.A=T2.A
```

Conditions such as these are null-rejected:

```
T2.B IS NOT NULL,
T2.B > 3,
T2.C <= T1.C,
T2.B < 2 OR T2.C > 1
```

Conditions such as these are not null-rejected:

```
T2.B IS NULL,
T1.B < 3 OR T2.B IS NOT NULL,
T1.B < 3 OR T2.B > 3
```

The general rules for checking whether a condition is null-rejected for an outer join operation are simple. A condition is null-rejected in the following cases:

- If it is of the form A IS NOT NULL, where A is an attribute of any of the inner tables
- If it is a predicate containing a reference to an inner table that evaluates to UNKNOWN when one of its arguments is NULL

- If it is a conjunction containing a null-rejected condition as a conjunct
- If it is a disjunction of null-rejected conditions

A condition can be null-rejected for one outer join operation in a query and not null-rejected for another. In the query:

```
SELECT * FROM T1 LEFT JOIN T2 ON T2.A=T1.A
                 LEFT JOIN T3 ON T3.B=T1.B
  WHERE T3.C > 0
```

the WHERE condition is null-rejected for the second outer join operation but is not null-rejected for the first one.

If the WHERE condition is null-rejected for an outer join operation in a query, the outer join operation is replaced by an inner join operation.

For example, the preceding query is replaced with the query:

```
SELECT * FROM T1 LEFT JOIN T2 ON T2.A=T1.A
                 INNER JOIN T3 ON T3.B=T1.B
  WHERE T3.C > 0
```

For the original query, the optimizer would evaluate plans compatible with only one access order T1,T2,T3. For the replacing query, it additionally considers the access sequence T3,T1,T2.

A conversion of one outer join operation may trigger a conversion of another. Thus, the query:

```
SELECT * FROM T1 LEFT JOIN T2 ON T2.A=T1.A
                 LEFT JOIN T3 ON T3.B=T2.B
  WHERE T3.C > 0
```

will be first converted to the query:

```
SELECT * FROM T1 LEFT JOIN T2 ON T2.A=T1.A
                 INNER JOIN T3 ON T3.B=T2.B
  WHERE T3.C > 0
```

which is equivalent to the query:

```
SELECT * FROM (T1 LEFT JOIN T2 ON T2.A=T1.A), T3
  WHERE T3.C > 0 AND T3.B=T2.B
```

Now the remaining outer join operation can be replaced by an inner join, too, because the condition T3.B=T2.B is null-rejected and we get a query without outer joins at all:

```
SELECT * FROM (T1 INNER JOIN T2 ON T2.A=T1.A), T3
  WHERE T3.C > 0 AND T3.B=T2.B
```

Sometimes we succeed in replacing an embedded outer join operation, but cannot convert the embedding outer join. The following query:

```
SELECT * FROM T1 LEFT JOIN
              (T2 LEFT JOIN T3 ON T3.B=T2.B)
              ON T2.A=T1.A
```

```
  WHERE T3.C > 0
```

is converted to:

```
SELECT * FROM T1 LEFT JOIN
              (T2 INNER JOIN T3 ON T3.B=T2.B)
              ON T2.A=T1.A
  WHERE T3.C > 0,
```

That can be rewritten only to the form still containing the embedding outer join operation:

```
SELECT * FROM T1 LEFT JOIN
              (T2,T3)
              ON (T2.A=T1.A AND T3.B=T2.B)
  WHERE T3.C > 0.
```

When trying to convert an embedded outer join operation in a query, we must take into account the join condition for the embedding outer join together with the WHERE condition. In the query:

```
SELECT * FROM T1 LEFT JOIN
              (T2 LEFT JOIN T3 ON T3.B=T2.B)
              ON T2.A=T1.A AND T3.C=T1.C
  WHERE T3.D > 0 OR T1.D > 0
```

the WHERE condition is not null-rejected for the embedded outer join, but the join condition of the embedding outer join T2.A=T1.A AND T3.C=T1.C is null-rejected. So the query can be converted to:

```
SELECT * FROM T1 LEFT JOIN
              (T2, T3)
              ON T2.A=T1.A AND T3.C=T1.C AND T3.B=T2.B
  WHERE T3.D > 0 OR T1.D > 0
```

8.2.1.13 Multi-Range Read Optimization

Reading rows using a range scan on a secondary index can result in many random disk accesses to the base table when the table is large and not stored in the storage engine's cache. With the Disk-Sweep Multi-Range Read (MRR) optimization, MySQL tries to reduce the number of random disk access for range scans by first scanning the index only and collecting the keys for the relevant rows. Then the keys are sorted and finally the rows are retrieved from the base table using the order of the primary key. The motivation for Disk-sweep MRR is to reduce the number of random disk accesses and instead achieve a more sequential scan of the base table data.

The Multi-Range Read optimization provides these benefits:

- MRR enables data rows to be accessed sequentially rather than in random order, based on index tuples. The server obtains a set of index tuples that satisfy the query conditions, sorts them according to data row ID order, and uses the sorted tuples to retrieve data rows in order. This makes data access more efficient and less expensive.

- MRR enables batch processing of requests for key access for operations that require access to data rows through index tuples, such as range index scans and equi-joins that use an index for the join attribute. MRR iterates over a sequence of index ranges to obtain qualifying index tuples. As these results accumulate, they are used to access the corresponding data rows. It is not necessary to acquire all index tuples before starting to read data rows.

The following scenarios illustrate when MRR optimization can be advantageous:

Scenario A: MRR can be used for InnoDB and MyISAM tables for index range scans and equi-join operations.

1. A portion of the index tuples are accumulated in a buffer.
2. The tuples in the buffer are sorted by their data row ID.
3. Data rows are accessed according to the sorted index tuple sequence.

Scenario B: MRR can be used for NDB tables for multiple-range index scans or when performing an equi-join by an attribute.

1. A portion of ranges, possibly single-key ranges, is accumulated in a buffer on the central node where the query is submitted.
2. The ranges are sent to the execution nodes that access data rows.
3. The accessed rows are packed into packages and sent back to the central node.
4. The received packages with data rows are placed in a buffer.
5. Data rows are read from the buffer.

When MRR is used, the Extra column in EXPLAIN output shows Using MRR.

InnoDB and MyISAM do not use MRR if full table rows need not be accessed to produce the query result. This is the case if results can be produced entirely on the basis on information in the index tuples (through a covering index); MRR provides no benefit.

Example query for which MRR can be used, assuming that there is an index on (*key_part1*, *key_part2*):

```
SELECT * FROM t
  WHERE key_part1 >= 1000 AND key_part1 < 2000
  AND key_part2 = 10000;
```

The index consists of tuples of (*key_part1*, *key_part2*) values, ordered first by *key_part1* and then by *key_part2*.

Without MRR, an index scan covers all index tuples for the *key_part1* range from 1000 up to 2000, regardless of the *key_part2* value in these tuples. The scan does extra work to the extent that tuples in the range contain *key_part2* values other than 10000.

With MRR, the scan is broken up into multiple ranges, each for a single value of *key_part1* (1000, 1001, ... , 1999). Each of these scans need look only for tuples with *key_part2* = 10000. If the index contains many tuples for which *key_part2* is not 10000, MRR results in many fewer index tuples being read.

To express this using interval notation, the non-MRR scan must examine the index range [{1000, 10000}, {2000, MIN_INT}), which may include many tuples other than those for which *key_part2* = 10000. The MRR scan examines multiple single-point intervals [{1000, 10000}], ..., [{1999, 10000}], which includes only tuples with *key_part2* = 10000.

Two optimizer_switch system variable flags provide an interface to the use of MRR optimization. The mrr flag controls whether MRR is enabled. If mrr is enabled (on), the mrr_cost_based flag controls whether the optimizer attempts to make a cost-based choice between using and not using MRR (on) or uses MRR whenever possible (off). By default, mrr is on and mrr_cost_based is on. See Section 8.8.6.2, “Controlling Switchable Optimizations”.

For MRR, a storage engine uses the value of the read_rnd_buffer_size system variable as a guideline for how much memory it can allocate for its buffer. The engine uses up to read_rnd_buffer_size bytes and determines the number of ranges to process in a single pass.

8.2.1.14 Block Nested-Loop and Batched Key Access Joins

In MySQL 5.7, a Batched Key Access (BKA) Join algorithm is available that uses both index access to the joined table and a join buffer. The BKA algorithm supports inner join, outer join, and semi-join operations, including nested outer joins. Benefits of BKA include improved join performance due to more efficient table scanning. Also, the Block Nested-Loop (BNL) Join algorithm previously used only for inner joins is extended and can be employed for outer join and semi-join operations, including nested outer joins.

The following sections discuss the join buffer management that underlies the extension of the original BNL algorithm, the extended BNL algorithm, and the BKA algorithm. For information about semi-join strategies, see Optimizing Subqueries with Semi-Join Transformations

Join Buffer Management for Block Nested-Loop and Batched Key Access Algorithms

In MySQL 5.7, MySQL Server can employ join buffers to execute not only inner joins without index access to the inner table, but also outer joins and semi-joins that appear after subquery flattening. Moreover, a join buffer can be effectively used when there is an index access to the inner table.

The join buffer management code slightly more efficiently utilizes join buffer space when storing the values of the interesting row columns: No additional bytes are allocated in buffers for a row column if its value is NULL, and the minimum number of bytes is allocated for any value of the VARCHAR type.

The code supports two types of buffers, regular and incremental. Suppose that join buffer B1 is employed to join tables t1 and t2 and the result of this operation is joined with table t3 using join buffer B2:

- A regular join buffer contains columns from each join operand. If B2 is a regular join buffer, each row *r* put into B2 is composed of the columns of a row *r1* from B1 and the interesting columns of a matching row *r2* from table t2.

- An incremental join buffer contains only columns from rows of the table produced by the second join operand. That is, it is incremental to a row from the first operand buffer. If B2 is an incremental join buffer, it contains the interesting columns of the row *r2* together with a link to the row *r1* from B1.

Incremental join buffers are always incremental relative to a join buffer from an earlier join operation, so the buffer from the first join operation is always a regular buffer. In the example just given, the buffer B1 used to join tables t1 and t2 must be a regular buffer.

Each row of the incremental buffer used for a join operation contains only the interesting columns of a row from the table to be joined. These columns are augmented with a reference to the interesting columns of the matched row from the table produced by the first join operand. Several rows in the incremental buffer can refer to the same row *r* whose columns are stored in the previous join buffers insofar as all these rows match row *r*.

Incremental buffers enable less frequent copying of columns from buffers used for previous join operations. This provides a savings in buffer space because in the general case a row produced by the first join operand can be matched by several rows produced by the second join operand. It is unnecessary to make several copies of a row from the first operand. Incremental buffers also provide a savings in processing time due to the reduction in copying time.

The block_nested_loop and batched_key_access flags of the optimizer_switch system variable control how the optimizer uses the Block Nested-Loop and Batched Key Access join algorithms. By default, block_nested_loop is on and batched_key_access is off. See Section 8.8.6.2, "Controlling Switchable Optimizations".

Before MySQL 5.6.3, the optimizer_join_cache_level system variable controls join buffer management. For the possible values of this variable and their meanings, see the description in Section 5.1.4, "Server System Variables".

For information about semi-join strategies, see Optimizing Subqueries with Semi-Join Transformations

Block Nested-Loop Algorithm for Outer Joins and Semi-Joins

In MySQL 5.7, the original implementation of the BNL algorithm is extended to support outer join and semi-join operations.

When these operations are executed with a join buffer, each row put into the buffer is supplied with a match flag.

If an outer join operation is executed using a join buffer, each row of the table produced by the second operand is checked for a match against each row in the join buffer. When a match is found, a new extended row is formed (the original row plus columns from the second operand) and sent for further extensions by the remaining join operations. In addition, the match flag of the matched row in the buffer is enabled. After all rows of the table to be joined have been examined, the join buffer is scanned. Each row from the buffer that does not have its match flag enabled is extended by NULL complements (NULL values for each column in the second operand) and sent for further extensions by the remaining join operations.

As of MySQL 5.6.3, the block_nested_loop flag of the optimizer_switch system variable controls how the optimizer uses the Block Nested-Loop algorithm. By default, block_nested_loop is on. See Section 8.8.6.2, "Controlling Switchable Optimizations".

Before MySQL 5.6.3, the optimizer_join_cache_level system variable controls join buffer management. For the possible values of this variable and their meanings, see the description in Section 5.1.4, "Server System Variables".

In EXPLAIN output, use of BNL for a table is signified when the Extra value contains Using join buffer (Block Nested Loop) and the type value is ALL, index, or range.

For information about semi-join strategies, see Optimizing Subqueries with Semi-Join Transformations

Batched Key Access Joins

MySQL Server implements a method of joining tables called the Batched Key Access (BKA) join algorithm. BKA can be applied when there is an index access to the table produced by the second join operand. Like the BNL join algorithm, the BKA join algorithm employs a join buffer to accumulate the interesting columns of the rows produced by the first operand of the join operation. Then the BKA algorithm builds keys to access the table to be joined for all rows in the buffer and submits these keys in a batch to the database engine for index lookups. The keys are submitted to the engine through the Multi-Range Read (MRR) interface (see Section 8.2.1.13, "Multi-Range Read Optimization"). After submission of the keys, the MRR engine functions perform lookups in the index in an optimal way, fetching the rows of the joined table found by these keys, and starts feeding the BKA join algorithm with matching rows. Each matching row is coupled with a reference to a row in the join buffer.

When BKA is used, the value of join_buffer_size defines how large the batch of keys is in each request to the storage engine. The larger the buffer, the more sequential access will be to the right hand table of a join operation, which can significantly improve performance.

For BKA to be used, the batched_key_access flag of the optimizer_switch system variable must be set to on. BKA uses MRR, so the mrr flag must also be on. Currently, the cost estimation for MRR is too pessimistic. Hence, it is also necessary for mrr_cost_based to be off for BKA to be used. The following setting enables BKA:

```
mysql> SET optimizer_switch='mrr=on,mrr_cost_based=off,batched_key_access=on';
```

There are two scenarios by which MRR functions execute:

- The first scenario is used for conventional disk-based storage engines such as InnoDB and MyISAM. For these engines, usually the keys for all rows from the join buffer are submitted to the MRR interface at once. Engine-specific MRR functions perform index lookups for the submitted keys, get row IDs (or primary keys) from them, and then fetch rows for all these selected row IDs one by one by request from BKA algorithm. Every row is returned with an association reference that enables access to the matched row in the join buffer. The rows are fetched by the MRR functions in an optimal way: They are fetched in the row ID (primary key) order. This improves performance because reads are in disk order rather than random order.

- The second scenario is used for remote storage engines such as NDB. A package of keys for a portion of rows from the join buffer, together with their associations, is sent by MySQL Server to NDB nodes. In return, the Server receives a package (or several packages) of matching rows coupled with corresponding associations. The BKA join algorithm takes these rows and builds new joined rows. Then a new portion of keys are sent to the data nodes, and the rows from the returned packages are used to build new joined rows. The process continues until the last keys from the join buffer are sent to the data nodes, and the MySQL server receives and joins all rows matching these keys. This improves performance because the fewer packages with keys the MySQL Server sends to the Cluster, the fewer round trips between the server and the Cluster nodes are required to perform the join operation.

With the first scenario, a portion of the join buffer is reserved to store row IDs (primary keys) selected by index lookups and passed as a parameter to the MRR functions.

There is no special buffer to store keys built for rows from the join buffer. Instead, a function that builds the key for the next row in the buffer is passed as a parameter to the MRR functions.

In EXPLAIN output, use of BKA for a table is signified when the Extra value contains Using join buffer (Batched Key Access) and the type value is ref or eq_ref.

8.2.1.15 ORDER BY Optimization

In some cases, MySQL can use an index to satisfy an ORDER BY clause without doing any extra sorting.

The index can also be used even if the ORDER BY does not match the index exactly, as long as all of the unused portions of the index and all the extra ORDER BY columns are constants in the WHERE clause. The following queries use the index to resolve the ORDER BY part:

```
SELECT * FROM t1
  ORDER BY key_part1,key_part2,... ;

SELECT * FROM t1
  WHERE key_part1 = constant
  ORDER BY key_part2;

SELECT * FROM t1
  ORDER BY key_part1 DESC, key_part2 DESC;

SELECT * FROM t1
```

```
  WHERE key_part1 = 1
  ORDER BY key_part1 DESC, key_part2 DESC;

SELECT * FROM t1
  WHERE key_part1 > constant
  ORDER BY key_part1 ASC;

SELECT * FROM t1
  WHERE key_part1 < constant
  ORDER BY key_part1 DESC;

SELECT * FROM t1
  WHERE key_part1 = constant1 AND key_part2 > constant2
  ORDER BY key_part2;
```

In some cases, MySQL *cannot* use indexes to resolve the ORDER BY, although it still uses indexes to find the rows that match the WHERE clause. These cases include the following:

- You use ORDER BY on different keys:

  ```
  SELECT * FROM t1 ORDER BY key1, key2;
  ```

- You use ORDER BY on nonconsecutive parts of a key:

  ```
  SELECT * FROM t1 WHERE key2=constant ORDER BY key_part2;
  ```

- You mix ASC and DESC:

  ```
  SELECT * FROM t1 ORDER BY key_part1 DESC, key_part2 ASC;
  ```

- The key used to fetch the rows is not the same as the one used in the ORDER BY:

  ```
  SELECT * FROM t1 WHERE key2=constant ORDER BY key1;
  ```

- You use ORDER BY with an expression that includes terms other than the key column name:

  ```
  SELECT * FROM t1 ORDER BY ABS(key);
  SELECT * FROM t1 ORDER BY -key;
  ```

- You are joining many tables, and the columns in the ORDER BY are not all from the first nonconstant table that is used to retrieve rows. (This is the first table in the EXPLAIN output that does not have a const join type.)

- You have different ORDER BY and GROUP BY expressions.

- You index only a prefix of a column named in the ORDER BY clause. In this case, the index cannot be used to fully resolve the sort order. For example, if you have a CHAR(20) column, but index only the first 10 bytes, the index cannot distinguish values past the 10th byte and a filesort will be needed.

- The type of table index used does not store rows in order. For example, this is true for a HASH index in a MEMORY table.

Availability of an index for sorting may be affected by the use of column aliases. Suppose that the column t1.a is indexed. In this statement, the name of the column in the select list is a. It refers to t1.a, so for the reference to a in the ORDER BY, the index can be used:

```
SELECT a FROM t1 ORDER BY a;
```

In this statement, the name of the column in the select list is also a, but it is the alias name. It refers to ABS(a), so for the reference to a in the ORDER BY, the index cannot be used:

```
SELECT ABS(a) AS a FROM t1 ORDER BY a;
```

In the following statement, the ORDER BY refers to a name that is not the name of a column in the select list. But there is a column in t1 named a, so the ORDER BY uses that, and the index can be used. (The resulting sort order may be completely different from the order for ABS(a), of course.)

```
SELECT ABS(a) AS b FROM t1 ORDER BY a;
```

By default, MySQL sorts all GROUP BY *col1*, *col2*, ... queries as if you specified ORDER BY *col1*, *col2*, ... in the query as well. If you include an explicit ORDER BY clause that contains the same column list, MySQL optimizes it away without any speed penalty, although the sorting still occurs. If a query includes GROUP BY but you want to avoid the overhead of sorting the result, you can suppress sorting by specifying ORDER BY NULL. For example:

```
INSERT INTO foo
SELECT a, COUNT(*) FROM bar GROUP BY a ORDER BY NULL;
```

Note

Relying on implicit GROUP BY sorting in MySQL 5.7 is deprecated. To achieve a specific sort order of grouped results, it is preferable to use an explicit ORDER BY clause. GROUP BY sorting is a MySQL extension that may change in a future release; for example, to make it possible for the optimizer to order groupings in whatever manner it deems most efficient and to avoid the sorting overhead.

With EXPLAIN SELECT ... ORDER BY, you can check whether MySQL can use indexes to resolve the query. It cannot if you see Using filesort in the Extra column. See Section 8.8.1, "Optimizing Queries with EXPLAIN". Filesort uses a fixed-length row-storage format similar to that used by the MEMORY storage engine. Variable-length types such as VARCHAR are stored using a fixed length.

MySQL has two filesort algorithms for sorting and retrieving results. The original method uses only the ORDER BY columns. The modified method uses not just the ORDER BY columns, but all the columns referenced by the query.

The optimizer selects which filesort algorithm to use. It normally uses the modified algorithm except when BLOB or TEXT columns are involved, in which case it uses the original algorithm. For both algorithms, the sort buffer size is the sort_buffer_size system variable value.

The original filesort algorithm works as follows:

1. Read all rows according to key or by table scanning. Skip rows that do not match the WHERE clause.

2. For each row, store a pair of values (the sort key value and the row ID) in the sort buffer.

3. If all pairs fit into the sort buffer, no temporary file is created. Otherwise, when the sort buffer becomes full, run a qsort (quicksort) on it in memory and write it to a temporary file. Save a pointer to the sorted block.

4. Repeat the preceding steps until all rows have been read.

5. Do a multi-merge of up to `MERGEBUFF` (7) regions to one block in another temporary file. Repeat until all blocks from the first file are in the second file.

6. Repeat the following until there are fewer than `MERGEBUFF2` (15) blocks left.

7. On the last multi-merge, only the row ID (the last part of the value pair) is written to a result file.

8. Read the rows in sorted order using the row IDs in the result file. To optimize this, read in a large block of row IDs, sort them, and use them to read the rows in sorted order into a row buffer. The row buffer size is the `read_rnd_buffer_size` system variable value. The code for this step is in the `sql/records.cc` source file.

One problem with this approach is that it reads rows twice: One time during `WHERE` clause evaluation, and again after sorting the value pairs. And even if the rows were accessed successively the first time (for example, if a table scan is done), the second time they are accessed randomly. (The sort keys are ordered, but the row positions are not.)

The modified `filesort` algorithm incorporates an optimization to avoid reading the rows twice: It records the sort key value, but instead of the row ID, it records the columns referenced by the query. The modified `filesort` algorithm works like this:

1. Read the rows that match the `WHERE` clause.

2. For each row, record a tuple of values consisting of the sort key value and the columns referenced by the query.

3. When the sort buffer becomes full, sort the tuples by sort key value in memory and write it to a temporary file.

4. After merge-sorting the temporary file, retrieve the rows in sorted order, but read the required columns directly from the sorted tuples rather than by accessing the table a second time.

Using the modified `filesort` algorithm, the tuples are longer than the pairs used in the original method, and fewer of them fit in the sort buffer. As a result, it is possible for the extra I/O to make the modified approach slower, not faster. To avoid a slowdown, the optimizer uses the modified algorithm only if the total size of the extra columns in the sort tuple does not exceed the value of the `max_length_for_sort_data` system variable. (A symptom of setting the value of this variable too high is a combination of high disk activity and low CPU activity.)

As of MySQL 5.7.3, the modified `filesort` algorithm includes an additional optimization designed to enable more tuples to fit into the sort buffer: For additional columns of type `CHAR` or `VARCHAR`, or any nullable fixed-size data type, the values are packed. For example, without packing, a `VARCHAR(255)` column value containing only 3 characters takes 255 characters in the sort buffer. With packing, the value requires only 3 characters plus a two-byte length indicator. `NULL` values require only a bitmask.

For data containing packable strings shorter than the maximum column length or many `NULL` values, more records fit into the sort buffer. This improves in-memory sorting of the sort buffer and performance of disk-based temporary file merge sorting.

In edge cases, packing may be disadvantageous: If packable strings are the maximum column length or there are few `NULL` values, the space required for the length indicators reduces the number of records that fit into the sort buffer and sorting is slower in memory and on disk.

If a `filesort` is done, `EXPLAIN` output includes `Using filesort` in the `Extra` column. Also, optimizer trace output includes a `filesort_summary` block. For example:

```
"filesort_summary": {
  "rows": 100,
  "examined_rows": 100,
  "number_of_tmp_files": 0,
  "sort_buffer_size": 25192,
  "sort_mode": "<sort_key, packed_additional_fields>"
}
```

The sort_mode value provides information about the algorithm used and the contents of tuples in the sort buffer:

- <sort_key, rowid>: Sort buffer tuples contain the sort key value and row ID of the original table row. Tuples are sorted by sort key value and the row ID is used to read the row from the table.

- <sort_key, additional_fields>: Sort buffer tuples contain the sort key value and columns referenced by the query. Tuples are sorted by sort key value and column values are read directly from the tuple.

- <sort_key, packed_additional_fields>: Sort buffer tuples contain the sort key value and packed columns referenced by the query. Tuples are sorted by sort key value and column values are read directly from the tuple.

For information about the optimizer trace, see MySQL Internals: Tracing the Optimizer.

Suppose that a table t1 has four VARCHAR columns a, b, c, and d and that the optimizer uses filesort for this query:

```
SELECT * FROM t1 ORDER BY a, b;
```

The query sorts by a and b, but returns all columns, so the columns referenced by the query are a, b, c, and d. Depending on which filesort algorithm the optimizer chooses, the query executes as follows:

For the original algorithm, sort buffer tuples have these contents:

```
(fixed size a value, fixed size b value,
row ID into t1)
```

The optimizer sorts on the fixed size values. After sorting, the optimizer reads the tuples in order and uses the row ID in each tuple to read rows from t1 to obtain the select list column values.

For the modified algorithm without packing, sort buffer tuples have these contents:

```
(fixed size a value, fixed size b value,
a value, b value, c value, d value)
```

The optimizer sorts on the fixed size values. After sorting, the optimizer reads the tuples in order and uses the values for a, b, c, and d to obtain the select list column values without reading t1 again.

For the modified algorithm with packing, sort buffer tuples have these contents:

```
(fixed size a value, fixed size b value,
a length, packed a value, b length, packed b value,
c length, packed c value, d length, packed d value)
```

If any of a, b, c, or d are NULL, they take no space in the sort buffer other than in the bitmask.

The optimizer sorts on the fixed size values. After sorting, the optimizer reads the tuples in order and uses the values for a, b, c, and d to obtain the select list column values without reading t1 again.

For slow queries for which filesort is not used, try lowering max_length_for_sort_data to a value that is appropriate to trigger a filesort.

To increase ORDER BY speed, check whether you can get MySQL to use indexes rather than an extra sorting phase. If this is not possible, you can try the following strategies:

- Increase the sort_buffer_size variable value.

- Increase the read_rnd_buffer_size variable value.

- Use less RAM per row by declaring columns only as large as they need to be to hold the values stored in them. For example, CHAR(16) is better than CHAR(200) if values never exceed 16 characters.

- Change the tmpdir system variable to point to a dedicated file system with large amounts of free space. The variable value can list several paths that are used in round-robin fashion; you can use this feature to spread the load across several directories. Paths should be separated by colon characters (“:”) on Unix and semicolon characters (“;”) on Windows. The paths should name directories in file systems located on different *physical* disks, not different partitions on the same disk.

If an index is not used for ORDER BY but a LIMIT clause is also present, the optimizer may be able to avoid using a merge file and sort the rows in memory. For details, see Section 8.2.1.19, “Optimizing LIMIT Queries”.

8.2.1.16 GROUP BY Optimization

The most general way to satisfy a GROUP BY clause is to scan the whole table and create a new temporary table where all rows from each group are consecutive, and then use this temporary table to discover groups and apply aggregate functions (if any). In some cases, MySQL is able to do much better than that and to avoid creation of temporary tables by using index access.

The most important preconditions for using indexes for GROUP BY are that all GROUP BY columns reference attributes from the same index, and that the index stores its keys in order (for example, this is a BTREE index and not a HASH index). Whether use of temporary tables can be replaced by index access also depends on which parts of an index are used in a query, the conditions specified for these parts, and the selected aggregate functions.

There are two ways to execute a GROUP BY query through index access, as detailed in the following sections. In the first method, the grouping operation is applied together with all range predicates (if any). The second method first performs a range scan, and then groups the resulting tuples.

In MySQL, GROUP BY is used for sorting, so the server may also apply ORDER BY optimizations to grouping. See Section 8.2.1.15, “ORDER BY Optimization”.

Loose Index Scan

The most efficient way to process GROUP BY is when an index is used to directly retrieve the grouping columns. With this access method, MySQL uses the property of some index types that the keys are ordered (for example, BTREE). This property enables use of lookup groups in an index without having to consider all keys in the index that satisfy all WHERE conditions. This access method considers only a fraction of the keys in an index, so it is called a *loose index scan*. When there is no WHERE clause, a loose index scan reads as many keys as the number of groups, which may be a much smaller number than that of all keys. If the WHERE clause contains range predicates (see the discussion of the range join type in

Section 8.8.1, “Optimizing Queries with EXPLAIN”), a loose index scan looks up the first key of each group that satisfies the range conditions, and again reads the least possible number of keys. This is possible under the following conditions:

- The query is over a single table.
- The GROUP BY names only columns that form a leftmost prefix of the index and no other columns. (If, instead of GROUP BY, the query has a DISTINCT clause, all distinct attributes refer to columns that form a leftmost prefix of the index.) For example, if a table t1 has an index on (c1,c2,c3), loose index scan is applicable if the query has GROUP BY c1, c2,. It is not applicable if the query has GROUP BY c2, c3 (the columns are not a leftmost prefix) or GROUP BY c1, c2, c4 (c4 is not in the index).
- The only aggregate functions used in the select list (if any) are MIN() and MAX(), and all of them refer to the same column. The column must be in the index and must follow the columns in the GROUP BY.
- Any other parts of the index than those from the GROUP BY referenced in the query must be constants (that is, they must be referenced in equalities with constants), except for the argument of MIN() or MAX() functions.
- For columns in the index, full column values must be indexed, not just a prefix. For example, with c1 VARCHAR(20), INDEX (c1(10)), the index cannot be used for loose index scan.

If loose index scan is applicable to a query, the EXPLAIN output shows Using index for group-by in the Extra column.

Assume that there is an index idx(c1,c2,c3) on table t1(c1,c2,c3,c4). The loose index scan access method can be used for the following queries:

```
SELECT c1, c2 FROM t1 GROUP BY c1, c2;
SELECT DISTINCT c1, c2 FROM t1;
SELECT c1, MIN(c2) FROM t1 GROUP BY c1;
SELECT c1, c2 FROM t1 WHERE c1 < const GROUP BY c1, c2;
SELECT MAX(c3), MIN(c3), c1, c2 FROM t1 WHERE c2 > const GROUP BY c1, c2;
SELECT c2 FROM t1 WHERE c1 < const GROUP BY c1, c2;
SELECT c1, c2 FROM t1 WHERE c3 = const GROUP BY c1, c2;
```

The following queries cannot be executed with this quick select method, for the reasons given:

- There are aggregate functions other than MIN() or MAX():

  ```
  SELECT c1, SUM(c2) FROM t1 GROUP BY c1;
  ```

- The columns in the GROUP BY clause do not form a leftmost prefix of the index:

  ```
  SELECT c1, c2 FROM t1 GROUP BY c2, c3;
  ```

- The query refers to a part of a key that comes after the GROUP BY part, and for which there is no equality with a constant:

  ```
  SELECT c1, c3 FROM t1 GROUP BY c1, c2;
  ```

  Were the query to include WHERE c3 = *const*, loose index scan could be used.

The loose index scan access method can be applied to other forms of aggregate function references in the select list, in addition to the MIN() and MAX() references already supported:

- AVG(DISTINCT), SUM(DISTINCT), and COUNT(DISTINCT) are supported. AVG(DISTINCT) and SUM(DISTINCT) take a single argument. COUNT(DISTINCT) can have more than one column argument.
- There must be no GROUP BY or DISTINCT clause in the query.
- The loose scan limitations described earlier still apply.

Assume that there is an index idx(c1,c2,c3) on table t1(c1,c2,c3,c4). The loose index scan access method can be used for the following queries:

```
SELECT COUNT(DISTINCT c1), SUM(DISTINCT c1) FROM t1;

SELECT COUNT(DISTINCT c1, c2), COUNT(DISTINCT c2, c1) FROM t1;
```

Loose index scan is not applicable for the following queries:

```
SELECT DISTINCT COUNT(DISTINCT c1) FROM t1;

SELECT COUNT(DISTINCT c1) FROM t1 GROUP BY c1;
```

Tight Index Scan

A tight index scan may be either a full index scan or a range index scan, depending on the query conditions.

When the conditions for a loose index scan are not met, it still may be possible to avoid creation of temporary tables for GROUP BY queries. If there are range conditions in the WHERE clause, this method reads only the keys that satisfy these conditions. Otherwise, it performs an index scan. Because this method reads all keys in each range defined by the WHERE clause, or scans the whole index if there are no range conditions, we term it a *tight index scan*. With a tight index scan, the grouping operation is performed only after all keys that satisfy the range conditions have been found.

For this method to work, it is sufficient that there is a constant equality condition for all columns in a query referring to parts of the key coming before or in between parts of the GROUP BY key. The constants from the equality conditions fill in any “gaps” in the search keys so that it is possible to form complete prefixes of the index. These index prefixes then can be used for index lookups. If we require sorting of the GROUP BY result, and it is possible to form search keys that are prefixes of the index, MySQL also avoids extra sorting operations because searching with prefixes in an ordered index already retrieves all the keys in order.

Assume that there is an index idx(c1,c2,c3) on table t1(c1,c2,c3,c4). The following queries do not work with the loose index scan access method described earlier, but still work with the tight index scan access method.

- There is a gap in the GROUP BY, but it is covered by the condition c2 = 'a':

  ```
  SELECT c1, c2, c3 FROM t1 WHERE c2 = 'a' GROUP BY c1, c3;
  ```

- The GROUP BY does not begin with the first part of the key, but there is a condition that provides a constant for that part:

  ```
  SELECT c1, c2, c3 FROM t1 WHERE c1 = 'a' GROUP BY c2, c3;
  ```

8.2.1.17 DISTINCT Optimization

DISTINCT combined with ORDER BY needs a temporary table in many cases.

Because DISTINCT may use GROUP BY, learn how MySQL works with columns in ORDER BY or HAVING clauses that are not part of the selected columns. See Section 12.17.3, “MySQL Extensions to GROUP BY”.

In most cases, a DISTINCT clause can be considered as a special case of GROUP BY. For example, the following two queries are equivalent:

```
SELECT DISTINCT c1, c2, c3 FROM t1
WHERE c1 > const;

SELECT c1, c2, c3 FROM t1
WHERE c1 > const GROUP BY c1, c2, c3;
```

Due to this equivalence, the optimizations applicable to GROUP BY queries can be also applied to queries with a DISTINCT clause. Thus, for more details on the optimization possibilities for DISTINCT queries, see Section 8.2.1.16, “GROUP BY Optimization”.

When combining LIMIT *row_count* with DISTINCT, MySQL stops as soon as it finds *row_count* unique rows.

If you do not use columns from all tables named in a query, MySQL stops scanning any unused tables as soon as it finds the first match. In the following case, assuming that t1 is used before t2 (which you can check with EXPLAIN), MySQL stops reading from t2 (for any particular row in t1) when it finds the first row in t2:

```
SELECT DISTINCT t1.a FROM t1, t2 where t1.a=t2.a;
```

8.2.1.18 Subquery Optimization

The MySQL query optimizer has different strategies available to evaluate subqueries. For IN (or =ANY) subqueries, the optimizer has these choices:

- Semi-join
- Materialization
- EXISTS strategy

For NOT IN (or <>ALL) subqueries, the optimizer has these choices:

- Materialization
- EXISTS strategy

The following sections provide more information about these optimization strategies.

Optimizing Subqueries with Semi-Join Transformations

The optimizer uses semi-join strategies to improve subquery execution, as described in this section.

For an inner join between two tables, the join returns a row from one table as many times as there are matches in the other table. But for some questions, the only information that matters is whether there is a match, not the number of matches. Suppose that there are tables named class and roster that list classes in a course curriculum and class rosters (students enrolled in each class), respectively. To list the classes that actually have students enrolled, you could use this join:

```
SELECT class.class_num, class.class_name
```

```
FROM class INNER JOIN roster
WHERE class.class_num = roster.class_num;
```

However, the result lists each class once for each enrolled student. For the question being asked, this is unnecessary duplication of information.

Assuming that `class_num` is a primary key in the `class` table, duplicate suppression could be achieved by using `SELECT DISTINCT`, but it is inefficient to generate all matching rows first only to eliminate duplicates later.

The same duplicate-free result can be obtained by using a subquery:

```
SELECT class_num, class_name
FROM class
WHERE class_num IN (SELECT class_num FROM roster);
```

Here, the optimizer can recognize that the `IN` clause requires the subquery to return only one instance of each class number from the `roster` table. In this case, the query can be executed as a *semi-join*—that is, an operation that returns only one instance of each row in `class` that is matched by rows in `roster`.

Before MySQL 5.6.6, the outer query specification was limited to simple table scans or inner joins using comma syntax, and view references were not possible. As of 5.6.6, outer join and inner join syntax is permitted in the outer query specification, and the restriction that table references must be base tables has been lifted.

In MySQL, a subquery must satisfy these criteria to be handled as a semi-join:

- It must be an `IN` (or `=ANY`) subquery that appears at the top level of the `WHERE` or `ON` clause, possibly as a term in an `AND` expression. For example:

  ```
  SELECT ...
  FROM ot1, ...
  WHERE (oe1, ...) IN (SELECT ie1, ... FROM it1, ... WHERE ...);
  ```

  Here, `ot_i` and `it_i` represent tables in the outer and inner parts of the query, and `oe_i` and `ie_i` represent expressions that refer to columns in the outer and inner tables.

- It must be a single `SELECT` without `UNION` constructs.

- It must not contain a `GROUP BY` or `HAVING` clause or aggregate functions.

- It must not have `ORDER BY` with `LIMIT`.

- The number of outer and inner tables together must be less than the maximum number of tables permitted in a join.

The subquery may be correlated or uncorrelated. `DISTINCT` is permitted, as is `LIMIT` unless `ORDER BY` is also used.

If a subquery meets the preceding criteria, MySQL converts it to a semi-join and makes a cost-based choice from these strategies:

- Convert the subquery to a join, or use table pullout and run the query as an inner join between subquery tables and outer tables. Table pullout pulls a table out from the subquery to the outer query.

- Duplicate Weedout: Run the semi-join as if it was a join and remove duplicate records using a temporary table.

- FirstMatch: When scanning the inner tables for row combinations and there are multiple instances of a given value group, choose one rather than returning them all. This "shortcuts" scanning and eliminates production of unnecessary rows.

- LooseScan: Scan a subquery table using an index that enables a single value to be chosen from each subquery's value group.

- Materialize the subquery into a temporary table with an index and use the temporary table to perform a join. The index is used to remove duplicates. The index might also be used later for lookups when joining the temporary table with the outer tables; if not, the table is scanned.

Each of these strategies except Duplicate Weedout can be enabled or disabled using the optimizer_switch system variable. The semijoin flag controls whether semi-joins are used. If it is set to on, the firstmatch, loosescan, and materialization flags enable finer control over the permitted semi-join strategies. These flags are on by default. See Section 8.8.6.2, “Controlling Switchable Optimizations”.

The use of semi-join strategies is indicated in EXPLAIN output as follows:

- Semi-joined tables show up in the outer select. EXPLAIN EXTENDED plus SHOW WARNINGS shows the rewritten query, which displays the semi-join structure. From this you can get an idea about which tables were pulled out of the semi-join. If a subquery was converted to a semi-join, you will see that the subquery predicate is gone and its tables and WHERE clause were merged into the outer query join list and WHERE clause.

- Temporary table use for Duplicate Weedout is indicated by Start temporary and End temporary in the Extra column. Tables that were not pulled out and are in the range of EXPLAIN output rows covered by Start temporary and End temporary will have their rowid in the temporary table.

- FirstMatch(*tbl_name*) in the Extra column indicates join shortcutting.

- LooseScan(*m*..*n*) in the Extra column indicates use of the LooseScan strategy. *m* and *n* are key part numbers.

- Temporary table use for materialization is indicated by rows with a select_type value of MATERIALIZED and rows with a table value of <subquery*N*>.

Optimizing Subqueries with Subquery Materialization

The optimizer uses subquery materialization as a strategy that enables more efficient subquery processing.

If materialization is not used, the optimizer sometimes rewrites a noncorrelated subquery as a correlated subquery. For example, the following IN subquery is noncorrelated (*where_condition* involves only columns from t2 and not t1):

```
SELECT * FROM t1
WHERE t1.a IN (SELECT t2.b FROM t2 WHERE where_condition);
```

The optimizer might rewrite this as an EXISTS correlated subquery:

```
SELECT * FROM t1
WHERE EXISTS (SELECT t2.b FROM t2 WHERE where_condition AND t1.a=t2.b);
```

Subquery materialization using a temporary table avoids such rewrites and makes it possible to execute the subquery only once rather than once per row of the outer query. Materialization speeds up query execution by generating a subquery result as a temporary table, normally in memory. The first time MySQL needs the subquery result, it materializes that result into a temporary table. Any subsequent time the result is needed, MySQL refers again to the temporary table. The table is indexed with a hash index to make lookups fast and inexpensive. The index is unique, which makes the table smaller because it has no duplicates.

Subquery materialization attempts to use an in-memory temporary table when possible, falling back to on-disk storage if the table becomes too large. See Section 8.4.4, “How MySQL Uses Internal Temporary Tables”.

For subquery materialization to be used in MySQL, the `materialization` flag of the `optimizer_switch` system variable must be `on`. Materialization then applies to subquery predicates that appear anywhere (in the select list, `WHERE`, `ON`, `GROUP BY`, `HAVING`, or `ORDER BY`), for predicates that fall into any of these use cases:

- The predicate has this form, when no outer expression *`oe_i`* or inner expression *`ie_i`* is nullable. *`N`* can be 1 or larger.

  ```
  (oe_1, oe_2, ..., oe_N) [NOT] IN (SELECT ie_1, i_2, ..., ie_N ...)
  ```

- The predicate has this form, when there is a single outer expression *`oe`* and inner expression *`ie`*. The expressions can be nullable.

  ```
  oe [NOT] IN (SELECT ie ...)
  ```

- The predicate is `IN` or `NOT IN` and a result of `UNKNOWN` (`NULL`) has the same meaning as a result of `FALSE`.

The following examples illustrate how the requirement for equivalence of `UNKNOWN` and `FALSE` predicate evaluation affects whether subquery materialization can be used. Assume that *`where_condition`* involves columns only from `t2` and not `t1` so that the subquery is noncorrelated.

This query is subject to materialization:

```
SELECT * FROM t1
WHERE t1.a IN (SELECT t2.b FROM t2 WHERE where_condition);
```

Here, it does not matter whether the `IN` predicate returns `UNKNOWN` or `FALSE`. Either way, the row from `t1` is not included in the query result.

An example where subquery materialization will not be used is the following query, where `t2.b` is a nullable column.

```
SELECT * FROM t1
WHERE (t1.a,t1.b) NOT IN (SELECT t2.a,t2.b FROM t2
                          WHERE where_condition);
```

Use of `EXPLAIN` with a query can give some indication of whether the optimizer uses subquery materialization. Compared to query execution that does not use materialization, `select_type` may change from `DEPENDENT SUBQUERY` to `SUBQUERY`. This indicates that, for a subquery that would be executed once per outer row, materialization enables the subquery to be executed just once. In addition, for `EXPLAIN EXTENDED`, the text displayed by a following `SHOW WARNINGS` will include `materialize materialize` and `materialized-subquery`.

Optimizing Subqueries in the `FROM` Clause (Derived Tables)

As of MySQL 5.6.3, the optimizer more efficiently handles subqueries in the `FROM` clause (that is, derived tables):

- Materialization of subqueries in the `FROM` clause is postponed until their contents are needed during query execution, which improves performance:
  - Previously, subqueries in the `FROM` clause were materialized for `EXPLAIN SELECT` statements. This resulted in partial `SELECT` execution, even though the purpose of `EXPLAIN` is to obtain query plan information, not to execute the query. This materialization no longer occurs, so `EXPLAIN` is faster for such queries.
  - For non-`EXPLAIN` queries, delay of materialization may result in not having to do it at all. Consider a query that joins the result of a subquery in the `FROM` clause to another table: If the optimizer processes that other table first and finds that it returns no rows, the join need not be carried out further and the optimizer can completely skip materializing the subquery.
- During query execution, the optimizer may add an index to a derived table to speed up row retrieval from it.

Consider the following `EXPLAIN` statement, for which a subquery appears in the `FROM` clause of a `SELECT` query:

```
EXPLAIN SELECT * FROM (SELECT * FROM t1);
```

The optimizer avoids materializing the subquery by delaying it until the result is needed during `SELECT` execution. In this case, the query is not executed, so the result is never needed.

Even for queries that are executed, delay of subquery materialization may permit the optimizer to avoid materialization entirely. Consider the following query, which joins the result of a subquery in the `FROM` clause to another table:

```
SELECT * FROM t1
  JOIN (SELECT t2.f1 FROM t2) AS derived_t2 ON t1.f2=derived_t2.f1
  WHERE t1.f1 > 0;
```

If the optimization processes `t1` first and the `WHERE` clause produces an empty result, the join must necessarily be empty and the subquery need not be materialized.

In the worst case (derived tables are materialized), query execution will take the same time as before MySQL 5.6.3 because no additional work is done. In the best case (derived tables are not materialized), query execution will be quicker by the time needed to perform materialization.

For cases when materialization is required for a subquery in the `FROM` clause, the optimizer may speed up access to the result by adding an index to the materialized table. If such an index would permit `ref` access to the table, it can greatly reduce amount of data that must be read during query execution. Consider the following query:

```
SELECT * FROM t1
  JOIN (SELECT * FROM t2) AS derived_t2 ON t1.f1=derived_t2.f1;
```

The optimizer constructs an index over column `f1` from `derived_t2` if doing so would permit the use of `ref` access for the lowest cost execution plan. After adding the index, the optimizer can treat the materialized derived table the same as a usual table with an index, and it benefits similarly from the generated index. The overhead of index creation is negligible compared to the cost of query execution without the index. If ref access would result in higher cost than some other access method, no index is created and the optimizer loses nothing.

Optimizing Subqueries with EXISTS Strategy

Certain optimizations are applicable to comparisons that use the IN operator to test subquery results (or that use =ANY, which is equivalent). This section discusses these optimizations, particularly with regard to the challenges that NULL values present. The last part of the discussion includes suggestions on what you can do to help the optimizer.

Consider the following subquery comparison:

```
outer_expr IN (SELECT inner_expr FROM ... WHERE subquery_where)
```

MySQL evaluates queries “from outside to inside.” That is, it first obtains the value of the outer expression *outer_expr*, and then runs the subquery and captures the rows that it produces.

A very useful optimization is to “inform” the subquery that the only rows of interest are those where the inner expression *inner_expr* is equal to *outer_expr*. This is done by pushing down an appropriate equality into the subquery's WHERE clause. That is, the comparison is converted to this:

```
EXISTS (SELECT 1 FROM ... WHERE subquery_where AND outer_expr=inner_expr)
```

After the conversion, MySQL can use the pushed-down equality to limit the number of rows that it must examine when evaluating the subquery.

More generally, a comparison of *N* values to a subquery that returns *N*-value rows is subject to the same conversion. If *oe_i* and *ie_i* represent corresponding outer and inner expression values, this subquery comparison:

```
(oe_1, ..., oe_N) IN
  (SELECT ie_1, ..., ie_N FROM ... WHERE subquery_where)
```

Becomes:

```
EXISTS (SELECT 1 FROM ... WHERE subquery_where
                          AND oe_1 = ie_1
                          AND ...
                          AND oe_N = ie_N)
```

The following discussion assumes a single pair of outer and inner expression values for simplicity.

The conversion just described has its limitations. It is valid only if we ignore possible NULL values. That is, the “pushdown” strategy works as long as both of these two conditions are true:

- *outer_expr* and *inner_expr* cannot be NULL.
- You do not need to distinguish NULL from FALSE subquery results. (If the subquery is a part of an OR or AND expression in the WHERE clause, MySQL assumes that you do not care.)

When either or both of those conditions do not hold, optimization is more complex.

Suppose that *outer_expr* is known to be a non-NULL value but the subquery does not produce a row such that *outer_expr* = *inner_expr*. Then *outer_expr* IN (SELECT ...) evaluates as follows:

- NULL, if the SELECT produces any row where *inner_expr* is NULL

- FALSE, if the SELECT produces only non-NULL values or produces nothing

In this situation, the approach of looking for rows with *outer_expr = inner_expr* is no longer valid. It is necessary to look for such rows, but if none are found, also look for rows where *inner_expr* is NULL. Roughly speaking, the subquery can be converted to:

```
EXISTS (SELECT 1 FROM ... WHERE subquery_where AND
        (outer_expr=inner_expr OR inner_expr IS NULL))
```

The need to evaluate the extra IS NULL condition is why MySQL has the ref_or_null access method:

```
mysql> EXPLAIN
    -> SELECT outer_expr IN (SELECT t2.maybe_null_key
    ->                       FROM t2, t3 WHERE ...)
    -> FROM t1;
*************************** 1. row ***************************
           id: 1
  select_type: PRIMARY
        table: t1
...
*************************** 2. row ***************************
           id: 2
  select_type: DEPENDENT SUBQUERY
        table: t2
         type: ref_or_null
possible_keys: maybe_null_key
          key: maybe_null_key
      key_len: 5
          ref: func
         rows: 2
        Extra: Using where; Using index
...
```

The unique_subquery and index_subquery subquery-specific access methods also have “or NULL” variants. However, prior to MySQL 5.7.3, they are not visible in EXPLAIN output, so you must use EXPLAIN EXTENDED followed by SHOW WARNINGS (note the checking NULL in the warning message):

```
mysql> EXPLAIN EXTENDED
    -> SELECT outer_expr IN (SELECT maybe_null_key FROM t2) FROM t1\G
*************************** 1. row ***************************
           id: 1
  select_type: PRIMARY
        table: t1
...
*************************** 2. row ***************************
           id: 2
  select_type: DEPENDENT SUBQUERY
        table: t2
         type: index_subquery
possible_keys: maybe_null_key
          key: maybe_null_key
      key_len: 5
          ref: func
         rows: 2
        Extra: Using index

mysql> SHOW WARNINGS\G
*************************** 1. row ***************************
  Level: Note
   Code: 1003
Message: select (`test`.`t1`.`outer_expr`,
         (((`test`.`t1`.`outer_expr`) in t2 on
         maybe_null_key checking NULL))) AS `outer_expr IN (SELECT
         maybe_null_key FROM t2)` from `test`.`t1`
```

The additional `OR ... IS NULL` condition makes query execution slightly more complicated (and some optimizations within the subquery become inapplicable), but generally this is tolerable.

The situation is much worse when *`outer_expr`* can be `NULL`. According to the SQL interpretation of `NULL` as “unknown value,” `NULL IN (SELECT inner_expr ...)` should evaluate to:

- `NULL`, if the `SELECT` produces any rows
- `FALSE`, if the `SELECT` produces no rows

For proper evaluation, it is necessary to be able to check whether the `SELECT` has produced any rows at all, so *`outer_expr = inner_expr`* cannot be pushed down into the subquery. This is a problem, because many real world subqueries become very slow unless the equality can be pushed down.

Essentially, there must be different ways to execute the subquery depending on the value of *`outer_expr`*.

The optimizer chooses SQL compliance over speed, so it accounts for the possibility that *`outer_expr`* might be `NULL`.

If *`outer_expr`* is `NULL`, to evaluate the following expression, it is necessary to run the `SELECT` to determine whether it produces any rows:

```
NULL IN (SELECT inner_expr FROM ... WHERE subquery_where)
```

It is necessary to run the original `SELECT` here, without any pushed-down equalities of the kind mentioned earlier.

On the other hand, when *`outer_expr`* is not `NULL`, it is absolutely essential that this comparison:

```
outer_expr IN (SELECT inner_expr FROM ... WHERE subquery_where)
```

be converted to this expression that uses a pushed-down condition:

```
EXISTS (SELECT 1 FROM ... WHERE subquery_where AND outer_expr=inner_expr)
```

Without this conversion, subqueries will be slow. To solve the dilemma of whether to push down or not push down conditions into the subquery, the conditions are wrapped in “trigger” functions. Thus, an expression of the following form:

```
outer_expr IN (SELECT inner_expr FROM ... WHERE subquery_where)
```

is converted into:

```
EXISTS (SELECT 1 FROM ... WHERE subquery_where
                          AND trigcond(outer_expr=inner_expr))
```

More generally, if the subquery comparison is based on several pairs of outer and inner expressions, the conversion takes this comparison:

```
(oe_1, ..., oe_N) IN (SELECT ie_1, ..., ie_N FROM ... WHERE subquery_where)
```

and converts it to this expression:

```
EXISTS (SELECT 1 FROM ... WHERE subquery_where
                          AND trigcond(oe_1=ie_1)
                          AND ...
                          AND trigcond(oe_N=ie_N)
       )
```

Each `trigcond(X)` is a special function that evaluates to the following values:

- `X` when the “linked” outer expression `oe_i` is not `NULL`
- `TRUE` when the “linked” outer expression `oe_i` is `NULL`

Note that trigger functions are *not* triggers of the kind that you create with `CREATE TRIGGER`.

Equalities that are wrapped into `trigcond()` functions are not first class predicates for the query optimizer. Most optimizations cannot deal with predicates that may be turned on and off at query execution time, so they assume any `trigcond(X)` to be an unknown function and ignore it. At the moment, triggered equalities can be used by those optimizations:

- Reference optimizations: `trigcond(X=Y [OR Y IS NULL])` can be used to construct `ref`, `eq_ref`, or `ref_or_null` table accesses.
- Index lookup-based subquery execution engines: `trigcond(X=Y)` can be used to construct `unique_subquery` or `index_subquery` accesses.
- Table-condition generator: If the subquery is a join of several tables, the triggered condition will be checked as soon as possible.

When the optimizer uses a triggered condition to create some kind of index lookup-based access (as for the first two items of the preceding list), it must have a fallback strategy for the case when the condition is turned off. This fallback strategy is always the same: Do a full table scan. In `EXPLAIN` output, the fallback shows up as `Full scan on NULL key` in the `Extra` column:

```
mysql> EXPLAIN SELECT t1.col1,
    -> t1.col1 IN (SELECT t2.key1 FROM t2 WHERE t2.col2=t1.col2) FROM t1\G
*************************** 1. row ***************************
           id: 1
  select_type: PRIMARY
        table: t1
        ...
*************************** 2. row ***************************
           id: 2
  select_type: DEPENDENT SUBQUERY
        table: t2
         type: index_subquery
possible_keys: key1
          key: key1
      key_len: 5
          ref: func
         rows: 2
        Extra: Using where; Full scan on NULL key
```

If you run `EXPLAIN EXTENDED` followed by `SHOW WARNINGS`, you can see the triggered condition:

```
*************************** 1. row ***************************
  Level: Note
   Code: 1003
Message: select `test`.`t1`.`col1` AS `col1`,
         <in_optimizer>(`test`.`t1`.`col1`,
         <exists>(<index_lookup>(<cache>(`test`.`t1`.`col1`) in t2
         on key1 checking NULL
         where (`test`.`t2`.`col2` = `test`.`t1`.`col2`) having
         trigcond(<is_not_null_test>(`test`.`t2`.`key1`))))) AS
         `t1.col1 IN (select t2.key1 from t2 where t2.col2=t1.col2)`
         from `test`.`t1`
```

The use of triggered conditions has some performance implications. A NULL IN (SELECT ...) expression now may cause a full table scan (which is slow) when it previously did not. This is the price paid for correct results (the goal of the trigger-condition strategy was to improve compliance and not speed).

For multiple-table subqueries, execution of NULL IN (SELECT ...) will be particularly slow because the join optimizer does not optimize for the case where the outer expression is NULL. It assumes that subquery evaluations with NULL on the left side are very rare, even if there are statistics that indicate otherwise. On the other hand, if the outer expression might be NULL but never actually is, there is no performance penalty.

To help the query optimizer better execute your queries, use these tips:

- Declare a column as NOT NULL if it really is. (This also helps other aspects of the optimizer by simplifying condition testing for the column.)

- If you do not need to distinguish a NULL from FALSE subquery result, you can easily avoid the slow execution path. Replace a comparison that looks like this:

  ```
  outer_expr IN (SELECT inner_expr FROM ...)
  ```

  with this expression:

  ```
  (outer_expr IS NOT NULL) AND (outer_expr IN (SELECT inner_expr FROM ...))
  ```

  Then NULL IN (SELECT ...) will never be evaluated because MySQL stops evaluating AND parts as soon as the expression result is clear.

The subquery_materialization_cost_based enables control over the choice between subquery materialization and IN -> EXISTS subquery transformation. See Section 8.8.6.2, “Controlling Switchable Optimizations”.

8.2.1.19 Optimizing LIMIT Queries

If you need only a specified number of rows from a result set, use a LIMIT clause in the query, rather than fetching the whole result set and throwing away the extra data.

MySQL sometimes optimizes a query that has a LIMIT *row_count* clause and no HAVING clause:

- If you select only a few rows with LIMIT, MySQL uses indexes in some cases when normally it would prefer to do a full table scan.

- If you use LIMIT *row_count* with ORDER BY, MySQL ends the sorting as soon as it has found the first *row_count* rows of the sorted result, rather than sorting the entire result. If ordering is done by using an index, this is very fast. If a filesort must be done, all rows that match the query without the LIMIT clause are selected, and most or all of them are sorted, before the first *row_count* are found. After the initial rows have been found, MySQL does not sort any remainder of the result set.

- When combining LIMIT *row_count* with DISTINCT, MySQL stops as soon as it finds *row_count* unique rows.

- In some cases, a GROUP BY can be resolved by reading the key in order (or doing a sort on the key) and then calculating summaries until the key value changes. In this case, LIMIT *row_count* does not calculate any unnecessary GROUP BY values.

- As soon as MySQL has sent the required number of rows to the client, it aborts the query unless you are using SQL_CALC_FOUND_ROWS.

- LIMIT 0 quickly returns an empty set. This can be useful for checking the validity of a query. When using one of the MySQL APIs, it can also be employed for obtaining the types of the result columns. (This trick does not work in the MySQL Monitor (the mysql program), which merely displays Empty set in such cases; instead, use SHOW COLUMNS or DESCRIBE for this purpose.)

- When the server uses temporary tables to resolve the query, it uses the LIMIT *row_count* clause to calculate how much space is required.

The optimizer does handle queries (and subqueries) of the following form:

```
SELECT ... FROM single_table ... ORDER BY non_index_column [DESC] LIMIT [M,]N;
```

That type of query is common in web applications that display only a few rows from a larger result set. For example:

```
SELECT col1, ... FROM t1 ... ORDER BY name LIMIT 10;
SELECT col1, ... FROM t1 ... ORDER BY RAND() LIMIT 15;
```

The sort buffer has a size of sort_buffer_size. If the sort elements for *N* rows are small enough to fit in the sort buffer (*M*+*N* rows if *M* was specified), the server can avoid using a merge file and perform the sort entirely in memory by treating the sort buffer as a priority queue:

- Scan the table, inserting the select list columns from each selected row in sorted order in the queue. If the queue is full, bump out the last row in the sort order.

- Return the first *N* rows from the queue. (If *M* was specified, skip the first *M* rows and return the next *N* rows.)

Previously, the server performed this operation by using a merge file for the sort:

- Scan the table, repeating these steps through the end of the table:

  - Select rows until the sort buffer is filled.

  - Write the first *N* rows in the buffer (*M*+*N* rows if *M* was specified) to a merge file.

- Sort the merge file and return the first *N* rows. (If *M* was specified, skip the first *M* rows and return the next *N* rows.)

The cost of the table scan is the same for the queue and merge-file methods, so the optimizer chooses between methods based on other costs:

- The queue method involves more CPU for inserting rows into the queue in order

- The merge-file method has I/O costs to write and read the file and CPU cost to sort it

The optimizer considers the balance between these factors for particular values of *N* and the row size.

8.2.1.20 How to Avoid Full Table Scans

The output from EXPLAIN shows ALL in the type column when MySQL uses a full table scan to resolve a query. This usually happens under the following conditions:

- The table is so small that it is faster to perform a table scan than to bother with a key lookup. This is common for tables with fewer than 10 rows and a short row length.

- There are no usable restrictions in the ON or WHERE clause for indexed columns.

- You are comparing indexed columns with constant values and MySQL has calculated (based on the index tree) that the constants cover too large a part of the table and that a table scan would be faster. See Section 8.2.1.2, “How MySQL Optimizes WHERE Clauses”.

- You are using a key with low cardinality (many rows match the key value) through another column. In this case, MySQL assumes that by using the key it probably will do many key lookups and that a table scan would be faster.

For small tables, a table scan often is appropriate and the performance impact is negligible. For large tables, try the following techniques to avoid having the optimizer incorrectly choose a table scan:

- Use ANALYZE TABLE *tbl_name* to update the key distributions for the scanned table. See Section 13.7.2.1, “ANALYZE TABLE Syntax”.

- Use FORCE INDEX for the scanned table to tell MySQL that table scans are very expensive compared to using the given index:

  ```
  SELECT * FROM t1, t2 FORCE INDEX (index_for_column)
    WHERE t1.col_name=t2.col_name;
  ```

  See Section 13.2.9.3, “Index Hint Syntax”.

- Start mysqld with the --max-seeks-for-key=1000 option or use SET max_seeks_for_key=1000 to tell the optimizer to assume that no key scan causes more than 1,000 key seeks. See Section 5.1.4, “Server System Variables”.

8.2.2 Optimizing DML Statements

This section explains how to speed up the data manipulation language (DML) statements, INSERT, UPDATE, and DELETE. Traditional OLTP applications and modern web applications typically do many small DML operations, where concurrency is vital. Data analysis and reporting applications typically run DML operations that affect many rows at once, where the main considerations is the I/O to write large amounts of data and keep indexes up-to-date. For inserting and updating large volumes of data (known in the industry as ETL, for “extract-transform-load”), sometimes you use other SQL statements or external commands, that mimic the effects of INSERT, UPDATE, and DELETE statements.

8.2.2.1 Speed of INSERT Statements

To optimize insert speed, combine many small operations into a single large operation. Ideally, you make a single connection, send the data for many new rows at once, and delay all index updates and consistency checking until the very end.

The time required for inserting a row is determined by the following factors, where the numbers indicate approximate proportions:

- Connecting: (3)

- Sending query to server: (2)

- Parsing query: (2)
- Inserting row: (1 × size of row)
- Inserting indexes: (1 × number of indexes)
- Closing: (1)

This does not take into consideration the initial overhead to open tables, which is done once for each concurrently running query.

The size of the table slows down the insertion of indexes by log *N*, assuming B-tree indexes.

You can use the following methods to speed up inserts:

- If you are inserting many rows from the same client at the same time, use INSERT statements with multiple VALUES lists to insert several rows at a time. This is considerably faster (many times faster in some cases) than using separate single-row INSERT statements. If you are adding data to a nonempty table, you can tune the bulk_insert_buffer_size variable to make data insertion even faster. See Section 5.1.4, "Server System Variables".
- When loading a table from a text file, use LOAD DATA INFILE. This is usually 20 times faster than using INSERT statements. See Section 13.2.6, "LOAD DATA INFILE Syntax".
- Take advantage of the fact that columns have default values. Insert values explicitly only when the value to be inserted differs from the default. This reduces the parsing that MySQL must do and improves the insert speed.
- See Section 8.5.4, "Bulk Data Loading for InnoDB Tables" for tips specific to InnoDB tables.
- See Section 8.6.2, "Bulk Data Loading for MyISAM Tables" for tips specific to MyISAM tables.

8.2.2.2 Speed of UPDATE Statements

An update statement is optimized like a SELECT query with the additional overhead of a write. The speed of the write depends on the amount of data being updated and the number of indexes that are updated. Indexes that are not changed do not get updated.

Another way to get fast updates is to delay updates and then do many updates in a row later. Performing multiple updates together is much quicker than doing one at a time if you lock the table.

For a MyISAM table that uses dynamic row format, updating a row to a longer total length may split the row. If you do this often, it is very important to use OPTIMIZE TABLE occasionally. See Section 13.7.2.4, "OPTIMIZE TABLE Syntax".

8.2.2.3 Speed of DELETE Statements

The time required to delete individual rows in a MyISAM table is exactly proportional to the number of indexes. To delete rows more quickly, you can increase the size of the key cache by increasing the key_buffer_size system variable. See Section 8.11.2, "Tuning Server Parameters".

To delete all rows from a MyISAM table, TRUNCATE TABLE *tbl_name* is faster than than DELETE FROM *tbl_name*. Truncate operations are not transaction-safe; an error occurs when attempting one in the course of an active transaction or active table lock. See Section 13.1.27, "TRUNCATE TABLE Syntax".

8.2.3 Optimizing Database Privileges

The more complex your privilege setup, the more overhead applies to all SQL statements. Simplifying the privileges established by GRANT statements enables MySQL to reduce permission-checking overhead when clients execute statements. For example, if you do not grant any table-level or column-level privileges, the server need not ever check the contents of the tables_priv and columns_priv tables. Similarly, if you place no resource limits on any accounts, the server does not have to perform resource counting. If you have a very high statement-processing load, consider using a simplified grant structure to reduce permission-checking overhead.

8.2.4 Optimizing INFORMATION_SCHEMA Queries

Applications that monitor the database can make frequent use of the INFORMATION_SCHEMA tables. Certain types of queries for INFORMATION_SCHEMA tables can be optimized to execute more quickly. The goal is to minimize file operations (for example, scanning a directory or opening a table file) to collect the information that makes up these dynamic tables. These optimizations do have an effect on how collations are used for searches in INFORMATION_SCHEMA tables. For more information, see Section 10.1.7.9, "Collation and INFORMATION_SCHEMA Searches".

1) Try to use constant lookup values for database and table names in the WHERE clause

You can take advantage of this principle as follows:

- To look up databases or tables, use expressions that evaluate to a constant, such as literal values, functions that return a constant, or scalar subqueries.
- Avoid queries that use a nonconstant database name lookup value (or no lookup value) because they require a scan of the data directory to find matching database directory names.
- Within a database, avoid queries that use a nonconstant table name lookup value (or no lookup value) because they require a scan of the database directory to find matching table files.

This principle applies to the INFORMATION_SCHEMA tables shown in the following table, which shows the columns for which a constant lookup value enables the server to avoid a directory scan. For example, if you are selecting from TABLES, using a constant lookup value for TABLE_SCHEMA in the WHERE clause enables a data directory scan to be avoided.

Table	Column to specify to avoid data directory scan	Column to specify to avoid database directory scan
COLUMNS	TABLE_SCHEMA	TABLE_NAME
KEY_COLUMN_USAGE	TABLE_SCHEMA	TABLE_NAME
PARTITIONS	TABLE_SCHEMA	TABLE_NAME
REFERENTIAL_CONSTRAINTS	CONSTRAINT_SCHEMA	TABLE_NAME
STATISTICS	TABLE_SCHEMA	TABLE_NAME
TABLES	TABLE_SCHEMA	TABLE_NAME
TABLE_CONSTRAINTS	TABLE_SCHEMA	TABLE_NAME
TRIGGERS	EVENT_OBJECT_SCHEMA	EVENT_OBJECT_TABLE
VIEWS	TABLE_SCHEMA	TABLE_NAME

The benefit of a query that is limited to a specific constant database name is that checks need be made only for the named database directory. Example:

```
SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES
WHERE TABLE_SCHEMA = 'test';
```

Use of the literal database name test enables the server to check only the test database directory, regardless of how many databases there might be. By contrast, the following query is less efficient because it requires a scan of the data directory to determine which database names match the pattern 'test%':

```
SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES
WHERE TABLE_SCHEMA LIKE 'test%';
```

For a query that is limited to a specific constant table name, checks need be made only for the named table within the corresponding database directory. Example:

```
SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES
WHERE TABLE_SCHEMA = 'test' AND TABLE_NAME = 't1';
```

Use of the literal table name t1 enables the server to check only the files for the t1 table, regardless of how many tables there might be in the test database. By contrast, the following query requires a scan of the test database directory to determine which table names match the pattern 't%':

```
SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES
WHERE TABLE_SCHEMA = 'test' AND TABLE_NAME LIKE 't%';
```

The following query requires a scan of the database directory to determine matching database names for the pattern 'test%', and for each matching database, it requires a scan of the database directory to determine matching table names for the pattern 't%':

```
SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES
WHERE TABLE_SCHEMA = 'test%' AND TABLE_NAME LIKE 't%';
```

2) Write queries that minimize the number of table files that must be opened

For queries that refer to certain INFORMATION_SCHEMA table columns, several optimizations are available that minimize the number of table files that must be opened. Example:

```
SELECT TABLE_NAME, ENGINE FROM INFORMATION_SCHEMA.TABLES
WHERE TABLE_SCHEMA = 'test';
```

In this case, after the server has scanned the database directory to determine the names of the tables in the database, those names become available with no further file system lookups. Thus, TABLE_NAME requires no files to be opened. The ENGINE (storage engine) value can be determined by opening the table's .frm file, without touching other table files such as the .MYD or .MYI file.

Some values, such as INDEX_LENGTH for MyISAM tables, require opening the .MYD or .MYI file as well.

The file-opening optimization types are denoted thus:

- SKIP_OPEN_TABLE: Table files do not need to be opened. The information has already become available within the query by scanning the database directory.
- OPEN_FRM_ONLY: Only the table's .frm file need be opened.
- OPEN_TRIGGER_ONLY: Only the table's .TRG file need be opened.
- OPEN_FULL_TABLE: The unoptimized information lookup. The .frm, .MYD, and .MYI files must be opened.

The following list indicates how the preceding optimization types apply to INFORMATION_SCHEMA table columns. For tables and columns not named, none of the optimizations apply.

- COLUMNS: OPEN_FRM_ONLY applies to all columns

- KEY_COLUMN_USAGE: OPEN_FULL_TABLE applies to all columns
- PARTITIONS: OPEN_FULL_TABLE applies to all columns
- REFERENTIAL_CONSTRAINTS: OPEN_FULL_TABLE applies to all columns
- STATISTICS:

Column	Optimization type
TABLE_CATALOG	OPEN_FRM_ONLY
TABLE_SCHEMA	OPEN_FRM_ONLY
TABLE_NAME	OPEN_FRM_ONLY
NON_UNIQUE	OPEN_FRM_ONLY
INDEX_SCHEMA	OPEN_FRM_ONLY
INDEX_NAME	OPEN_FRM_ONLY
SEQ_IN_INDEX	OPEN_FRM_ONLY
COLUMN_NAME	OPEN_FRM_ONLY
COLLATION	OPEN_FRM_ONLY
CARDINALITY	OPEN_FULL_TABLE
SUB_PART	OPEN_FRM_ONLY
PACKED	OPEN_FRM_ONLY
NULLABLE	OPEN_FRM_ONLY
INDEX_TYPE	OPEN_FULL_TABLE
COMMENT	OPEN_FRM_ONLY

- TABLES:

Column	Optimization type
TABLE_CATALOG	SKIP_OPEN_TABLE
TABLE_SCHEMA	SKIP_OPEN_TABLE
TABLE_NAME	SKIP_OPEN_TABLE
TABLE_TYPE	OPEN_FRM_ONLY
ENGINE	OPEN_FRM_ONLY
VERSION	OPEN_FRM_ONLY
ROW_FORMAT	OPEN_FULL_TABLE
TABLE_ROWS	OPEN_FULL_TABLE
AVG_ROW_LENGTH	OPEN_FULL_TABLE
DATA_LENGTH	OPEN_FULL_TABLE
MAX_DATA_LENGTH	OPEN_FULL_TABLE
INDEX_LENGTH	OPEN_FULL_TABLE
DATA_FREE	OPEN_FULL_TABLE
AUTO_INCREMENT	OPEN_FULL_TABLE
CREATE_TIME	OPEN_FULL_TABLE

Column	Optimization type
UPDATE_TIME	OPEN_FULL_TABLE
CHECK_TIME	OPEN_FULL_TABLE
TABLE_COLLATION	OPEN_FRM_ONLY
CHECKSUM	OPEN_FULL_TABLE
CREATE_OPTIONS	OPEN_FRM_ONLY
TABLE_COMMENT	OPEN_FRM_ONLY

- TABLE_CONSTRAINTS: OPEN_FULL_TABLE applies to all columns

- TRIGGERS: OPEN_TRIGGER_ONLY applies to all columns

- VIEWS:

Column	Optimization type
TABLE_CATALOG	OPEN_FRM_ONLY
TABLE_SCHEMA	OPEN_FRM_ONLY
TABLE_NAME	OPEN_FRM_ONLY
VIEW_DEFINITION	OPEN_FRM_ONLY
CHECK_OPTION	OPEN_FRM_ONLY
IS_UPDATABLE	OPEN_FULL_TABLE
DEFINER	OPEN_FRM_ONLY
SECURITY_TYPE	OPEN_FRM_ONLY
CHARACTER_SET_CLIENT	OPEN_FRM_ONLY
COLLATION_CONNECTION	OPEN_FRM_ONLY

3) Use EXPLAIN to determine whether the server can use INFORMATION_SCHEMA optimizations for a query

This applies particularly for INFORMATION_SCHEMA queries that search for information from more than one database, which might take a long time and impact performance. The Extra value in EXPLAIN output indicates which, if any, of the optimizations described earlier the server can use to evaluate INFORMATION_SCHEMA queries. The following examples demonstrate the kinds of information you can expect to see in the Extra value.

```
mysql> EXPLAIN SELECT TABLE_NAME FROM INFORMATION_SCHEMA.VIEWS WHERE
    -> TABLE_SCHEMA = 'test' AND TABLE_NAME = 'v1'\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: VIEWS
         type: ALL
possible_keys: NULL
          key: TABLE_SCHEMA,TABLE_NAME
      key_len: NULL
          ref: NULL
         rows: NULL
        Extra: Using where; Open_frm_only; Scanned 0 databases
```

Use of constant database and table lookup values enables the server to avoid directory scans. For references to VIEWS.TABLE_NAME, only the .frm file need be opened.

```
mysql> EXPLAIN SELECT TABLE_NAME, ROW_FORMAT FROM INFORMATION_SCHEMA.TABLES\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: TABLES
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: NULL
        Extra: Open_full_table; Scanned all databases
```

No lookup values are provided (there is no WHERE clause), so the server must scan the data directory and each database directory. For each table thus identified, the table name and row format are selected. TABLE_NAME requires no further table files to be opened (the SKIP_OPEN_TABLE optimization applies). ROW_FORMAT requires all table files to be opened (OPEN_FULL_TABLE applies). EXPLAIN reports OPEN_FULL_TABLE because it is more expensive than SKIP_OPEN_TABLE.

```
mysql> EXPLAIN SELECT TABLE_NAME, TABLE_TYPE FROM INFORMATION_SCHEMA.TABLES
    -> WHERE TABLE_SCHEMA = 'test'\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: TABLES
         type: ALL
possible_keys: NULL
          key: TABLE_SCHEMA
      key_len: NULL
          ref: NULL
         rows: NULL
        Extra: Using where; Open_frm_only; Scanned 1 database
```

No table name lookup value is provided, so the server must scan the test database directory. For the TABLE_NAME and TABLE_TYPE columns, the SKIP_OPEN_TABLE and OPEN_FRM_ONLY optimizations apply, respectively. EXPLAIN reports OPEN_FRM_ONLY because it is more expensive.

```
mysql> EXPLAIN SELECT B.TABLE_NAME
    -> FROM INFORMATION_SCHEMA.TABLES AS A, INFORMATION_SCHEMA.COLUMNS AS B
    -> WHERE A.TABLE_SCHEMA = 'test'
    -> AND A.TABLE_NAME = 't1'
    -> AND B.TABLE_NAME = A.TABLE_NAME\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: A
         type: ALL
possible_keys: NULL
          key: TABLE_SCHEMA,TABLE_NAME
      key_len: NULL
          ref: NULL
         rows: NULL
        Extra: Using where; Skip_open_table; Scanned 0 databases
*************************** 2. row ***************************
           id: 1
  select_type: SIMPLE
        table: B
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: NULL
        Extra: Using where; Open_frm_only; Scanned all databases;
               Using join buffer
```

For the first EXPLAIN output row: Constant database and table lookup values enable the server to avoid directory scans for TABLES values. References to TABLES.TABLE_NAME require no further table files.

For the second EXPLAIN output row: All COLUMNS table values are OPEN_FRM_ONLY lookups, so COLUMNS.TABLE_NAME requires the .frm file to be opened.

```
mysql> EXPLAIN SELECT * FROM INFORMATION_SCHEMA.COLLATIONS\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: COLLATIONS
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: NULL
        Extra:
```

In this case, no optimizations apply because COLLATIONS is not one of the INFORMATION_SCHEMA tables for which optimizations are available.

8.2.5 Other Optimization Tips

This section lists a number of miscellaneous tips for improving query processing speed:

- Use persistent connections to the database to avoid connection overhead. If you cannot use persistent connections and you are initiating many new connections to the database, you may want to change the value of the thread_cache_size variable. See Section 8.11.2, “Tuning Server Parameters”.

- Always check whether all your queries really use the indexes that you have created in the tables. In MySQL, you can do this with the EXPLAIN statement. See Section 8.8.1, “Optimizing Queries with EXPLAIN”.

- Try to avoid complex SELECT queries on MyISAM tables that are updated frequently, to avoid problems with table locking that occur due to contention between readers and writers.

- MyISAM supports concurrent inserts: If a table has no free blocks in the middle of the data file, you can INSERT new rows into it at the same time that other threads are reading from the table. If it is important to be able to do this, consider using the table in ways that avoid deleting rows. Another possibility is to run OPTIMIZE TABLE to defragment the table after you have deleted a lot of rows from it. This behavior is altered by setting the concurrent_insert variable. You can force new rows to be appended (and therefore permit concurrent inserts), even in tables that have deleted rows. See Section 8.10.3, “Concurrent Inserts”.

- To fix any compression issues that may have occurred with ARCHIVE tables, you can use OPTIMIZE TABLE. See Section 14.6, “The ARCHIVE Storage Engine”.

- Use ALTER TABLE ... ORDER BY *expr1, expr2, ...* if you usually retrieve rows in *expr1, expr2, ...* order. By using this option after extensive changes to the table, you may be able to get higher performance.

- In some cases, it may make sense to introduce a column that is “hashed” based on information from other columns. If this column is short, reasonably unique, and indexed, it may be much faster than a “wide” index on many columns. In MySQL, it is very easy to use this extra column:

```
SELECT * FROM tbl_name
  WHERE hash_col=MD5(CONCAT(col1,col2))
  AND col1='constant' AND col2='constant';
```

- For MyISAM tables that change frequently, try to avoid all variable-length columns (VARCHAR, BLOB, and TEXT). The table uses dynamic row format if it includes even a single variable-length column. See Chapter 14, *Storage Engines*.

- It is normally not useful to split a table into different tables just because the rows become large. In accessing a row, the biggest performance hit is the disk seek needed to find the first byte of the row. After finding the data, most modern disks can read the entire row fast enough for most applications. The only cases where splitting up a table makes an appreciable difference is if it is a MyISAM table using dynamic row format that you can change to a fixed row size, or if you very often need to scan the table but do not need most of the columns. See Chapter 14, *Storage Engines*.

- If you often need to calculate results such as counts based on information from a lot of rows, it may be preferable to introduce a new table and update the counter in real time. An update of the following form is very fast:

  ```
  UPDATE tbl_name SET count_col=count_col+1 WHERE key_col=constant;
  ```

  This is very important when you use MySQL storage engines such as MyISAM that has only table-level locking (multiple readers with single writers). This also gives better performance with most database systems, because the row locking manager in this case has less to do.

- If you need to collect statistics from large log tables, use summary tables instead of scanning the entire log table. Maintaining the summaries should be much faster than trying to calculate statistics "live." Regenerating new summary tables from the logs when things change (depending on business decisions) is faster than changing the running application.

- If possible, classify reports as "live" or as "statistical," where data needed for statistical reports is created only from summary tables that are generated periodically from the live data.

- Take advantage of the fact that columns have default values. Insert values explicitly only when the value to be inserted differs from the default. This reduces the parsing that MySQL must do and improves the insert speed.

- In some cases, it is convenient to pack and store data into a BLOB column. In this case, you must provide code in your application to pack and unpack information, but this may save a lot of accesses at some stage. This is practical when you have data that does not conform well to a rows-and-columns table structure.

- Normally, try to keep all data nonredundant (observing what is referred to in database theory as *third normal form*). However, there may be situations in which it can be advantageous to duplicate information or create summary tables to gain more speed.

- Stored routines or UDFs (user-defined functions) may be a good way to gain performance for some tasks. See Section 18.2, "Using Stored Routines (Procedures and Functions)", and Section 22.3, "Adding New Functions to MySQL", for more information.

- You can increase performance by caching queries or answers in your application and then executing many inserts or updates together. If your database system supports table locks (as does MySQL), this should help to ensure that the index cache is only flushed once after all updates. You can also take advantage of MySQL's query cache to achieve similar results; see Section 8.9.3, "The MySQL Query Cache".

- Use multiple-row INSERT statements to store many rows with one SQL statement. (This is a relatively portable technique.)

- Use LOAD DATA INFILE to load large amounts of data. This is faster than using INSERT statements.

- Use AUTO_INCREMENT columns so that each row in a table can be identified by a single unique value.
- Use OPTIMIZE TABLE once in a while to avoid fragmentation with dynamic-format MyISAM tables. See Section 14.3.3, “MyISAM Table Storage Formats”.
- Use MEMORY tables when possible to get more speed. See Section 14.4, “The MEMORY Storage Engine”. MEMORY tables are useful for noncritical data that is accessed often, such as information about the last displayed banner for users who don't have cookies enabled in their Web browser. User sessions are another alternative available in many Web application environments for handling volatile state data.
- With Web servers, images and other binary assets should normally be stored as files. That is, store only a reference to the file rather than the file itself in the database. Most Web servers are better at caching files than database contents, so using files is generally faster.
- Columns with identical information in different tables should be declared to have identical data types so that joins based on the corresponding columns will be faster.
- Try to keep column names simple. For example, in a table named customer, use a column name of name instead of customer_name. To make your names portable to other SQL servers, consider keeping them shorter than 18 characters.
- If you need really high speed, look at the low-level interfaces for data storage that the different SQL servers support. For example, by accessing the MySQL MyISAM storage engine directly, you could get a speed increase of two to five times compared to using the SQL interface. To be able to do this, the data must be on the same server as the application, and usually it should only be accessed by one process (because external file locking is really slow). One could eliminate these problems by introducing low-level MyISAM commands in the MySQL server (this could be one easy way to get more performance if needed). By carefully designing the database interface, it should be quite easy to support this type of optimization.
- If you are using numeric data, it is faster in many cases to access information from a database (using a live connection) than to access a text file. Information in the database is likely to be stored in a more compact format than in the text file, so accessing it involves fewer disk accesses. You also save code in your application because you need not parse your text files to find line and column boundaries.
- Replication can provide a performance benefit for some operations. You can distribute client retrievals among replication servers to split up the load. To avoid slowing down the master while making backups, you can make backups using a slave server. See Chapter 16, *Replication*.
- Declaring a MyISAM table with the DELAY_KEY_WRITE=1 table option makes index updates faster because they are not flushed to disk until the table is closed. The downside is that if something kills the server while such a table is open, you must ensure that the table is okay by running the server with the --myisam-recover-options option, or by running myisamchk before restarting the server. (However, even in this case, you should not lose anything by using DELAY_KEY_WRITE, because the key information can always be generated from the data rows.)
- Use INSERT LOW_PRIORITY for supported nontransactional tables when you want to give SELECT statements higher priority than your inserts.
- Use SELECT HIGH_PRIORITY for supported nontransactional tables to get retrievals that jump the queue. That is, the SELECT is executed even if there is another client waiting to do a write.

  LOW_PRIORITY and HIGH_PRIORITY have an effect only for nontransactional storage engines that use only table-level locking.

8.3 Optimization and Indexes

The best way to improve the performance of SELECT operations is to create indexes on one or more of the columns that are tested in the query. The index entries act like pointers to the table rows, allowing the query to quickly determine which rows match a condition in the WHERE clause, and retrieve the other column values for those rows. All MySQL data types can be indexed.

Although it can be tempting to create an indexes for every possible column used in a query, unnecessary indexes waste space and waste time for MySQL to determine which indexes to use. You must find the right balance to achieve fast queries using the optimal set of indexes.

8.3.1 How MySQL Uses Indexes

Indexes are used to find rows with specific column values quickly. Without an index, MySQL must begin with the first row and then read through the entire table to find the relevant rows. The larger the table, the more this costs. If the table has an index for the columns in question, MySQL can quickly determine the position to seek to in the middle of the data file without having to look at all the data. If a table has 1,000 rows, this is at least 100 times faster than reading sequentially.

Most MySQL indexes (PRIMARY KEY, UNIQUE, INDEX, and FULLTEXT) are stored in B-trees. Exceptions are that indexes on spatial data types use R-trees, and that MEMORY tables also support hash indexes.

In general, indexes are used as described in the following discussion. Characteristics specific to hash indexes (as used in MEMORY tables) are described at the end of this section.

MySQL uses indexes for these operations:

- To find the rows matching a WHERE clause quickly.

- To eliminate rows from consideration. If there is a choice between multiple indexes, MySQL normally uses the index that finds the smallest number of rows (the most selective index).

- To retrieve rows from other tables when performing joins. MySQL can use indexes on columns more efficiently if they are declared as the same type and size. In this context, VARCHAR and CHAR are considered the same if they are declared as the same size. For example, VARCHAR(10) and CHAR(10) are the same size, but VARCHAR(10) and CHAR(15) are not.

  Comparison of dissimilar columns may prevent use of indexes if values cannot be compared directly without conversion. Suppose that a numeric column is compared to a string column. For a given value such as 1 in the numeric column, it might compare equal to any number of values in the string column such as '1', ' 1', '00001', or '01.e1'. This rules out use of any indexes for the string column.

- To find the MIN() or MAX() value for a specific indexed column *key_col*. This is optimized by a preprocessor that checks whether you are using WHERE *key_part_N* = *constant* on all key parts that occur before *key_col* in the index. In this case, MySQL does a single key lookup for each MIN() or MAX() expression and replaces it with a constant. If all expressions are replaced with constants, the query returns at once. For example:

```
SELECT MIN(key_part2),MAX(key_part2)
  FROM tbl_name WHERE key_part1=10;
```

- To sort or group a table if the sorting or grouping is done on a leftmost prefix of a usable key (for example, ORDER BY *key_part1*, *key_part2*). If all key parts are followed by DESC, the key is read in reverse order. See Section 8.2.1.15, “ORDER BY Optimization”, and Section 8.2.1.16, “GROUP BY Optimization”.

- In some cases, a query can be optimized to retrieve values without consulting the data rows. (An index that provides all the necessary results for a query is called a covering index.) If a query uses only columns from a table that are numeric and that form a leftmost prefix for some key, the selected values can be retrieved from the index tree for greater speed:

```
SELECT key_part3 FROM tbl_name
  WHERE key_part1=1
```

Indexes are less important for queries on small tables, or big tables where report queries process most or all of the rows. When a query needs to access most of the rows, reading sequentially is faster than working through an index. Sequential reads minimize disk seeks, even if not all the rows are needed for the query. See Section 8.2.1.20, “How to Avoid Full Table Scans” for details.

8.3.2 Using Primary Keys

The primary key for a table represents the column or set of columns that you use in your most vital queries. It has an associated index, for fast query performance. Query performance benefits from the `NOT NULL` optimization, because it cannot include any `NULL` values. With the `InnoDB` storage engine, the table data is physically organized to do ultra-fast lookups and sorts based on the primary key column or columns.

If your table is big and important, but does not have an obvious column or set of columns to use as a primary key, you might create a separate column with auto-increment values to use as the primary key. These unique IDs can serve as pointers to corresponding rows in other tables when you join tables using foreign keys.

8.3.3 Using Foreign Keys

If a table has many columns, and you query many different combinations of columns, it might be efficient to split the less-frequently used data into separate tables with a few columns each, and relate them back to the main table by duplicating the numeric ID column from the main table. That way, each small table can have a primary key for fast lookups of its data, and you can query just the set of columns that you need using a join operation. Depending on how the data is distributed, the queries might perform less I/O and take up less cache memory because the relevant columns are packed together on disk. (To maximize performance, queries try to read as few data blocks as possible from disk; tables with only a few columns can fit more rows in each data block.)

8.3.4 Column Indexes

The most common type of index involves a single column, storing copies of the values from that column in a data structure, allowing fast lookups for the rows with the corresponding column values. The B-tree data structure lets the index quickly find a specific value, a set of values, or a range of values, corresponding to operators such as `=`, `>`, `≤`, `BETWEEN`, `IN`, and so on, in a `WHERE` clause.

The maximum number of indexes per table and the maximum index length is defined per storage engine. See Chapter 14, *Storage Engines*. All storage engines support at least 16 indexes per table and a total index length of at least 256 bytes. Most storage engines have higher limits.

Prefix Indexes

With *`col_name(N)`* syntax in an index specification, you can create an index that uses only the first *`N`* characters of a string column. Indexing only a prefix of column values in this way can make the index file much smaller. When you index a `BLOB` or `TEXT` column, you *must* specify a prefix length for the index. For example:

```
CREATE TABLE test (blob_col BLOB, INDEX(blob_col(10)));
```

Prefixes can be up to 1000 bytes long (767 bytes for InnoDB tables, unless you have innodb_large_prefix set).

> **Note**
>
> Prefix limits are measured in bytes, while the prefix length in CREATE TABLE statements is interpreted as number of characters. *Take this into account when specifying a prefix length for a column that uses a multi-byte character set*.

FULLTEXT Indexes

You can also create FULLTEXT indexes. These are used for full-text searches. Only the InnoDB and MyISAM storage engines support FULLTEXT indexes and only for CHAR, VARCHAR, and TEXT columns. Indexing always takes place over the entire column and column prefix indexing is not supported. For details, see Section 12.9, “Full-Text Search Functions”.

Optimizations are applied to certain kinds of FULLTEXT queries against single InnoDB tables. Queries with these characteristics are particularly efficient:

- FULLTEXT queries that only return the document ID, or the document ID and the search rank.
- FULLTEXT queries that sort the matching rows in descending order of score and apply a LIMIT clause to take the top N matching rows. For this optimization to apply, there must be no WHERE clauses and only a single ORDER BY clause in descending order.
- FULLTEXT queries that retrieve only the COUNT(*) value of rows matching a search term, with no additional WHERE clauses. Code the WHERE clause as WHERE MATCH(*text*) AGAINST ('*other_text*'), without any > 0 comparison operator.

Spatial Indexes

You can also create indexes on spatial data types. Currently, only MyISAM supports R-tree indexes on spatial types. Other storage engines use B-trees for indexing spatial types (except for ARCHIVE, which does not support spatial type indexing).

Indexes in the MEMORY Storage Engine

The MEMORY storage engine uses HASH indexes by default, but also supports BTREE indexes.

8.3.5 Multiple-Column Indexes

MySQL can create composite indexes (that is, indexes on multiple columns). An index may consist of up to 16 columns. For certain data types, you can index a prefix of the column (see Section 8.3.4, “Column Indexes”).

MySQL can use multiple-column indexes for queries that test all the columns in the index, or queries that test just the first column, the first two columns, the first three columns, and so on. If you specify the columns in the right order in the index definition, a single composite index can speed up several kinds of queries on the same table.

A multiple-column index can be considered a sorted array, the rows of which contain values that are created by concatenating the values of the indexed columns.

> **Note**
>
> As an alternative to a composite index, you can introduce a column that is “hashed” based on information from other columns. If this column is short, reasonably unique,

> and indexed, it might be faster than a “wide” index on many columns. In MySQL, it is very easy to use this extra column:
>
> ```
> SELECT * FROM tbl_name
>   WHERE hash_col=MD5(CONCAT(val1,val2))
>   AND col1=val1 AND col2=val2;
> ```

Suppose that a table has the following specification:

```
CREATE TABLE test (
    id         INT NOT NULL,
    last_name  CHAR(30) NOT NULL,
    first_name CHAR(30) NOT NULL,
    PRIMARY KEY (id),
    INDEX name (last_name,first_name)
);
```

The name index is an index over the last_name and first_name columns. The index can be used for lookups in queries that specify values in a known range for combinations of last_name and first_name values. It can also be used for queries that specify just a last_name value because that column is a leftmost prefix of the index (as described later in this section). Therefore, the name index is used for lookups in the following queries:

```
SELECT * FROM test WHERE last_name='Widenius';

SELECT * FROM test
  WHERE last_name='Widenius' AND first_name='Michael';

SELECT * FROM test
  WHERE last_name='Widenius'
  AND (first_name='Michael' OR first_name='Monty');

SELECT * FROM test
  WHERE last_name='Widenius'
  AND first_name >='M' AND first_name < 'N';
```

However, the name index is *not* used for lookups in the following queries:

```
SELECT * FROM test WHERE first_name='Michael';

SELECT * FROM test
  WHERE last_name='Widenius' OR first_name='Michael';
```

Suppose that you issue the following SELECT statement:

```
mysql> SELECT * FROM tbl_name WHERE col1=val1 AND col2=val2;
```

If a multiple-column index exists on col1 and col2, the appropriate rows can be fetched directly. If separate single-column indexes exist on col1 and col2, the optimizer attempts to use the Index Merge optimization (see Section 8.2.1.4, “Index Merge Optimization”), or attempts to find the most restrictive index by deciding which index excludes more rows and using that index to fetch the rows.

If the table has a multiple-column index, any leftmost prefix of the index can be used by the optimizer to find rows. For example, if you have a three-column index on (col1, col2, col3), you have indexed search capabilities on (col1), (col1, col2), and (col1, col2, col3).

MySQL cannot use the index to perform lookups if the columns do not form a leftmost prefix of the index. Suppose that you have the SELECT statements shown here:

```
SELECT * FROM tbl_name WHERE col1=val1;
SELECT * FROM tbl_name WHERE col1=val1 AND col2=val2;

SELECT * FROM tbl_name WHERE col2=val2;
SELECT * FROM tbl_name WHERE col2=val2 AND col3=val3;
```

If an index exists on `(col1, col2, col3)`, only the first two queries use the index. The third and fourth queries do involve indexed columns, but `(col2)` and `(col2, col3)` are not leftmost prefixes of `(col1, col2, col3)`.

8.3.6 Verifying Index Usage

Always check whether all your queries really use the indexes that you have created in the tables. Use the `EXPLAIN` statement, as described in Section 8.8.1, “Optimizing Queries with `EXPLAIN`”.

8.3.7 `InnoDB` and `MyISAM` Index Statistics Collection

Storage engines collect statistics about tables for use by the optimizer. Table statistics are based on value groups, where a value group is a set of rows with the same key prefix value. For optimizer purposes, an important statistic is the average value group size.

MySQL uses the average value group size in the following ways:

- To estimate how may rows must be read for each `ref` access
- To estimate how many row a partial join will produce; that is, the number of rows that an operation of this form will produce:

  ```
  (...) JOIN tbl_name ON tbl_name.key = expr
  ```

As the average value group size for an index increases, the index is less useful for those two purposes because the average number of rows per lookup increases: For the index to be good for optimization purposes, it is best that each index value target a small number of rows in the table. When a given index value yields a large number of rows, the index is less useful and MySQL is less likely to use it.

The average value group size is related to table cardinality, which is the number of value groups. The `SHOW INDEX` statement displays a cardinality value based on *`N/S`*, where *`N`* is the number of rows in the table and *`S`* is the average value group size. That ratio yields an approximate number of value groups in the table.

For a join based on the `<=>` comparison operator, `NULL` is not treated differently from any other value: `NULL <=> NULL`, just as *`N`* `<=>` *`N`* for any other *`N`*.

However, for a join based on the `=` operator, `NULL` is different from non-`NULL` values: *`expr1`* `=` *`expr2`* is not true when *`expr1`* or *`expr2`* (or both) are `NULL`. This affects `ref` accesses for comparisons of the form *`tbl_name.key`* `=` *`expr`*: MySQL will not access the table if the current value of *`expr`* is `NULL`, because the comparison cannot be true.

For `=` comparisons, it does not matter how many `NULL` values are in the table. For optimization purposes, the relevant value is the average size of the non-`NULL` value groups. However, MySQL does not currently enable that average size to be collected or used.

For `InnoDB` and `MyISAM` tables, you have some control over collection of table statistics by means of the `innodb_stats_method` and `myisam_stats_method` system variables, respectively. These variables have three possible values, which differ as follows:

- When the variable is set to nulls_equal, all NULL values are treated as identical (that is, they all form a single value group).

  If the NULL value group size is much higher than the average non-NULL value group size, this method skews the average value group size upward. This makes index appear to the optimizer to be less useful than it really is for joins that look for non-NULL values. Consequently, the nulls_equal method may cause the optimizer not to use the index for ref accesses when it should.

- When the variable is set to nulls_unequal, NULL values are not considered the same. Instead, each NULL value forms a separate value group of size 1.

  If you have many NULL values, this method skews the average value group size downward. If the average non-NULL value group size is large, counting NULL values each as a group of size 1 causes the optimizer to overestimate the value of the index for joins that look for non-NULL values. Consequently, the nulls_unequal method may cause the optimizer to use this index for ref lookups when other methods may be better.

- When the variable is set to nulls_ignored, NULL values are ignored.

If you tend to use many joins that use <=> rather than =, NULL values are not special in comparisons and one NULL is equal to another. In this case, nulls_equal is the appropriate statistics method.

The innodb_stats_method system variable has a global value; the myisam_stats_method system variable has both global and session values. Setting the global value affects statistics collection for tables from the corresponding storage engine. Setting the session value affects statistics collection only for the current client connection. This means that you can force a table's statistics to be regenerated with a given method without affecting other clients by setting the session value of myisam_stats_method.

To regenerate table statistics, you can use any of the following methods:

- Execute myisamchk --stats_method=*method_name* --analyze
- Change the table to cause its statistics to go out of date (for example, insert a row and then delete it), and then set myisam_stats_method and issue an ANALYZE TABLE statement

Some caveats regarding the use of innodb_stats_method and myisam_stats_method:

- You can force table statistics to be collected explicitly, as just described. However, MySQL may also collect statistics automatically. For example, if during the course of executing statements for a table, some of those statements modify the table, MySQL may collect statistics. (This may occur for bulk inserts or deletes, or some ALTER TABLE statements, for example.) If this happens, the statistics are collected using whatever value innodb_stats_method or myisam_stats_method has at the time. Thus, if you collect statistics using one method, but the system variable is set to the other method when a table's statistics are collected automatically later, the other method will be used.
- There is no way to tell which method was used to generate statistics for a given table.
- These variables apply only to InnoDB and MyISAM tables. Other storage engines have only one method for collecting table statistics. Usually it is closer to the nulls_equal method.

8.3.8 Comparison of B-Tree and Hash Indexes

Understanding the B-tree and hash data structures can help predict how different queries perform on different storage engines that use these data structures in their indexes, particularly for the MEMORY storage engine that lets you choose B-tree or hash indexes.

B-Tree Index Characteristics

A B-tree index can be used for column comparisons in expressions that use the `=`, `>`, `>=`, `<`, `<=`, or `BETWEEN` operators. The index also can be used for `LIKE` comparisons if the argument to `LIKE` is a constant string that does not start with a wildcard character. For example, the following `SELECT` statements use indexes:

```
SELECT * FROM tbl_name WHERE key_col LIKE 'Patrick%';
SELECT * FROM tbl_name WHERE key_col LIKE 'Pat%_ck%';
```

In the first statement, only rows with `'Patrick' <= key_col < 'Patricl'` are considered. In the second statement, only rows with `'Pat' <= key_col < 'Pau'` are considered.

The following `SELECT` statements do not use indexes:

```
SELECT * FROM tbl_name WHERE key_col LIKE '%Patrick%';
SELECT * FROM tbl_name WHERE key_col LIKE other_col;
```

In the first statement, the `LIKE` value begins with a wildcard character. In the second statement, the `LIKE` value is not a constant.

If you use `... LIKE '%string%'` and *`string`* is longer than three characters, MySQL uses the *Turbo Boyer-Moore algorithm* to initialize the pattern for the string and then uses this pattern to perform the search more quickly.

A search using `col_name IS NULL` employs indexes if *`col_name`* is indexed.

Any index that does not span all `AND` levels in the `WHERE` clause is not used to optimize the query. In other words, to be able to use an index, a prefix of the index must be used in every `AND` group.

The following `WHERE` clauses use indexes:

```
... WHERE index_part1=1 AND index_part2=2 AND other_column=3
    /* index = 1 OR index = 2 */
... WHERE index=1 OR A=10 AND index=2
    /* optimized like "index_part1='hello'" */
... WHERE index_part1='hello' AND index_part3=5
    /* Can use index on index1 but not on index2 or index3 */
... WHERE index1=1 AND index2=2 OR index1=3 AND index3=3;
```

These `WHERE` clauses do *not* use indexes:

```
    /* index_part1 is not used */
... WHERE index_part2=1 AND index_part3=2

    /*  Index is not used in both parts of the WHERE clause  */
... WHERE index=1 OR A=10

    /* No index spans all rows  */
... WHERE index_part1=1 OR index_part2=10
```

Sometimes MySQL does not use an index, even if one is available. One circumstance under which this occurs is when the optimizer estimates that using the index would require MySQL to access a very large percentage of the rows in the table. (In this case, a table scan is likely to be much faster because it requires fewer seeks.) However, if such a query uses `LIMIT` to retrieve only some of the rows, MySQL uses an index anyway, because it can much more quickly find the few rows to return in the result.

Hash Index Characteristics

Hash indexes have somewhat different characteristics from those just discussed:

- They are used only for equality comparisons that use the = or <=> operators (but are *very* fast). They are not used for comparison operators such as < that find a range of values. Systems that rely on this type of single-value lookup are known as “key-value stores”; to use MySQL for such applications, use hash indexes wherever possible.

- The optimizer cannot use a hash index to speed up ORDER BY operations. (This type of index cannot be used to search for the next entry in order.)

- MySQL cannot determine approximately how many rows there are between two values (this is used by the range optimizer to decide which index to use). This may affect some queries if you change a MyISAM table to a hash-indexed MEMORY table.

- Only whole keys can be used to search for a row. (With a B-tree index, any leftmost prefix of the key can be used to find rows.)

8.4 Optimizing Database Structure

In your role as a database designer, look for the most efficient way to organize your schemas, tables, and columns. As when tuning application code, you minimize I/O, keep related items together, and plan ahead so that performance stays high as the data volume increases. Starting with an efficient database design makes it easier for team members to write high-performing application code, and makes the database likely to endure as applications evolve and are rewritten.

8.4.1 Optimizing Data Size

Design your tables to minimize their space on the disk. This can result in huge improvements by reducing the amount of data written to and read from disk. Smaller tables normally require less main memory while their contents are being actively processed during query execution. Any space reduction for table data also results in smaller indexes that can be processed faster.

MySQL supports many different storage engines (table types) and row formats. For each table, you can decide which storage and indexing method to use. Choosing the proper table format for your application can give you a big performance gain. See Chapter 14, *Storage Engines*.

You can get better performance for a table and minimize storage space by using the techniques listed here:

Table Columns

- Use the most efficient (smallest) data types possible. MySQL has many specialized types that save disk space and memory. For example, use the smaller integer types if possible to get smaller tables. MEDIUMINT is often a better choice than INT because a MEDIUMINT column uses 25% less space.

- Declare columns to be NOT NULL if possible. It makes SQL operations faster, by enabling better use of indexes and eliminating overhead for testing whether each value is NULL. You also save some storage space, one bit per column. If you really need NULL values in your tables, use them. Just avoid the default setting that allows NULL values in every column.

Row Format

- InnoDB tables use a compact storage format. In versions of MySQL earlier than 5.0.3, InnoDB rows contain some redundant information, such as the number of columns and the length of each column, even for fixed-size columns. By default, tables are created in the compact format (ROW_FORMAT=COMPACT). If you wish to downgrade to older versions of MySQL, you can request the old format with ROW_FORMAT=REDUNDANT.

The presence of the compact row format decreases row storage space by about 20% at the cost of increasing CPU use for some operations. If your workload is a typical one that is limited by cache hit rates and disk speed it is likely to be faster. If it is a rare case that is limited by CPU speed, it might be slower.

The compact InnoDB format also changes how CHAR columns containing UTF-8 data are stored. With ROW_FORMAT=REDUNDANT, a UTF-8 CHAR(*N*) occupies 3 × *N* bytes, given that the maximum length of a UTF-8 encoded character is three bytes. Many languages can be written primarily using single-byte UTF-8 characters, so a fixed storage length often wastes space. With ROW_FORMAT=COMPACT format, InnoDB allocates a variable amount of storage in the range from *N* to 3 × *N* bytes for these columns by stripping trailing spaces if necessary. The minimum storage length is kept as *N* bytes to facilitate in-place updates in typical cases.

- To minimize space even further by storing table data in compressed form, specify ROW_FORMAT=COMPRESSED when creating InnoDB tables, or run the myisampack command on an existing MyISAM table. (InnoDB tables compressed tables are readable and writable, while MyISAM compressed tables are read-only.)

- For MyISAM tables, if you do not have any variable-length columns (VARCHAR, TEXT, or BLOB columns), a fixed-size row format is used. This is faster but may waste some space. See Section 14.3.3, “MyISAM Table Storage Formats”. You can hint that you want to have fixed length rows even if you have VARCHAR columns with the CREATE TABLE option ROW_FORMAT=FIXED.

Indexes

- The primary index of a table should be as short as possible. This makes identification of each row easy and efficient. For InnoDB tables, the primary key columns are duplicated in each secondary index entry, so a short primary key saves considerable space if you have many secondary indexes.

- Create only the indexes that you need to improve query performance. Indexes are good for retrieval, but slow down insert and update operations. If you access a table mostly by searching on a combination of columns, create a single composite index on them rather than a separate index for each column. The first part of the index should be the column most used. If you *always* use many columns when selecting from the table, the first column in the index should be the one with the most duplicates, to obtain better compression of the index.

- If it is very likely that a long string column has a unique prefix on the first number of characters, it is better to index only this prefix, using MySQL's support for creating an index on the leftmost part of the column (see Section 13.1.11, “CREATE INDEX Syntax”). Shorter indexes are faster, not only because they require less disk space, but because they also give you more hits in the index cache, and thus fewer disk seeks. See Section 8.11.2, “Tuning Server Parameters”.

Joins

- In some circumstances, it can be beneficial to split into two a table that is scanned very often. This is especially true if it is a dynamic-format table and it is possible to use a smaller static format table that can be used to find the relevant rows when scanning the table.

- Declare columns with identical information in different tables with identical data types, to speed up joins based on the corresponding columns.

- Keep column names simple, so that you can use the same name across different tables and simplify join queries. For example, in a table named customer, use a column name of name instead of customer_name. To make your names portable to other SQL servers, consider keeping them shorter than 18 characters.

Normalization

- Normally, try to keep all data nonredundant (observing what is referred to in database theory as *third normal form*). Instead of repeating lengthy values such as names and addresses, assign them unique IDs, repeat these IDs as needed across multiple smaller tables, and join the tables in queries by referencing the IDs in the join clause.

- If speed is more important than disk space and the maintenance costs of keeping multiple copies of data, for example in a business intelligence scenario where you analyze all the data from large tables, you can relax the normalization rules, duplicating information or creating summary tables to gain more speed.

8.4.2 Optimizing MySQL Data Types

8.4.2.1 Optimizing for Numeric Data

- For unique IDs or other values that can be represented as either strings or numbers, prefer numeric columns to string columns. Since large numeric values can be stored in fewer bytes than the corresponding strings, it is faster and takes less memory to transfer and compare them.

- If you are using numeric data, it is faster in many cases to access information from a database (using a live connection) than to access a text file. Information in the database is likely to be stored in a more compact format than in the text file, so accessing it involves fewer disk accesses. You also save code in your application because you can avoid parsing the text file to find line and column boundaries.

8.4.2.2 Optimizing for Character and String Types

For character and string columns, follow these guidelines:

- Use binary collation order for fast comparison and sort operations, when you do not need language-specific collation features. You can use the BINARY operator to use binary collation within a particular query.

- When comparing values from different columns, declare those columns with the same character set and collation wherever possible, to avoid string conversions while running the query.

- For column values less than 8KB in size, use binary `VARCHAR` instead of `BLOB`. The `GROUP BY` and `ORDER BY` clauses can generate temporary tables, and these temporary tables can use the `MEMORY` storage engine if the original table does not contain any `BLOB` columns.

- If a table contains string columns such as name and address, but many queries do not retrieve those columns, consider splitting the string columns into a separate table and using join queries with a foreign key when necessary. When MySQL retrieves any value from a row, it reads a data block containing all the columns of that row (and possibly other adjacent rows). Keeping each row small, with only the most frequently used columns, allows more rows to fit in each data block. Such compact tables reduce disk I/O and memory usage for common queries.

- When you use a randomly generated value as a primary key in an `InnoDB` table, prefix it with an ascending value such as the current date and time if possible. When consecutive primary values are physically stored near each other, `InnoDB` can insert and retrieve them faster.

- See Section 8.4.2.1, "Optimizing for Numeric Data" for reasons why a numeric column is usually preferable to an equivalent string column.

8.4.2.3 Optimizing for BLOB Types

- When storing a large blob containing textual data, consider compressing it first. Do not use this technique when the entire table is compressed by `InnoDB` or `MyISAM`.
- For a table with several columns, to reduce memory requirements for queries that do not use the BLOB column, consider splitting the BLOB column into a separate table and referencing it with a join query when needed.
- Since the performance requirements to retrieve and display a BLOB value might be very different from other data types, you could put the BLOB-specific table on a different storage device or even a separate database instance. For example, to retrieve a BLOB might require a large sequential disk read that is better suited to a traditional hard drive than to an SSD device.
- See Section 8.4.2.2, "Optimizing for Character and String Types" for reasons why a binary `VARCHAR` column is sometimes preferable to an equivalent BLOB column.
- Rather than testing for equality against a very long text string, you can store a hash of the column value in a separate column, index that column, and test the hashed value in queries. (Use the `MD5()` or `CRC32()` function to produce the hash value.) Since hash functions can produce duplicate results for different inputs, you still include a clause `AND blob_column = long_string_value` in the query to guard against false matches; the performance benefit comes from the smaller, easily scanned index for the hashed values.

8.4.2.4 Using PROCEDURE ANALYSE

`ANALYSE([max_elements[,max_memory]])`

`ANALYSE()` examines the result from a query and returns an analysis of the results that suggests optimal data types for each column that may help reduce table sizes. To obtain this analysis, append `PROCEDURE ANALYSE` to the end of a `SELECT` statement:

```
SELECT ... FROM ... WHERE ... PROCEDURE ANALYSE([max_elements,[max_memory]])
```

For example:

```
SELECT col1, col2 FROM table1 PROCEDURE ANALYSE(10, 2000);
```

The results show some statistics for the values returned by the query, and propose an optimal data type for the columns. This can be helpful for checking your existing tables, or after importing new data. You may need to try different settings for the arguments so that `PROCEDURE ANALYSE()` does not suggest the `ENUM` data type when it is not appropriate.

The arguments are optional and are used as follows:

- *max_elements* (default 256) is the maximum number of distinct values that `ANALYSE()` notices per column. This is used by `ANALYSE()` to check whether the optimal data type should be of type `ENUM`; if there are more than *max_elements* distinct values, then `ENUM` is not a suggested type.
- *max_memory* (default 8192) is the maximum amount of memory that `ANALYSE()` should allocate per column while trying to find all distinct values.

8.4.3 Optimizing for Many Tables

Some techniques for keeping individual queries fast involve splitting data across many tables. When the number of tables runs into the thousands or even millions, the overhead of dealing with all these tables becomes a new performance consideration.

8.4.3.1 How MySQL Opens and Closes Tables

When you execute a `mysqladmin status` command, you should see something like this:

```
Uptime: 426 Running threads: 1 Questions: 11082
Reloads: 1 Open tables: 12
```

The `Open tables` value of 12 can be somewhat puzzling if you have only six tables.

MySQL is multi-threaded, so there may be many clients issuing queries for a given table simultaneously. To minimize the problem with multiple client sessions having different states on the same table, the table is opened independently by each concurrent session. This uses additional memory but normally increases performance. With `MyISAM` tables, one extra file descriptor is required for the data file for each client that has the table open. (By contrast, the index file descriptor is shared between all sessions.)

The `table_open_cache` and `max_connections` system variables affect the maximum number of files the server keeps open. If you increase one or both of these values, you may run up against a limit imposed by your operating system on the per-process number of open file descriptors. Many operating systems permit you to increase the open-files limit, although the method varies widely from system to system. Consult your operating system documentation to determine whether it is possible to increase the limit and how to do so.

`table_open_cache` is related to `max_connections`. For example, for 200 concurrent running connections, specify a table cache size of at least `200 * N`, where `N` is the maximum number of tables per join in any of the queries which you execute. You must also reserve some extra file descriptors for temporary tables and files.

Make sure that your operating system can handle the number of open file descriptors implied by the `table_open_cache` setting. If `table_open_cache` is set too high, MySQL may run out of file descriptors and refuse connections, fail to perform queries, and be very unreliable. You also have to take into account that the `MyISAM` storage engine needs two file descriptors for each unique open table. You can increase the number of file descriptors available to MySQL using the `--open-files-limit` startup option to `mysqld`. See Section C.5.2.18, “`'File' Not Found` and Similar Errors”.

The cache of open tables is kept at a level of `table_open_cache` entries. The server autosizes the cache size at startup. To set the size explicitly, set the `table_open_cache` system variable at startup. Note that MySQL may temporarily open more tables than this to execute queries.

MySQL closes an unused table and removes it from the table cache under the following circumstances:

- When the cache is full and a thread tries to open a table that is not in the cache.
- When the cache contains more than `table_open_cache` entries and a table in the cache is no longer being used by any threads.
- When a table flushing operation occurs. This happens when someone issues a `FLUSH TABLES` statement or executes a `mysqladmin flush-tables` or `mysqladmin refresh` command.

When the table cache fills up, the server uses the following procedure to locate a cache entry to use:

- Tables that are not currently in use are released, beginning with the table least recently used.
- If a new table needs to be opened, but the cache is full and no tables can be released, the cache is temporarily extended as necessary. When the cache is in a temporarily extended state and a table goes from a used to unused state, the table is closed and released from the cache.

A MyISAM table is opened for each concurrent access. This means the table needs to be opened twice if two threads access the same table or if a thread accesses the table twice in the same query (for example, by joining the table to itself). Each concurrent open requires an entry in the table cache. The first open of any MyISAM table takes two file descriptors: one for the data file and one for the index file. Each additional use of the table takes only one file descriptor for the data file. The index file descriptor is shared among all threads.

If you are opening a table with the HANDLER *tbl_name* OPEN statement, a dedicated table object is allocated for the thread. This table object is not shared by other threads and is not closed until the thread calls HANDLER *tbl_name* CLOSE or the thread terminates. When this happens, the table is put back in the table cache (if the cache is not full). See Section 13.2.4, “HANDLER Syntax”.

You can determine whether your table cache is too small by checking the mysqld status variable Opened_tables, which indicates the number of table-opening operations since the server started:

```
mysql> SHOW GLOBAL STATUS LIKE 'Opened_tables';
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| Opened_tables | 2741  |
+---------------+-------+
```

If the value is very large or increases rapidly, even when you have not issued many FLUSH TABLES statements, increase the table cache size. See Section 5.1.4, “Server System Variables”, and Section 5.1.6, “Server Status Variables”.

8.4.3.2 Disadvantages of Creating Many Tables in the Same Database

If you have many MyISAM tables in the same database directory, open, close, and create operations are slow. If you execute SELECT statements on many different tables, there is a little overhead when the table cache is full, because for every table that has to be opened, another must be closed. You can reduce this overhead by increasing the number of entries permitted in the table cache.

8.4.4 How MySQL Uses Internal Temporary Tables

In some cases, the server creates internal temporary tables while processing queries. Such a table can be held in memory and processed by the MEMORY storage engine, or stored on disk and processed by the MyISAM storage engine. The server may create a temporary table initially as an in-memory table, then convert it to an on-disk table if it becomes too large. Users have no direct control over when the server creates an internal temporary table or which storage engine the server uses to manage it.

Temporary tables can be created under conditions such as these:

- UNION queries use temporary tables, with some exceptions described later.
- Some views require temporary tables, such those evaluated using the TEMPTABLE algorithm, or that use UNION or aggregation.
- If there is an ORDER BY clause and a different GROUP BY clause, or if the ORDER BY or GROUP BY contains columns from tables other than the first table in the join queue, a temporary table is created.
- DISTINCT combined with ORDER BY may require a temporary table.
- If you use the SQL_SMALL_RESULT option, MySQL uses an in-memory temporary table, unless the query also contains elements (described later) that require on-disk storage.
- Derived tables (subqueries in the FROM clause).

- Tables created for subquery or semi-join materialization.

To determine whether a query requires a temporary table, use EXPLAIN and check the Extra column to see whether it says Using temporary (see Section 8.8.1, "Optimizing Queries with EXPLAIN"). EXPLAIN will not necessarily say Using temporary for derived or materialized temporary tables.

If an internal temporary table is created initially as an in-memory table but becomes too large, MySQL automatically converts it to an on-disk table. The maximum size for in-memory temporary tables is the minimum of the tmp_table_size and max_heap_table_size values. This differs from MEMORY tables explicitly created with CREATE TABLE: For such tables, only the max_heap_table_size system variable determines how large the table is permitted to grow and there is no conversion to on-disk format.

When the server creates an internal temporary table (either in memory or on disk), it increments the Created_tmp_tables status variable. If the server creates the table on disk (either initially or by converting an in-memory table) it increments the Created_tmp_disk_tables status variable.

Some conditions prevent the use of an in-memory temporary table, in which case the server uses an on-disk table instead:

- Presence of a BLOB or TEXT column in the table
- Presence of any string column in a GROUP BY or DISTINCT clause larger than 512 bytes for binary strings or 512 characters for nonbinary strings. (Before MySQL 5.7.3, the limit is 512 bytes regardless of string type.)
- Presence of any string column with a maximum length larger than 512 (bytes for binary strings, characters for nonbinary strings) in the SELECT list, if UNION or UNION ALL is used

As of MySQL 5.7.3, the server does not use a temporary table for UNION statements that meet certain qualifications. Instead, it retains from temporary table creation only the data structures necessary to perform result column typecasting. The table is not fully instantiated and no rows are written to or read from it; rows are sent directly to the client. As a result, The result is reduced memory and disk requirements, and smaller delay before the first row is sent to the client because the server need not wait until the last query block is executed. EXPLAIN and optimizer trace output will change: The UNION RESULT query block will not be present because that block is the part that reads from the temporary table.

The conditions that qualify a UNION for evaluation without a temporary table are:

- The union is UNION ALL, not UNION or UNION DISTINCT.
- There is no global ORDER BY clause.
- The union is not the top-level query block of an {INSERT | REPLACE} ... SELECT ... statement.

8.5 Optimizing for InnoDB Tables

InnoDB is the storage engine that MySQL customers typically use in production databases where reliability and concurrency are important. Because InnoDB is the default storage engine in MySQL 5.5 and higher, you can expect to see InnoDB tables more often than before. This section explains how to optimize database operations for InnoDB tables.

8.5.1 Optimizing Storage Layout for InnoDB Tables

- Once your data reaches a stable size, or a growing table has increased by tens or some hundreds of megabytes, consider using the OPTIMIZE TABLE statement to reorganize the table and compact any wasted space. The reorganized tables require less disk I/O to perform full table scans. This is a straightforward technique that can improve performance when other techniques such as improving index usage or tuning application code are not practical.

OPTIMIZE TABLE copies the data part of the table and rebuilds the indexes. The benefits come from improved packing of data within indexes, and reduced fragmentation within the tablespaces and on disk. The benefits vary depending on the data in each table. You may find that there are significant gains for some and not for others, or that the gains decrease over time until you next optimize the table. This operation can be slow if the table is large or if the indexes being rebuilt don't fit into the buffer pool. The first run after adding a lot of data to a table is often much slower than later runs.

- In InnoDB, having a long PRIMARY KEY (either a single column with a lengthy value, or several columns that form a long composite value) wastes a lot of disk space. The primary key value for a row is duplicated in all the secondary index records that point to the same row. (See Section 14.2.2.14, “InnoDB Table and Index Structures”.) Create an AUTO_INCREMENT column as the primary key if your primary key is long, or index a prefix of a long VARCHAR column instead of the entire column.

- Use the VARCHAR data type instead of CHAR to store variable-length strings or for columns with many NULL values. A CHAR(*N*) column always takes *N* characters to store data, even if the string is shorter or its value is NULL. Smaller tables fit better in the buffer pool and reduce disk I/O.

  When using COMPACT row format (the default InnoDB format in MySQL 5.7) and variable-length character sets, such as utf8 or sjis, CHAR(*N*) columns occupy a variable amount of space, but still at least *N* bytes.

- For tables that are big, or contain lots of repetitive text or numeric data, consider using COMPRESSED row format. Less disk I/O is required to bring data into the buffer pool, or to perform full table scans. Before making a permanent decision, measure the amount of compression you can achieve by using COMPRESSED versus COMPACT row format.

8.5.2 Optimizing InnoDB Transaction Management

To optimize InnoDB transaction processing, find the ideal balance between the performance overhead of transactional features and the workload of your server. For example, an application might encounter performance issues if it commits thousands of times per second, and different performance issues if it commits only every 2-3 hours.

- The default MySQL setting AUTOCOMMIT=1 can impose performance limitations on a busy database server. Where practical, wrap several related DML operations into a single transaction, by issuing SET AUTOCOMMIT=0 or a START TRANSACTION statement, followed by a COMMIT statement after making all the changes.

  InnoDB must flush the log to disk at each transaction commit if that transaction made modifications to the database. When each change is followed by a commit (as with the default autocommit setting), the I/O throughput of the storage device puts a cap on the number of potential operations per second.

- Alternatively, for transactions that consist only of a single SELECT statement, turning on AUTOCOMMIT helps InnoDB to recognize read-only transactions and optimize them. See Optimizations for Read-Only Transactions for requirements.

- Avoid performing rollbacks after inserting, updating, or deleting huge numbers of rows. If a big transaction is slowing down server performance, rolling it back can make the problem worse, potentially taking several times as long to perform as the original DML operations. Killing the database process does not help, because the rollback starts again on server startup.

  To minimize the chance of this issue occurring: increase the size of the buffer pool so that all the DML changes can be cached rather than immediately written to disk; set innodb_change_buffering=all so that update and delete operations are buffered in addition to inserts; and consider issuing COMMIT statements periodically during the big DML operation, possibly breaking a single delete or update into multiple statements that operate on smaller numbers of rows.

To get rid of a runaway rollback once it occurs, increase the buffer pool so that the rollback becomes CPU-bound and runs fast, or kill the server and restart with innodb_force_recovery=3, as explained in Section 14.2.14.1, “The InnoDB Recovery Process”.

This issue is expected to be less prominent in MySQL 5.5 and up, or in MySQL 5.1 with the InnoDB Plugin, because the default setting innodb_change_buffering=all allows update and delete operations to be cached in memory, making them faster to perform in the first place, and also faster to roll back if needed. Make sure to use this parameter setting on servers that process long-running transactions with many inserts, updates, or deletes.

- If you can afford the loss of some of the latest committed transactions if a crash occurs, you can set the innodb_flush_log_at_trx_commit parameter to 0. InnoDB tries to flush the log once per second anyway, although the flush is not guaranteed. Also, set the value of innodb_support_xa to 0, which will reduce the number of disk flushes due to synchronizing on disk data and the binary log.

- When rows are modified or deleted, the rows and associated undo logs are not physically removed immediately, or even immediately after the transaction commits. The old data is preserved until transactions that started earlier or concurrently are finished, so that those transactions can access the previous state of modified or deleted rows. Thus, a long-running transaction can prevent InnoDB from purging data that was changed by a different transaction.

- When rows are modified or deleted within a long-running transaction, other transactions using the READ COMMITTED and REPEATABLE READ isolation levels have to do more work to reconstruct the older data if they read those same rows.

- When a long-running transaction modifies a table, queries against that table from other transactions do not make use of the covering index technique. Queries that normally could retrieve all the result columns from a secondary index, instead look up the appropriate values from the table data.

  If secondary index pages are found to have a PAGE_MAX_TRX_ID that is too new, or if records in the secondary index are delete-marked, InnoDB may need to look up records using a clustered index.

8.5.3 Optimizing InnoDB Logging

- Make your log files big, even as big as the buffer pool. When InnoDB has written the log files full, it must write the modified contents of the buffer pool to disk in a checkpoint. Small log files cause many unnecessary disk writes. Although historically big log files caused lengthy recovery times, recovery is now much faster and you can confidently use large log files.

- Make the log buffer quite large as well (on the order of 8MB).

8.5.4 Bulk Data Loading for InnoDB Tables

These performance tips supplement the general guidelines for fast inserts in Section 8.2.2.1, “Speed of INSERT Statements”.

- When importing data into InnoDB, turn off autocommit mode, because it performs a log flush to disk for every insert. To disable autocommit during your import operation, surround it with SET autocommit and COMMIT statements:

```
SET autocommit=0;
... SQL import statements ...
```

```
COMMIT;
```

The mysqldump option --opt creates dump files that are fast to import into an InnoDB table, even without wrapping them with the SET autocommit and COMMIT statements.

- If you have UNIQUE constraints on secondary keys, you can speed up table imports by temporarily turning off the uniqueness checks during the import session:

  ```
  SET unique_checks=0;
  ... SQL import statements ...
  SET unique_checks=1;
  ```

  For big tables, this saves a lot of disk I/O because InnoDB can use its insert buffer to write secondary index records in a batch. Be certain that the data contains no duplicate keys.

- If you have FOREIGN KEY constraints in your tables, you can speed up table imports by turning off the foreign key checks for the duration of the import session:

  ```
  SET foreign_key_checks=0;
  ... SQL import statements ...
  SET foreign_key_checks=1;
  ```

  For big tables, this can save a lot of disk I/O.

- Use the multiple-row INSERT syntax to reduce communication overhead between the client and the server if you need to insert many rows:

  ```
  INSERT INTO yourtable VALUES (1,2), (5,5), ...;
  ```

  This tip is valid for inserts into any table, not just InnoDB tables.

- When doing bulk inserts into tables with auto-increment columns, set innodb_autoinc_lock_mode to 2 instead of the default value 1. See Configurable InnoDB Auto-Increment Locking for details.

- For optimal performance when loading data into an InnoDB FULLTEXT index, follow this set of steps:

  - Define a column FTS_DOC_ID at table creation time, of type BIGINT UNSIGNED NOT NULL, with a unique index named FTS_DOC_ID_INDEX. For example:

    ```
    CREATE TABLE t1 (
    FTS_DOC_ID BIGINT unsigned NOT NULL AUTO_INCREMENT,
    title varchar(255) NOT NULL DEFAULT ”,
    text mediumtext NOT NULL,
    PRIMARY KEY (`FTS_DOC_ID`)
    ) ENGINE=InnoDB DEFAULT CHARSET=latin1;
    CREATE UNIQUE INDEX FTS_DOC_ID_INDEX on t1(FTS_DOC_ID);
    ```

  - Load the data into the table.

  - Create the FULLTEXT index after the data is loaded.

Note

When adding FTS_DOC_ID column at table creation time, ensure that the FTS_DOC_ID column is updated when the FULLTEXT indexed column is updated, as the FTS_DOC_ID must increase monotonically with each INSERT or UPDATE. If you choose not to add the FTS_DOC_ID at table creation time and have InnoDB manage DOC IDs for you, InnoDB will add the FTS_DOC_ID as a hidden column with the next CREATE FULLTEXT INDEX call. This approach, however, requires a table rebuild which will impact performance.

8.5.5 Optimizing InnoDB Queries

To tune queries for InnoDB tables, create an appropriate set of indexes on each table. See Section 8.3.1, “How MySQL Uses Indexes” for details. Follow these guidelines for InnoDB indexes:

- Because each InnoDB table has a primary key (whether you request one or not), specify a set of primary key columns for each table, columns that are used in the most important and time-critical queries.

- Do not specify too many or too long columns in the primary key, because these column values are duplicated in each secondary index. When an index contains unnecessary data, the I/O to read this data and memory to cache it reduce the performance and scalability of the server.

- Do not create a separate secondary index for each column, because each query can only make use of one index. Indexes on rarely tested columns or columns with only a few different values might not be helpful for any queries. If you have many queries for the same table, testing different combinations of columns, try to create a small number of concatenated indexes rather than a large number of single-column indexes. If an index contains all the columns needed for the result set (known as a covering index), the query might be able to avoid reading the table data at all.

- If an indexed column cannot contain any NULL values, declare it as NOT NULL when you create the table. The optimizer can better determine which index is most effective to use for a query, when it knows whether each column contains NULL values or not.

- You can optimize single-query transactions for InnoDB tables, using the technique in Optimizations for Read-Only Transactions.

- If you often have recurring queries for tables that are not updated frequently, enable the query cache:

```
[mysqld]
query_cache_type = 1
query_cache_size = 10M
```

8.5.6 Optimizing InnoDB DDL Operations

- For DDL operations on tables and indexes (CREATE, ALTER, and DROP statements), the most significant aspect for InnoDB tables is that creating and dropping secondary indexes is much faster in MySQL 5.5 and higher, than in MySQL 5.1 and before. See InnoDB Fast Index Creation for details.

- “Fast index creation” makes it faster in some cases to drop an index before loading data into a table, then re-create the index after loading the data.

- Use TRUNCATE TABLE to empty a table, not DELETE FROM *tbl_name*. Foreign key constraints can make a TRUNCATE statement work like a regular DELETE statement, in which case a sequence of commands like DROP TABLE and CREATE TABLE might be fastest.

- Because the primary key is integral to the storage layout of each InnoDB table, and changing the definition of the primary key involves reorganizing the whole table, always set up the primary key as part of the CREATE TABLE statement, and plan ahead so that you do not need to ALTER or DROP the primary key afterward.

8.5.7 Optimizing InnoDB Disk I/O

If you follow the best practices for database design and the tuning techniques for SQL operations, but your database is still slowed by heavy disk I/O activity, explore these low-level techniques related to disk I/O. If the Unix top tool or the Windows Task Manager shows that the CPU usage percentage with your workload is less than 70%, your workload is probably disk-bound.

- When table data is cached in the InnoDB buffer pool, it can be processed over and over by queries without requiring any disk I/O. Specify the size of the buffer pool with the innodb_buffer_pool_size option. This memory area is important enough that busy databases often specify a size approximately 80% of the amount of physical memory. For more information, see Section 8.9.1, “The InnoDB Buffer Pool”.

- In some versions of GNU/Linux and Unix, flushing files to disk with the Unix fsync() call (which InnoDB uses by default) and similar methods is surprisingly slow. If database write performance is an issue, conduct benchmarks with the innodb_flush_method parameter set to O_DSYNC.

- When using the InnoDB storage engine on Solaris 10 for x86_64 architecture (AMD Opteron), use direct I/O for InnoDB-related files, to avoid degradation of InnoDB performance. To use direct I/O for an entire UFS file system used for storing InnoDB-related files, mount it with the forcedirectio option; see mount_ufs(1M). (The default on Solaris 10/x86_64 is *not* to use this option.) To apply direct I/O only to InnoDB file operations rather than the whole file system, set innodb_flush_method = O_DIRECT. With this setting, InnoDB calls directio() instead of fcntl() for I/O to data files (not for I/O to log files).

- When using the InnoDB storage engine with a large innodb_buffer_pool_size value on any release of Solaris 2.6 and up and any platform (sparc/x86/x64/amd64), conduct benchmarks with InnoDB data files and log files on raw devices or on a separate direct I/O UFS file system, using the forcedirectio mount option as described earlier. (It is necessary to use the mount option rather than setting innodb_flush_method if you want direct I/O for the log files.) Users of the Veritas file system VxFS should use the convosync=direct mount option.

  Do not place other MySQL data files, such as those for MyISAM tables, on a direct I/O file system. Executables or libraries *must not* be placed on a direct I/O file system.

- If you have additional storage devices available to set up a RAID configuration or symbolic links to different disks, Section 8.11.3, “Optimizing Disk I/O” for additional low-level I/O tips.

- If throughput drops periodically because of InnoDB checkpoint operations, consider increasing the value of the innodb_io_capacity configuration option. Higher values cause more frequent flushing, avoiding the backlog of work that can cause dips in throughput.

- If the system is not falling behind with InnoDB flushing operations, consider lowering the value of the innodb_io_capacity configuration option. Typically, you keep this option value as low as practical, but not so low that it causes periodic drops in throughput as mentioned in the preceding bullet. In a typical scenario where you could lower the option value, you might see a combination like this in the output from SHOW ENGINE INNODB STATUS:

  - History list length low, below a few thousand.

  - Insert buffer merges close to rows inserted.

  - Modified pages in buffer pool consistently well below innodb_max_dirty_pages_pct of the buffer pool. (Measure at a time when the server is not doing bulk inserts; it is normal during bulk inserts for the modified pages percentage to rise significantly.)

  - Log sequence number - Last checkpoint is at less than 7/8 or ideally less than 6/8 of the total size of the InnoDB log files.

- Other InnoDB configuration options to consider when tuning I/O-bound workloads include innodb_adaptive_flushing, innodb_change_buffer_max_size, innodb_change_buffering, innodb_flush_neighbors, innodb_log_buffer_size, innodb_log_file_size, innodb_lru_scan_depth, innodb_max_dirty_pages_pct, innodb_max_purge_lag, innodb_open_files, innodb_page_size, innodb_random_read_ahead, innodb_read_ahead_threshold, innodb_read_io_threads, innodb_rollback_segments, innodb_write_io_threads, and sync_binlog.

8.5.8 Optimizing InnoDB Configuration Variables

Different settings work best for servers with light, predictable loads, versus servers that are running near full capacity all the time, or that experience spikes of high activity.

Because the InnoDB storage engine performs many of its optimizations automatically, many performance-tuning tasks involve monitoring to ensure that the database is performing well, and changing configuration options when performance drops. See Integration with the MySQL Performance Schema for information about detailed InnoDB performance monitoring.

For information about the most important and most recent InnoDB performance features, see Section 14.2.12.2, “InnoDB Performance and Scalability Enhancements”. Even if you have used InnoDB tables in prior versions, these features might be new to you, because they are from the “InnoDB Plugin”. The Plugin co-existed alongside the built-in InnoDB in MySQL 5.1, and becomes the default storage engine in MySQL 5.5 and higher.

The main configuration steps you can perform include:

- Enabling InnoDB to use high-performance memory allocators on systems that include them. See Using Operating System Memory Allocators.

- Controlling the types of DML operations for which InnoDB buffers the changed data, to avoid frequent small disk writes. See Controlling InnoDB Change Buffering. Because the default is to buffer all types of DML operations, only change this setting if you need to reduce the amount of buffering.

- Turning the adaptive hash indexing feature on and off. See Controlling Adaptive Hash Indexing. You might change this setting during periods of unusual activity, then restore it to its original setting.

- Setting a limit on the number of concurrent threads that InnoDB processes, if context switching is a bottleneck. See Changes Regarding Thread Concurrency.

- Controlling the amount of prefetching that InnoDB does with its read-ahead operations. When the system has unused I/O capacity, more read-ahead can improve the performance of queries. Too much read-ahead can cause periodic drops in performance on a heavily loaded system. See Changes in the Read-Ahead Algorithm.

- Increasing the number of background threads for read or write operations, if you have a high-end I/O subsystem that is not fully utilized by the default values. See Multiple Background InnoDB I/O Threads.

- Controlling how much I/O InnoDB performs in the background. See Controlling the InnoDB Master Thread I/O Rate. The amount of background I/O is higher than in MySQL 5.1, so you might scale back this setting if you observe periodic drops in performance.

- Controlling the algorithm that determines when InnoDB performs certain types of background writes. See Controlling the Flushing Rate of Dirty Pages from the InnoDB Buffer Pool. The algorithm works for some types of workloads but not others, so might turn off this setting if you observe periodic drops in performance.

- Taking advantage of multicore processors and their cache memory configuration, to minimize delays in context switching. See Control of Spin Lock Polling.
- Preventing one-time operations such as table scans from interfering with the frequently accessed data stored in the InnoDB buffer cache. See Making the Buffer Pool Scan Resistant.
- Adjusting your log files to a size that makes sense for reliability and crash recovery. Historically, people have kept their InnoDB log files small to avoid long startup times after a crash. Internal improvements in InnoDB make startup much faster, so the log file size is not such a performance factor anymore. If your log files are artificially small, increasing the size can help performance by reducing the I/O that occurs as redo log records are recycled.
- Configuring the size and number of instances for the InnoDB buffer pool, especially important for systems with multi-gigabyte buffer pools. See Improvements to Performance from Multiple Buffer Pools.
- Increasing the maximum number of concurrent transactions, which dramatically improves scalability for the busiest databases. See Better Scalability with Multiple Rollback Segments. Although this feature does not require any action during day-to-day operation, you must perform a slow shutdown during or after upgrading the database to MySQL 5.5 to enable the higher limit.
- Moving purge operations (a type of garbage collection) into a background thread. See Better Scalability with Improved Purge Scheduling. To effectively measure the results of this setting, tune the other I/O-related and thread-related configuration settings first.
- Reducing the amount of switching that InnoDB does between concurrent threads, so that SQL operations on a busy server do not queue up and form a “traffic jam”. Set a value for the `innodb_thread_concurrency` option, up to approximately 32 for a high-powered modern system. Increase the value for the `innodb_concurrency_tickets` option, typically to 5000 or so, This combination of options sets a cap on the number of threads that InnoDB processes at any one time, and allows each thread to do substantial work before being swapped out, so that the number of waiting threads stays low and operations can complete without excessive context switching.

8.5.9 Optimizing InnoDB for Systems with Many Tables

- InnoDB computes index cardinality values for a table the first time that table is accessed after startup, instead of storing such values in the table. This step can take significant time on systems that partition the data into many tables. Since this overhead only applies to the initial table open operation, to “warm up” a table for later use, access it immediately after startup by issuing a statement such as `SELECT 1 FROM *tbl_name* LIMIT 1`.

8.6 Optimizing for MyISAM Tables

The MyISAM storage engine performs best with read-mostly data or with low-concurrency operations, because table locks limit the ability to perform simultaneous updates. In MySQL 5.7, InnoDB is the default storage engine rather than MyISAM.

8.6.1 Optimizing MyISAM Queries

Some general tips for speeding up queries on MyISAM tables:

- To help MySQL better optimize queries, use `ANALYZE TABLE` or run `myisamchk --analyze` on a table after it has been loaded with data. This updates a value for each index part that indicates the average number of rows that have the same value. (For unique indexes, this is always 1.) MySQL uses this to decide which index to choose when you join two tables based on a nonconstant expression. You can check the result from the table analysis by using SHOW INDEX FROM *tbl_name* and examining the Cardinality value. myisamchk --description --verbose shows index distribution information.

- To sort an index and data according to an index, use myisamchk --sort-index --sort-records=1 (assuming that you want to sort on index 1). This is a good way to make queries faster if you have a unique index from which you want to read all rows in order according to the index. The first time you sort a large table this way, it may take a long time.
- Try to avoid complex SELECT queries on MyISAM tables that are updated frequently, to avoid problems with table locking that occur due to contention between readers and writers.
- MyISAM supports concurrent inserts: If a table has no free blocks in the middle of the data file, you can INSERT new rows into it at the same time that other threads are reading from the table. If it is important to be able to do this, consider using the table in ways that avoid deleting rows. Another possibility is to run OPTIMIZE TABLE to defragment the table after you have deleted a lot of rows from it. This behavior is altered by setting the concurrent_insert variable. You can force new rows to be appended (and therefore permit concurrent inserts), even in tables that have deleted rows. See Section 8.10.3, “Concurrent Inserts”.
- For MyISAM tables that change frequently, try to avoid all variable-length columns (VARCHAR, BLOB, and TEXT). The table uses dynamic row format if it includes even a single variable-length column. See Chapter 14, *Storage Engines*.
- It is normally not useful to split a table into different tables just because the rows become large. In accessing a row, the biggest performance hit is the disk seek needed to find the first byte of the row. After finding the data, most modern disks can read the entire row fast enough for most applications. The only cases where splitting up a table makes an appreciable difference is if it is a MyISAM table using dynamic row format that you can change to a fixed row size, or if you very often need to scan the table but do not need most of the columns. See Chapter 14, *Storage Engines*.
- Use ALTER TABLE ... ORDER BY *expr1*, *expr2*, ... if you usually retrieve rows in *expr1*, *expr2*, ... order. By using this option after extensive changes to the table, you may be able to get higher performance.
- If you often need to calculate results such as counts based on information from a lot of rows, it may be preferable to introduce a new table and update the counter in real time. An update of the following form is very fast:

  ```
  UPDATE tbl_name SET count_col=count_col+1 WHERE key_col=constant;
  ```

  This is very important when you use MySQL storage engines such as MyISAM that has only table-level locking (multiple readers with single writers). This also gives better performance with most database systems, because the row locking manager in this case has less to do.
- Use OPTIMIZE TABLE periodically to avoid fragmentation with dynamic-format MyISAM tables. See Section 14.3.3, “MyISAM Table Storage Formats”.
- Declaring a MyISAM table with the DELAY_KEY_WRITE=1 table option makes index updates faster because they are not flushed to disk until the table is closed. The downside is that if something kills the server while such a table is open, you must ensure that the table is okay by running the server with the --myisam-recover-options option, or by running myisamchk before restarting the server. (However, even in this case, you should not lose anything by using DELAY_KEY_WRITE, because the key information can always be generated from the data rows.)
- Strings are automatically prefix- and end-space compressed in MyISAM indexes. See Section 13.1.11, “CREATE INDEX Syntax”.

- You can increase performance by caching queries or answers in your application and then executing many inserts or updates together. Locking the table during this operation ensures that the index cache is only flushed once after all updates. You can also take advantage of MySQL's query cache to achieve similar results; see Section 8.9.3, “The MySQL Query Cache”.

8.6.2 Bulk Data Loading for MyISAM Tables

These performance tips supplement the general guidelines for fast inserts in Section 8.2.2.1, “Speed of INSERT Statements”.

- For a MyISAM table, you can use concurrent inserts to add rows at the same time that SELECT statements are running, if there are no deleted rows in middle of the data file. See Section 8.10.3, “Concurrent Inserts”.

- With some extra work, it is possible to make LOAD DATA INFILE run even faster for a MyISAM table when the table has many indexes. Use the following procedure:

  1. Execute a FLUSH TABLES statement or a mysqladmin flush-tables command.

  2. Use myisamchk --keys-used=0 -rq *path/to/db/tbl_name* to remove all use of indexes for the table.

  3. Insert data into the table with LOAD DATA INFILE. This does not update any indexes and therefore is very fast.

  4. If you intend only to read from the table in the future, use myisampack to compress it. See Section 14.3.3.3, “Compressed Table Characteristics”.

  5. Re-create the indexes with myisamchk -rq *path/to/db/tbl_name*. This creates the index tree in memory before writing it to disk, which is much faster that updating the index during LOAD DATA INFILE because it avoids lots of disk seeks. The resulting index tree is also perfectly balanced.

  6. Execute a FLUSH TABLES statement or a mysqladmin flush-tables command.

  LOAD DATA INFILE performs the preceding optimization automatically if the MyISAM table into which you insert data is empty. The main difference between automatic optimization and using the procedure explicitly is that you can let myisamchk allocate much more temporary memory for the index creation than you might want the server to allocate for index re-creation when it executes the LOAD DATA INFILE statement.

  You can also disable or enable the nonunique indexes for a MyISAM table by using the following statements rather than myisamchk. If you use these statements, you can skip the FLUSH TABLE operations:

  ```
  ALTER TABLE tbl_name DISABLE KEYS;
  ALTER TABLE tbl_name ENABLE KEYS;
  ```

- To speed up INSERT operations that are performed with multiple statements for nontransactional tables, lock your tables:

  ```
  LOCK TABLES a WRITE;
  INSERT INTO a VALUES (1,23),(2,34),(4,33);
  INSERT INTO a VALUES (8,26),(6,29);
  ...
  UNLOCK TABLES;
  ```

This benefits performance because the index buffer is flushed to disk only once, after all INSERT statements have completed. Normally, there would be as many index buffer flushes as there are INSERT statements. Explicit locking statements are not needed if you can insert all rows with a single INSERT.

Locking also lowers the total time for multiple-connection tests, although the maximum wait time for individual connections might go up because they wait for locks. Suppose that five clients attempt to perform inserts simultaneously as follows:

- Connection 1 does 1000 inserts
- Connections 2, 3, and 4 do 1 insert
- Connection 5 does 1000 inserts

If you do not use locking, connections 2, 3, and 4 finish before 1 and 5. If you use locking, connections 2, 3, and 4 probably do not finish before 1 or 5, but the total time should be about 40% faster.

INSERT, UPDATE, and DELETE operations are very fast in MySQL, but you can obtain better overall performance by adding locks around everything that does more than about five successive inserts or updates. If you do very many successive inserts, you could do a LOCK TABLES followed by an UNLOCK TABLES once in a while (each 1,000 rows or so) to permit other threads to access table. This would still result in a nice performance gain.

INSERT is still much slower for loading data than LOAD DATA INFILE, even when using the strategies just outlined.

- To increase performance for MyISAM tables, for both LOAD DATA INFILE and INSERT, enlarge the key cache by increasing the key_buffer_size system variable. See Section 8.11.2, “Tuning Server Parameters”.

8.6.3 Speed of REPAIR TABLE Statements

REPAIR TABLE for MyISAM tables is similar to using myisamchk for repair operations, and some of the same performance optimizations apply:

- myisamchk has variables that control memory allocation. You may be able to its improve performance by setting these variables, as described in Section 4.6.3.6, “myisamchk Memory Usage”.
- For REPAIR TABLE, the same principle applies, but because the repair is done by the server, you set server system variables instead of myisamchk variables. Also, in addition to setting memory-allocation variables, increasing the myisam_max_sort_file_size system variable increases the likelihood that the repair will use the faster filesort method and avoid the slower repair by key cache method. Set the variable to the maximum file size for your system, after checking to be sure that there is enough free space to hold a copy of the table files. The free space must be available in the file system containing the original table files.

Suppose that a myisamchk table-repair operation is done using the following options to set its memory-allocation variables:

```
--key_buffer_size=128M --myisam_sort_buffer_size=256M
--read_buffer_size=64M --write_buffer_size=64M
```

Some of those myisamchk variables correspond to server system variables:

myisamchk Variable	System Variable
key_buffer_size	key_buffer_size

myisamchk Variable	System Variable
myisam_sort_buffer_size	myisam_sort_buffer_size
read_buffer_size	read_buffer_size
write_buffer_size	none

Each of the server system variables can be set at runtime, and some of them (myisam_sort_buffer_size, read_buffer_size) have a session value in addition to a global value. Setting a session value limits the effect of the change to your current session and does not affect other users. Changing a global-only variable (key_buffer_size, myisam_max_sort_file_size) affects other users as well. For key_buffer_size, you must take into account that the buffer is shared with those users. For example, if you set the myisamchk key_buffer_size variable to 128MB, you could set the corresponding key_buffer_size system variable larger than that (if it is not already set larger), to permit key buffer use by activity in other sessions. However, changing the global key buffer size invalidates the buffer, causing increased disk I/O and slowdown for other sessions. An alternative that avoids this problem is to use a separate key cache, assign to it the indexes from the table to be repaired, and deallocate it when the repair is complete. See Section 8.9.2.2, “Multiple Key Caches”.

Based on the preceding remarks, a REPAIR TABLE operation can be done as follows to use settings similar to the myisamchk command. Here a separate 128MB key buffer is allocated and the file system is assumed to permit a file size of at least 100GB.

```
SET SESSION myisam_sort_buffer_size = 256*1024*1024;
SET SESSION read_buffer_size = 64*1024*1024;
SET GLOBAL myisam_max_sort_file_size = 100*1024*1024*1024;
SET GLOBAL repair_cache.key_buffer_size = 128*1024*1024;
CACHE INDEX tbl_name IN repair_cache;
LOAD INDEX INTO CACHE tbl_name;
REPAIR TABLE tbl_name ;
SET GLOBAL repair_cache.key_buffer_size = 0;
```

If you intend to change a global variable but want to do so only for the duration of a REPAIR TABLE operation to minimally affect other users, save its value in a user variable and restore it afterward. For example:

```
SET @old_myisam_sort_buffer_size = @@global.myisam_max_sort_file_size;
SET GLOBAL myisam_max_sort_file_size = 100*1024*1024*1024;
REPAIR TABLE tbl_name ;
SET GLOBAL myisam_max_sort_file_size = @old_myisam_max_sort_file_size;
```

The system variables that affect REPAIR TABLE can be set globally at server startup if you want the values to be in effect by default. For example, add these lines to the server my.cnf file:

```
[mysqld]
myisam_sort_buffer_size=256M
key_buffer_size=1G
myisam_max_sort_file_size=100G
```

These settings do not include read_buffer_size. Setting read_buffer_size globally to a large value does so for all sessions and can cause performance to suffer due to excessive memory allocation for a server with many simultaneous sessions.

8.7 Optimizing for MEMORY Tables

Consider using MEMORY tables for noncritical data that is accessed often, and is read-only or rarely updated. Benchmark your application against equivalent InnoDB or MyISAM tables under a realistic workload, to confirm that any additional performance is worth the risk of losing data, or the overhead of copying data from a disk-based table at application start.

For best performance with MEMORY tables, examine the kinds of queries against each table, and specify the type to use for each associated index, either a B-tree index or a hash index. On the CREATE INDEX statement, use the clause USING BTREE or USING HASH. B-tree indexes are fast for queries that do greater-than or less-than comparisons through operators such as > or BETWEEN. Hash indexes are only fast for queries that look up single values through the = operator, or a restricted set of values through the IN operator. For why USING BTREE is often a better choice than the default USING HASH, see Section 8.2.1.20, “How to Avoid Full Table Scans”. For implementation details of the different types of MEMORY indexes, see Section 8.3.8, “Comparison of B-Tree and Hash Indexes”.

8.8 Understanding the Query Execution Plan

Depending on the details of your tables, columns, indexes, and the conditions in your WHERE clause, the MySQL optimizer considers many techniques to efficiently perform the lookups involved in an SQL query. A query on a huge table can be performed without reading all the rows; a join involving several tables can be performed without comparing every combination of rows. The set of operations that the optimizer chooses to perform the most efficient query is called the “query execution plan”, also known as the EXPLAIN plan. Your goals are to recognize the aspects of the EXPLAIN plan that indicate a query is optimized well, and to learn the SQL syntax and indexing techniques to improve the plan if you see some inefficient operations.

8.8.1 Optimizing Queries with EXPLAIN

The EXPLAIN statement can be used to obtain information about how MySQL executes a statement:

- In MySQL 5.7, permitted explainable statements for EXPLAIN are SELECT, DELETE, INSERT, REPLACE, and UPDATE.

- When EXPLAIN is used with an explainable statement, MySQL displays information from the optimizer about the statement execution plan. That is, MySQL explains how it would process the statement, including information about how tables are joined and in which order. For information about using EXPLAIN to obtain execution plan information, see Section 8.8.2, “EXPLAIN Output Format”.

- When EXPLAIN is used with FOR CONNECTION *connection_id* rather than an explainable statement, it displays the execution plan for the statement executing in the named connection. See Section 8.8.3, “Obtaining Execution Plan Information for a Named Connection”.

- Before MySQL 5.7.3, EXPLAIN EXTENDED can be used to obtain additional execution plan information. See Section 8.8.4, “EXPLAIN EXTENDED Output Format”. As of MySQL 5.7.3, extended output is enabled by default and the EXTENDED keyword is unnecessry.

- Before MySQL 5.7.3, EXPLAIN PARTITIONS is useful for examining queries involving partitioned tables. See Section 17.3.5, “Obtaining Information About Partitions”. As of MySQL 5.7.3, partition information is enabled by default and the PARTITIONS keyword is unnecessry.

- The FORMAT option can be used to select the output format. TRADITIONAL presents the output in tabular format. This is the default if no FORMAT option is present. JSON format displays the information in JSON format. With FORMAT = JSON, the output includes extended and partition information.

With the help of EXPLAIN, you can see where you should add indexes to tables so that the statement executes faster by using indexes to find rows. You can also use EXPLAIN to check whether the optimizer joins the tables in an optimal order. To give a hint to the optimizer to use a join order corresponding to the order in which the tables are named in a SELECT statement, begin the statement with SELECT STRAIGHT_JOIN rather than just SELECT. (See Section 13.2.9, “SELECT Syntax”.)

The optimizer trace may sometimes provide information complementary to that of EXPLAIN. However, the optimizer trace format and content are subject to change between versions. For details, see MySQL Internals: Tracing the Optimizer.

If you have a problem with indexes not being used when you believe that they should be, run ANALYZE TABLE to update table statistics, such as cardinality of keys, that can affect the choices the optimizer makes. See Section 13.7.2.1, "ANALYZE TABLE Syntax".

Note

EXPLAIN can also be used to obtain information about the columns in a table. EXPLAIN *tbl_name* is synonymous with DESCRIBE *tbl_name* and SHOW COLUMNS FROM *tbl_name*. For more information, see Section 13.8.1, "DESCRIBE Syntax", and Section 13.7.5.5, "SHOW COLUMNS Syntax".

8.8.2 EXPLAIN Output Format

The EXPLAIN statement provides information about the execution plan for a SELECT statement.

EXPLAIN returns a row of information for each table used in the SELECT statement. It lists the tables in the output in the order that MySQL would read them while processing the statement. MySQL resolves all joins using a nested-loop join method. This means that MySQL reads a row from the first table, and then finds a matching row in the second table, the third table, and so on. When all tables are processed, MySQL outputs the selected columns and backtracks through the table list until a table is found for which there are more matching rows. The next row is read from this table and the process continues with the next table.

Before MySQL 5.7.3, when the EXTENDED keyword is used, EXPLAIN produces extra information that can be viewed by issuing a SHOW WARNINGS statement following the EXPLAIN statement. EXPLAIN EXTENDED also displays the filtered column. See Section 8.8.4, "EXPLAIN EXTENDED Output Format". As of MySQL 5.7.3, extended output is enabled by default and the EXTENDED keyword is unnecessry.

Note

You cannot use the EXTENDED and PARTITIONS keywords together in the same EXPLAIN statement.

- EXPLAIN Output Columns
- EXPLAIN Join Types
- EXPLAIN Extra Information
- EXPLAIN Output Interpretation

EXPLAIN Output Columns

This section describes the output columns produced by EXPLAIN. Later sections provide additional information about the type and Extra columns.

Each output row from EXPLAIN provides information about one table. Each row contains the values summarized in Table 8.1, "EXPLAIN Output Columns", and described in more detail following the table.

Table 8.1 EXPLAIN Output Columns

Column	Meaning
id	The SELECT identifier
select_type	The SELECT type
table	The table for the output row

Column	Meaning
partitions	The matching partitions
type	The join type
possible_keys	The possible indexes to choose
key	The index actually chosen
key_len	The length of the chosen key
ref	The columns compared to the index
rows	Estimate of rows to be examined
filtered	Percentage of rows filtered by table condition
Extra	Additional information

- id

  The SELECT identifier. This is the sequential number of the SELECT within the query. The value can be NULL if the row refers to the union result of other rows. In this case, the table column shows a value like <union*M*,*N*> to indicate that the row refers to the union of the rows with id values of *M* and *N*.

- select_type

  The type of SELECT, which can be any of those shown in the following table.

select_type Value	Meaning
SIMPLE	Simple SELECT (not using UNION or subqueries)
PRIMARY	Outermost SELECT
UNION	Second or later SELECT statement in a UNION
DEPENDENT UNION	Second or later SELECT statement in a UNION, dependent on outer query
UNION RESULT	Result of a UNION.
SUBQUERY	First SELECT in subquery
DEPENDENT SUBQUERY	First SELECT in subquery, dependent on outer query
DERIVED	Derived table SELECT (subquery in FROM clause)
MATERIALIZED	Materialized subquery
UNCACHEABLE SUBQUERY	A subquery for which the result cannot be cached and must be re-evaluated for each row of the outer query
UNCACHEABLE UNION	The second or later select in a UNION that belongs to an uncacheable subquery (see UNCACHEABLE SUBQUERY)

  DEPENDENT typically signifies the use of a correlated subquery. See Section 13.2.10.7, “Correlated Subqueries”.

  DEPENDENT SUBQUERY evaluation differs from UNCACHEABLE SUBQUERY evaluation. For DEPENDENT SUBQUERY, the subquery is re-evaluated only once for each set of different values of the variables from its outer context. For UNCACHEABLE SUBQUERY, the subquery is re-evaluated for each row of the outer context.

  Cacheability of subqueries differs from caching of query results in the query cache (which is described in Section 8.9.3.1, “How the Query Cache Operates”). Subquery caching occurs during query execution, whereas the query cache is used to store results only after query execution finishes.

As of MySQL 5.7.2, the select_type value for non-SELECT statements displays the statement type for affected tables. For example, select_type is DELETE for DELETE statements.

- table

  The name of the table to which the row of output refers. This can also be one of the following values:

  - <union*M*,*N*>: The row refers to the union of the rows with id values of *M* and *N*.
  - <derived*N*>: The row refers to the derived table result for the row with an id value of *N*. A derived table may result, for example, from a subquery in the FROM clause.
  - <subquery*N*>: The row refers to the result of a materialized subquery for the row with an id value of *N*. See Optimizing Subqueries with Subquery Materialization.

- partitions

  The partitions from which records would be matched by the query. Before MySQL 5.7.3, this column is displayed only if the PARTITIONS keyword is used. The value is NULL for nonpartitioned tables. See Section 17.3.5, “Obtaining Information About Partitions”. As of MySQL 5.7.3, partition information is enabled by default and the PARTITIONS keyword is unnecessry.

- type

  The join type. For descriptions of the different types, see EXPLAIN Join Types.

- possible_keys

  The possible_keys column indicates which indexes MySQL can choose from use to find the rows in this table. Note that this column is totally independent of the order of the tables as displayed in the output from EXPLAIN. That means that some of the keys in possible_keys might not be usable in practice with the generated table order.

  If this column is NULL, there are no relevant indexes. In this case, you may be able to improve the performance of your query by examining the WHERE clause to check whether it refers to some column or columns that would be suitable for indexing. If so, create an appropriate index and check the query with EXPLAIN again. See Section 13.1.6, “ALTER TABLE Syntax”.

  To see what indexes a table has, use SHOW INDEX FROM *tbl_name*.

- key

  The key column indicates the key (index) that MySQL actually decided to use. If MySQL decides to use one of the possible_keys indexes to look up rows, that index is listed as the key value.

  It is possible that key will name an index that is not present in the possible_keys value. This can happen if none of the possible_keys indexes are suitable for looking up rows, but all the columns selected by the query are columns of some other index. That is, the named index covers the selected columns, so although it is not used to determine which rows to retrieve, an index scan is more efficient than a data row scan.

  For InnoDB, a secondary index might cover the selected columns even if the query also selects the primary key because InnoDB stores the primary key value with each secondary index. If key is NULL, MySQL found no index to use for executing the query more efficiently.

  To force MySQL to use or ignore an index listed in the possible_keys column, use FORCE INDEX, USE INDEX, or IGNORE INDEX in your query. See Section 13.2.9.3, “Index Hint Syntax”.

For `MyISAM` tables, running `ANALYZE TABLE` helps the optimizer choose better indexes. For `MyISAM` tables, `myisamchk --analyze` does the same. See Section 13.7.2.1, “`ANALYZE TABLE` Syntax”, and Section 7.6, “`MyISAM` Table Maintenance and Crash Recovery”.

- `key_len`

  The `key_len` column indicates the length of the key that MySQL decided to use. The length is `NULL` if the `key` column says `NULL`. Note that the value of `key_len` enables you to determine how many parts of a multiple-part key MySQL actually uses.

- `ref`

  The `ref` column shows which columns or constants are compared to the index named in the `key` column to select rows from the table.

- `rows`

  The `rows` column indicates the number of rows MySQL believes it must examine to execute the query.

  For `InnoDB` tables, this number is an estimate, and may not always be exact.

- `filtered`

  The `filtered` column indicates an estimated percentage of table rows that will be filtered by the table condition. That is, `rows` shows the estimated number of rows examined and `rows` × `filtered` / `100` shows the number of rows that will be joined with previous tables. Before MySQL 5.7.3, this column is displayed if you use `EXPLAIN EXTENDED`. As of MySQL 5.7.3, extended output is enabled by default and the `EXTENDED` keyword is unnecessry.

- `Extra`

  This column contains additional information about how MySQL resolves the query. For descriptions of the different values, see `EXPLAIN` Extra Information.

`EXPLAIN` Join Types

The `type` column of `EXPLAIN` output describes how tables are joined. The following list describes the join types, ordered from the best type to the worst:

- `system`

  The table has only one row (= system table). This is a special case of the `const` join type.

- `const`

  The table has at most one matching row, which is read at the start of the query. Because there is only one row, values from the column in this row can be regarded as constants by the rest of the optimizer. `const` tables are very fast because they are read only once.

  `const` is used when you compare all parts of a `PRIMARY KEY` or `UNIQUE` index to constant values. In the following queries, *`tbl_name`* can be used as a `const` table:

```
SELECT * FROM tbl_name WHERE primary_key=1;

SELECT * FROM tbl_name
  WHERE primary_key_part1=1 AND primary_key_part2=2;
```

- `eq_ref`

  One row is read from this table for each combination of rows from the previous tables. Other than the `system` and `const` types, this is the best possible join type. It is used when all parts of an index are used by the join and the index is a `PRIMARY KEY` or `UNIQUE NOT NULL` index.

  `eq_ref` can be used for indexed columns that are compared using the `=` operator. The comparison value can be a constant or an expression that uses columns from tables that are read before this table. In the following examples, MySQL can use an `eq_ref` join to process *`ref_table`*:

  ```
  SELECT * FROM ref_table,other_table
    WHERE ref_table.key_column=other_table.column;

  SELECT * FROM ref_table,other_table
    WHERE ref_table.key_column_part1=other_table.column
    AND ref_table.key_column_part2=1;
  ```

- `ref`

  All rows with matching index values are read from this table for each combination of rows from the previous tables. `ref` is used if the join uses only a leftmost prefix of the key or if the key is not a `PRIMARY KEY` or `UNIQUE` index (in other words, if the join cannot select a single row based on the key value). If the key that is used matches only a few rows, this is a good join type.

  `ref` can be used for indexed columns that are compared using the `=` or `<=>` operator. In the following examples, MySQL can use a `ref` join to process *`ref_table`*:

  ```
  SELECT * FROM ref_table WHERE key_column=expr;

  SELECT * FROM ref_table,other_table
    WHERE ref_table.key_column=other_table.column;

  SELECT * FROM ref_table,other_table
    WHERE ref_table.key_column_part1=other_table.column
    AND ref_table.key_column_part2=1;
  ```

- `fulltext`

  The join is performed using a `FULLTEXT` index.

- `ref_or_null`

  This join type is like `ref`, but with the addition that MySQL does an extra search for rows that contain `NULL` values. This join type optimization is used most often in resolving subqueries. In the following examples, MySQL can use a `ref_or_null` join to process *`ref_table`*:

  ```
  SELECT * FROM ref_table
    WHERE key_column=expr OR key_column IS NULL;
  ```

  See Section 8.2.1.8, “`IS NULL` Optimization”.

- `index_merge`

  This join type indicates that the Index Merge optimization is used. In this case, the `key` column in the output row contains a list of indexes used, and `key_len` contains a list of the longest key parts for the indexes used. For more information, see Section 8.2.1.4, “Index Merge Optimization”.

- `unique_subquery`

This type replaces ref for some IN subqueries of the following form:

```
value IN (SELECT primary_key FROM single_table WHERE some_expr)
```

unique_subquery is just an index lookup function that replaces the subquery completely for better efficiency.

- index_subquery

  This join type is similar to unique_subquery. It replaces IN subqueries, but it works for nonunique indexes in subqueries of the following form:

  ```
  value IN (SELECT key_column FROM single_table WHERE some_expr)
  ```

- range

  Only rows that are in a given range are retrieved, using an index to select the rows. The key column in the output row indicates which index is used. The key_len contains the longest key part that was used. The ref column is NULL for this type.

  range can be used when a key column is compared to a constant using any of the =, <>, >, >=, <, <=, IS NULL, <=>, BETWEEN, or IN() operators:

  ```
  SELECT * FROM tbl_name
    WHERE key_column = 10;

  SELECT * FROM tbl_name
    WHERE key_column BETWEEN 10 and 20;

  SELECT * FROM tbl_name
    WHERE key_column IN (10,20,30);

  SELECT * FROM tbl_name
    WHERE key_part1 = 10 AND key_part2 IN (10,20,30);
  ```

- index

  The index join type is the same as ALL, except that the index tree is scanned. This occurs two ways:

  - If the index is a covering index for the queries and can be used to satisfy all data required from the table, only the index tree is scanned. In this case, the Extra column says Using index. An index-only scan usually is faster than ALL because the size of the index usually is smaller than the table data.

  - A full table scan is performed using reads from the index to look up data rows in index order. Uses index does not appear in the Extra column.

  MySQL can use this join type when the query uses only columns that are part of a single index.

- ALL

  A full table scan is done for each combination of rows from the previous tables. This is normally not good if the table is the first table not marked const, and usually *very* bad in all other cases. Normally, you can avoid ALL by adding indexes that enable row retrieval from the table based on constant values or column values from earlier tables.

EXPLAIN Extra Information

The Extra column of EXPLAIN output contains additional information about how MySQL resolves the query. The following list explains the values that can appear in this column. If you want to make your queries as fast as possible, look out for Extra values of Using filesort and Using temporary.

- const row not found

  For a query such as SELECT ... FROM *tbl_name*, the table was empty.

- Deleting all rows

  For DELETE, some storage engines (such as MyISAM) support a handler method that removes all table rows in a simple and fast way. This Extra value is displayed if the engine uses this optimization.

- Distinct

  MySQL is looking for distinct values, so it stops searching for more rows for the current row combination after it has found the first matching row.

- FirstMatch(*tbl_name*)

  The semi-join FirstMatch join shortcutting strategy is used for *tbl_name*.

- Full scan on NULL key

  This occurs for subquery optimization as a fallback strategy when the optimizer cannot use an index-lookup access method.

- Impossible HAVING

  The HAVING clause is always false and cannot select any rows.

- Impossible WHERE

  The WHERE clause is always false and cannot select any rows.

- Impossible WHERE noticed after reading const tables

  MySQL has read all const (and system) tables and notice that the WHERE clause is always false.

- LooseScan(*m*..*n*)

  The semi-join LooseScan strategy is used. *m* and *n* are key part numbers.

- No matching min/max row

  No row satisfies the condition for a query such as SELECT MIN(...) FROM ... WHERE *condition*.

- no matching row in const table

  For a query with a join, there was an empty table or a table with no rows satisfying a unique index condition.

- No matching rows after partition pruning

  For DELETE or UPDATE, the optimizer found nothing to delete or update after partition pruning. It is similar in meaning to Impossible WHERE for SELECT statements.

- No tables used

The query has no `FROM` clause, or has a `FROM DUAL` clause.

For `INSERT` or `REPLACE` statements, `EXPLAIN` displays this value when there is no `SELECT` part. For example, it appears for `EXPLAIN INSERT INTO t VALUES(10)` because that is equivalent to `EXPLAIN INSERT INTO t SELECT 10 FROM DUAL`.

- `Not exists`

  MySQL was able to do a `LEFT JOIN` optimization on the query and does not examine more rows in this table for the previous row combination after it finds one row that matches the `LEFT JOIN` criteria. Here is an example of the type of query that can be optimized this way:

  ```
  SELECT * FROM t1 LEFT JOIN t2 ON t1.id=t2.id
    WHERE t2.id IS NULL;
  ```

  Assume that `t2.id` is defined as `NOT NULL`. In this case, MySQL scans `t1` and looks up the rows in `t2` using the values of `t1.id`. If MySQL finds a matching row in `t2`, it knows that `t2.id` can never be `NULL`, and does not scan through the rest of the rows in `t2` that have the same `id` value. In other words, for each row in `t1`, MySQL needs to do only a single lookup in `t2`, regardless of how many rows actually match in `t2`.

- `Plan isn't ready yet`

  This value occurs with `EXPLAIN FOR CONNECTION` when the optimizer has not finished creating the execution plan for the statement executing in the named connection. If execution plan output comprises multiple lines, any or all of them could have this `Extra` value, depending on the progress of the optimizer in determining the full execution plan.

- `Range checked for each record (index map: N)`

  MySQL found no good index to use, but found that some of indexes might be used after column values from preceding tables are known. For each row combination in the preceding tables, MySQL checks whether it is possible to use a `range` or `index_merge` access method to retrieve rows. This is not very fast, but is faster than performing a join with no index at all. The applicability criteria are as described in Section 8.2.1.3, "Range Optimization", and Section 8.2.1.4, "Index Merge Optimization", with the exception that all column values for the preceding table are known and considered to be constants.

  Indexes are numbered beginning with 1, in the same order as shown by `SHOW INDEX` for the table. The index map value `N` is a bitmask value that indicates which indexes are candidates. For example, a value of `0x19` (binary 11001) means that indexes 1, 4, and 5 will be considered.

- `Scanned N databases`

  This indicates how many directory scans the server performs when processing a query for `INFORMATION_SCHEMA` tables, as described in Section 8.2.4, "Optimizing `INFORMATION_SCHEMA` Queries". The value of `N` can be 0, 1, or `all`.

- `Select tables optimized away`

  The query contained only aggregate functions (`MIN()`, `MAX()`) that were all resolved using an index, or `COUNT(*)` for `MyISAM`, and no `GROUP BY` clause. The optimizer determined that only one row should be returned.

- `Skip_open_table`, `Open_frm_only`, `Open_trigger_only`, `Open_full_table`

These values indicate file-opening optimizations that apply to queries for INFORMATION_SCHEMA tables, as described in Section 8.2.4, “Optimizing INFORMATION_SCHEMA Queries”.

  - Skip_open_table: Table files do not need to be opened. The information has already become available within the query by scanning the database directory.
  - Open_frm_only: Only the table's .frm file need be opened.
  - Open_trigger_only: Only the table's .TRG file need be opened.
  - Open_full_table: The unoptimized information lookup. The .frm, .MYD, and .MYI files must be opened.

- Start temporary, End temporary

  This indicates temporary table use for the semi-join Duplicate Weedout strategy.

- unique row not found

  For a query such as SELECT ... FROM *tbl_name*, no rows satisfy the condition for a UNIQUE index or PRIMARY KEY on the table.

- Using filesort

  MySQL must do an extra pass to find out how to retrieve the rows in sorted order. The sort is done by going through all rows according to the join type and storing the sort key and pointer to the row for all rows that match the WHERE clause. The keys then are sorted and the rows are retrieved in sorted order. See Section 8.2.1.15, “ORDER BY Optimization”.

- Using index

  The column information is retrieved from the table using only information in the index tree without having to do an additional seek to read the actual row. This strategy can be used when the query uses only columns that are part of a single index.

  If the Extra column also says Using where, it means the index is being used to perform lookups of key values. Without Using where, the optimizer may be reading the index to avoid reading data rows but not using it for lookups. For example, if the index is a covering index for the query, the optimizer may scan it without using it for lookups.

  For InnoDB tables that have a user-defined clustered index, that index can be used even when Using index is absent from the Extra column. This is the case if type is index and key is PRIMARY.

- Using index condition

  Tables are read by accessing index tuples and testing them first to determine whether to read full table rows. In this way, index information is used to defer (“push down”) reading full table rows unless it is necessary. See Section 8.2.1.6, “Index Condition Pushdown Optimization”.

- Using index for group-by

  Similar to the Using index table access method, Using index for group-by indicates that MySQL found an index that can be used to retrieve all columns of a GROUP BY or DISTINCT query without any extra disk access to the actual table. Additionally, the index is used in the most efficient way so that for each group, only a few index entries are read. For details, see Section 8.2.1.16, “GROUP BY Optimization”.

- Using join buffer (Block Nested Loop), Using join buffer (Batched Key Access)

  Tables from earlier joins are read in portions into the join buffer, and then their rows are used from the buffer to perform the join with the current table. (Block Nested Loop) indicates use of the Block Nested-Loop algorithm and (Batched Key Access) indicates use of the Batched Key Access algorithm. That is, the keys from the table on the preceding line of the EXPLAIN output will be buffered, and the matching rows will be fetched in batches from the table represented by the line in which Using join buffer appears.

- Using MRR

  Tables are read using the Multi-Range Read optimization strategy. See Section 8.2.1.13, "Multi-Range Read Optimization".

- Using sort_union(...), Using union(...), Using intersect(...)

  These indicate how index scans are merged for the index_merge join type. See Section 8.2.1.4, "Index Merge Optimization".

- Using temporary

  To resolve the query, MySQL needs to create a temporary table to hold the result. This typically happens if the query contains GROUP BY and ORDER BY clauses that list columns differently.

- Using where

  A WHERE clause is used to restrict which rows to match against the next table or send to the client. Unless you specifically intend to fetch or examine all rows from the table, you may have something wrong in your query if the Extra value is not Using where and the table join type is ALL or index.

- Using where with pushed condition

  This item applies to NDB tables *only*. It means that MySQL Cluster is using the Condition Pushdown optimization to improve the efficiency of a direct comparison between a nonindexed column and a constant. In such cases, the condition is "pushed down" to the cluster's data nodes and is evaluated on all data nodes simultaneously. This eliminates the need to send nonmatching rows over the network, and can speed up such queries by a factor of 5 to 10 times over cases where Condition Pushdown could be but is not used. For more information, see Section 8.2.1.5, "Engine Condition Pushdown Optimization".

EXPLAIN Output Interpretation

You can get a good indication of how good a join is by taking the product of the values in the rows column of the EXPLAIN output. This should tell you roughly how many rows MySQL must examine to execute the query. If you restrict queries with the max_join_size system variable, this row product also is used to determine which multiple-table SELECT statements to execute and which to abort. See Section 8.11.2, "Tuning Server Parameters".

The following example shows how a multiple-table join can be optimized progressively based on the information provided by EXPLAIN.

Suppose that you have the SELECT statement shown here and that you plan to examine it using EXPLAIN:

```
EXPLAIN SELECT tt.TicketNumber, tt.TimeIn,
               tt.ProjectReference, tt.EstimatedShipDate,
               tt.ActualShipDate, tt.ClientID,
               tt.ServiceCodes, tt.RepetitiveID,
               tt.CurrentProcess, tt.CurrentDPPerson,
               tt.RecordVolume, tt.DPPrinted, et.COUNTRY,
               et_1.COUNTRY, do.CUSTNAME
        FROM tt, et, et AS et_1, do
        WHERE tt.SubmitTime IS NULL
          AND tt.ActualPC = et.EMPLOYID
          AND tt.AssignedPC = et_1.EMPLOYID
          AND tt.ClientID = do.CUSTNMBR;
```

For this example, make the following assumptions:

- The columns being compared have been declared as follows.

Table	Column	Data Type
tt	ActualPC	CHAR(10)
tt	AssignedPC	CHAR(10)
tt	ClientID	CHAR(10)
et	EMPLOYID	CHAR(15)
do	CUSTNMBR	CHAR(15)

- The tables have the following indexes.

Table	Index
tt	ActualPC
tt	AssignedPC
tt	ClientID
et	EMPLOYID (primary key)
do	CUSTNMBR (primary key)

- The tt.ActualPC values are not evenly distributed.

Initially, before any optimizations have been performed, the EXPLAIN statement produces the following information:

```
table type possible_keys key  key_len ref  rows  Extra
et    ALL  PRIMARY       NULL NULL    NULL 74
do    ALL  PRIMARY       NULL NULL    NULL 2135
et_1  ALL  PRIMARY       NULL NULL    NULL 74
tt    ALL  AssignedPC,   NULL NULL    NULL 3872
           ClientID,
           ActualPC
      Range checked for each record (index map: 0x23)
```

Because type is ALL for each table, this output indicates that MySQL is generating a Cartesian product of all the tables; that is, every combination of rows. This takes quite a long time, because the product of the number of rows in each table must be examined. For the case at hand, this product is 74 × 2135 × 74 × 3872 = 45,268,558,720 rows. If the tables were bigger, you can only imagine how long it would take.

One problem here is that MySQL can use indexes on columns more efficiently if they are declared as the same type and size. In this context, VARCHAR and CHAR are considered the same if they are declared as the same size. tt.ActualPC is declared as CHAR(10) and et.EMPLOYID is CHAR(15), so there is a length mismatch.

To fix this disparity between column lengths, use ALTER TABLE to lengthen ActualPC from 10 characters to 15 characters:

```
mysql> ALTER TABLE tt MODIFY ActualPC VARCHAR(15);
```

Now tt.ActualPC and et.EMPLOYID are both VARCHAR(15). Executing the EXPLAIN statement again produces this result:

```
table type   possible_keys key     key_len ref         rows    Extra
tt    ALL    AssignedPC,   NULL    NULL    NULL        3872    Using
             ClientID,                                         where
             ActualPC
do    ALL    PRIMARY       NULL    NULL    NULL        2135
      Range checked for each record (index map: 0x1)
et_1  ALL    PRIMARY       NULL    NULL    NULL        74
      Range checked for each record (index map: 0x1)
et    eq_ref PRIMARY       PRIMARY 15      tt.ActualPC 1
```

This is not perfect, but is much better: The product of the rows values is less by a factor of 74. This version executes in a couple of seconds.

A second alteration can be made to eliminate the column length mismatches for the tt.AssignedPC = et_1.EMPLOYID and tt.ClientID = do.CUSTNMBR comparisons:

```
mysql> ALTER TABLE tt MODIFY AssignedPC VARCHAR(15),
    ->                MODIFY ClientID   VARCHAR(15);
```

After that modification, EXPLAIN produces the output shown here:

```
table type   possible_keys key      key_len ref           rows Extra
et    ALL    PRIMARY       NULL     NULL    NULL          74
tt    ref    AssignedPC,   ActualPC 15      et.EMPLOYID   52   Using
             ClientID,                                         where
             ActualPC
et_1  eq_ref PRIMARY       PRIMARY  15      tt.AssignedPC 1
do    eq_ref PRIMARY       PRIMARY  15      tt.ClientID   1
```

At this point, the query is optimized almost as well as possible. The remaining problem is that, by default, MySQL assumes that values in the tt.ActualPC column are evenly distributed, and that is not the case for the tt table. Fortunately, it is easy to tell MySQL to analyze the key distribution:

```
mysql> ANALYZE TABLE tt;
```

With the additional index information, the join is perfect and EXPLAIN produces this result:

```
table type   possible_keys key     key_len ref          rows Extra
tt    ALL    AssignedPC    NULL    NULL    NULL          3872 Using
             ClientID,                                        where
             ActualPC
et    eq_ref PRIMARY       PRIMARY 15      tt.ActualPC   1
et_1  eq_ref PRIMARY       PRIMARY 15      tt.AssignedPC 1
do    eq_ref PRIMARY       PRIMARY 15      tt.ClientID   1
```

Note that the rows column in the output from EXPLAIN is an educated guess from the MySQL join optimizer. Check whether the numbers are even close to the truth by comparing the rows product with the actual number of rows that the query returns. If the numbers are quite different, you might get better performance by using STRAIGHT_JOIN in your SELECT statement and trying to list the tables in a different order in the FROM clause.

It is possible in some cases to execute statements that modify data when EXPLAIN SELECT is used with a subquery; for more information, see Section 13.2.10.8, “Subqueries in the FROM Clause”.

8.8.3 Obtaining Execution Plan Information for a Named Connection

To obtain the execution plan for an explainable statement executing in a named connection, use this statement:

```
EXPLAIN [options] FOR CONNECTION connection_id;
```

For example, if you are running a statement in one session that is taking a long time to complete, using `EXPLAIN FOR CONNECTION` in another session may yield useful information about the cause of the delay and help you optimize the statement.

`connection_id` is the connection identifier, as obtained from the `INFORMATION_SCHEMA PROCESSLIST` table or the `SHOW PROCESSLIST` statement. If you have the `PROCESS` privilege, you can specify the identifier for any connection. Otherwise, you can specify the identifier only for your own connections.

If the named connection is not executing a statement, the result is empty. Otherwise, `EXPLAIN FOR CONNECTION` applies only if the statement being executed in the named connection is explainable. This includes `SELECT`, `DELETE`, `INSERT`, `REPLACE`, and `UPDATE`.

If the named connection is executing an explainable statement, the output is what you would obtain by using `EXPLAIN` on the statement itself.

If the named connection is executing a statement that is not explainable, an error occurs. For example, you cannot name the connection identifier for your current session because `EXPLAIN` is not explainable:

```
mysql> SELECT CONNECTION_ID();
+-----------------+
| CONNECTION_ID() |
+-----------------+
|             373 |
+-----------------+
1 row in set (0.00 sec)

mysql> EXPLAIN FOR CONNECTION 373;
ERROR 1889 (HY000): EXPLAIN FOR CONNECTION command is supported
only for SELECT/UPDATE/INSERT/DELETE/REPLACE
```

8.8.4 `EXPLAIN EXTENDED` Output Format

When `EXPLAIN` is used with the `EXTENDED` keyword, the output includes a `filtered` column not otherwise displayed. This column indicates the estimated percentage of table rows that will be filtered by the table condition. In addition, the statement produces extra information that can be viewed by issuing a `SHOW WARNINGS` statement following the `EXPLAIN` statement. The `Message` value in `SHOW WARNINGS` output displays how the optimizer qualifies table and column names in the `SELECT` statement, what the `SELECT` looks like after the application of rewriting and optimization rules, and possibly other notes about the optimization process.

Note

As of MySQL 5.7.3, the `EXPLAIN` statement is changed so that the effect of the `EXTENDED` keyword is always enabled. `EXTENDED` is still recognized, but is superfluous and is deprecated. It will be removed from `EXPLAIN` syntax in a future MySQL release.

Here is an example of extended output:

```
mysql> EXPLAIN EXTENDED
    -> SELECT t1.a, t1.a IN (SELECT t2.a FROM t2) FROM t1\G
*************************** 1. row ***************************
           id: 1
  select_type: PRIMARY
        table: t1
         type: index
possible_keys: NULL
          key: PRIMARY
      key_len: 4
          ref: NULL
         rows: 4
     filtered: 100.00
        Extra: Using index
*************************** 2. row ***************************
           id: 2
  select_type: SUBQUERY
        table: t2
         type: index
possible_keys: a
          key: a
      key_len: 5
          ref: NULL
         rows: 3
     filtered: 100.00
        Extra: Using index
2 rows in set, 1 warning (0.00 sec)

mysql> SHOW WARNINGS\G
*************************** 1. row ***************************
  Level: Note
   Code: 1003
Message: /* select#1 */ select `test`.`t1`.`a` AS `a`,
         <in_optimizer>(`test`.`t1`.`a`,`test`.`t1`.`a` in
         ( <materialize> (/* select#2 */ select `test`.`t2`.`a`
         from `test`.`t2` where 1 having 1 ),
         <primary_index_lookup>(`test`.`t1`.`a` in
         <temporary table> on <auto_key>
         where ((`test`.`t1`.`a` = `materialized-subquery`.`a`))))) AS `t1.a
         IN (SELECT t2.a FROM t2)` from `test`.`t1`
1 row in set (0.00 sec)
```

EXPLAIN EXTENDED can be used with SELECT, DELETE, INSERT, REPLACE, and UPDATE statements. However, the following SHOW WARNINGS statement displays a nonempty result only for SELECT statements.

Because the statement displayed by SHOW WARNINGS may contain special markers to provide information about query rewriting or optimizer actions, the statement is not necessarily valid SQL and is not intended to be executed. The output may also include rows with Message values that provide additional non-SQL explanatory notes about actions taken by the optimizer.

The following list describes special markers that can appear in EXTENDED output displayed by SHOW WARNINGS:

- `<auto_key>`

  An automatically generated key for a temporary table.

- `<cache>(`*expr*`)`

  The expression (such as a scalar subquery) is executed once and the resulting value is saved in memory for later use. For results consisting of multiple values, a temporary table may be created and you will see `<temporary table>` instead.

- `<exists>(`*`query fragment`*`)`

  The subquery predicate is converted to an `EXISTS` predicate and the subquery is transformed so that it can be used together with the `EXISTS` predicate.

- `<in_optimizer>(`*`query fragment`*`)`

  This is an internal optimizer object with no user significance.

- `<index_lookup>(`*`query fragment`*`)`

  The query fragment is processed using an index lookup to find qualifying rows.

- `<if>(`*`condition, expr1, expr2`*`)`

  If the condition is true, evaluate to *`expr1`*, otherwise *`expr2`*.

- `<is_not_null_test>(`*`expr`*`)`

  A test to verify that the expression does not evaluate to `NULL`.

- `<materialize>(`*`query fragment`*`)`

  Subquery materialization is used.

- `` `materialized-subquery`. ``*`col_name`*

  A reference to the column *`col_name`* in an internal temporary table materialized to hold the result from evaluating a subquery.

- `<primary_index_lookup>(`*`query fragment`*`)`

  The query fragment is processed using a primary key lookup to find qualifying rows.

- `<ref_null_helper>(`*`expr`*`)`

  This is an internal optimizer object with no user significance.

- `/* select#`*`N`*` */ `*`select_stmt`*

  The `SELECT` is associated with the row in non-`EXTENDED` `EXPLAIN` output that has an `id` value of *`N`*.

- *`outer_tables`*` semi join (`*`inner_tables`*`)`

  A semi-join operation. *`inner_tables`* shows the tables that were not pulled out. See Optimizing Subqueries with Semi-Join Transformations.

- `<temporary table>`

  This represents an internal temporary table created to cache an intermediate result.

When some tables are of `const` or `system` type, expressions involving columns from these tables are evaluated early by the optimizer and are not part of the displayed statement. However, with `FORMAT=JSON`, some `const` table accesses are displayed as a `ref` access that uses a const value.

8.8.5 Estimating Query Performance

In most cases, you can estimate query performance by counting disk seeks. For small tables, you can usually find a row in one disk seek (because the index is probably cached). For bigger tables, you can estimate that, using B-tree indexes, you need this many seeks to find a row: `log(row_count) / log(index_block_length / 3 * 2 / (index_length + data_pointer_length)) + 1`.

In MySQL, an index block is usually 1,024 bytes and the data pointer is usually four bytes. For a 500,000-row table with a key value length of three bytes (the size of `MEDIUMINT`), the formula indicates `log(500,000)/log(1024/3*2/(3+4)) + 1` = `4` seeks.

This index would require storage of about 500,000 * 7 * 3/2 = 5.2MB (assuming a typical index buffer fill ratio of 2/3), so you probably have much of the index in memory and so need only one or two calls to read data to find the row.

For writes, however, you need four seek requests to find where to place a new index value and normally two seeks to update the index and write the row.

Note that the preceding discussion does not mean that your application performance slowly degenerates by log *N*. As long as everything is cached by the OS or the MySQL server, things become only marginally slower as the table gets bigger. After the data gets too big to be cached, things start to go much slower until your applications are bound only by disk seeks (which increase by log *N*). To avoid this, increase the key cache size as the data grows. For `MyISAM` tables, the key cache size is controlled by the `key_buffer_size` system variable. See Section 8.11.2, “Tuning Server Parameters”.

8.8.6 Controlling the Query Optimizer

MySQL provides optimizer control through system variables that affect how query plans are evaluated and which switchable optimizations are enabled.

8.8.6.1 Controlling Query Plan Evaluation

The task of the query optimizer is to find an optimal plan for executing an SQL query. Because the difference in performance between “good” and “bad” plans can be orders of magnitude (that is, seconds versus hours or even days), most query optimizers, including that of MySQL, perform a more or less exhaustive search for an optimal plan among all possible query evaluation plans. For join queries, the number of possible plans investigated by the MySQL optimizer grows exponentially with the number of tables referenced in a query. For small numbers of tables (typically less than 7 to 10) this is not a problem. However, when larger queries are submitted, the time spent in query optimization may easily become the major bottleneck in the server's performance.

A more flexible method for query optimization enables the user to control how exhaustive the optimizer is in its search for an optimal query evaluation plan. The general idea is that the fewer plans that are investigated by the optimizer, the less time it spends in compiling a query. On the other hand, because the optimizer skips some plans, it may miss finding an optimal plan.

The behavior of the optimizer with respect to the number of plans it evaluates can be controlled using two system variables:

- The `optimizer_prune_level` variable tells the optimizer to skip certain plans based on estimates of the number of rows accessed for each table. Our experience shows that this kind of “educated guess” rarely misses optimal plans, and may dramatically reduce query compilation times. That is why this option is on (`optimizer_prune_level=1`) by default. However, if you believe that the optimizer missed a better query plan, this option can be switched off (`optimizer_prune_level=0`) with the risk that query compilation may take much longer. Note that, even with the use of this heuristic, the optimizer still explores a roughly exponential number of plans.

- The `optimizer_search_depth` variable tells how far into the “future” of each incomplete plan the optimizer should look to evaluate whether it should be expanded further. Smaller values of `optimizer_search_depth` may result in orders of magnitude smaller query compilation times. For example, queries with 12, 13, or more tables may easily require hours and even days to compile if optimizer_search_depth is close to the number of tables in the query. At the same time, if compiled with optimizer_search_depth equal to 3 or 4, the optimizer may compile in less than a minute for the same query. If you are unsure of what a reasonable value is for optimizer_search_depth, this variable can be set to 0 to tell the optimizer to determine the value automatically.

8.8.6.2 Controlling Switchable Optimizations

The optimizer_switch system variable enables control over optimizer behavior. Its value is a set of flags, each of which has a value of on or off to indicate whether the corresponding optimizer behavior is enabled or disabled. This variable has global and session values and can be changed at runtime. The global default can be set at server startup.

To see the current set of optimizer flags, select the variable value:

```
mysql> SELECT @@optimizer_switch\G
*************************** 1. row ***************************
@@optimizer_switch: index_merge=on,index_merge_union=on,
                    index_merge_sort_union=on,
                    index_merge_intersection=on,
                    engine_condition_pushdown=on,
                    index_condition_pushdown=on,
                    mrr=on,mrr_cost_based=on,
                    block_nested_loop=on,batched_key_access=off,
                    materialization=on,semijoin=on,loosescan=on,
                    firstmatch=on,
                    subquery_materialization_cost_based=on,
                    use_index_extensions=on
```

To change the value of optimizer_switch, assign a value consisting of a comma-separated list of one or more commands:

```
SET [GLOBAL|SESSION] optimizer_switch='command[,command]...';
```

Each *command* value should have one of the forms shown in the following table.

Command Syntax	Meaning
default	Reset every optimization to its default value
opt_name=default	Set the named optimization to its default value
opt_name=off	Disable the named optimization
opt_name=on	Enable the named optimization

The order of the commands in the value does not matter, although the default command is executed first if present. Setting an *opt_name* flag to default sets it to whichever of on or off is its default value. Specifying any given *opt_name* more than once in the value is not permitted and causes an error. Any errors in the value cause the assignment to fail with an error, leaving the value of optimizer_switch unchanged.

The following table lists the permissible *opt_name* flag names, grouped by optimization strategy.

Optimization	Flag Name	Meaning
Batched Key Access	batched_key_access	Controls use of BKA join algorithm
Block Nested-Loop	block_nested_loop	Controls use of BNL join algorithm
Engine Condition Pushdown	engine_condition_pushdown	Controls engine condition pushdown

Optimization	Flag Name	Meaning
Index Condition Pushdown	index_condition_pushdown	Controls index condition pushdown
Index Extensions	use_index_extensions	Controls use of index extensions
Index Merge	index_merge	Controls all Index Merge optimizations
	index_merge_intersection	Controls the Index Merge Intersection Access optimization
	index_merge_sort_union	Controls the Index Merge Sort-Union Access optimization
	index_merge_union	Controls the Index Merge Union Access optimization
Multi-Range Read	mrr	Controls the Multi-Range Read strategy
	mrr_cost_based	Controls use of cost-based MRR if mrr=on
Semi-join	semijoin	Controls all semi-join strategies
	firstmatch	Controls the semi-join FirstMatch strategy
	loosescan	Controls the semi-join LooseScan strategy (not to be confused with LooseScan for GROUP BY)
Subquery materialization	materialization	Controls materialization (including semi-join materialization)
	subquery_materialization_cost_based	Used cost-based materialization choice

For batched_key_access to have any effect when set to on, the mrr flag must also be on. Currently, the cost estimation for MRR is too pessimistic. Hence, it is also necessary for mrr_cost_based to be off for BKA to be used.

The semijoin, firstmatch, loosescan, and materialization flags enable control over semi-join and subquery materialization strategies. The semijoin flag controls whether semi-joins are used. If it is set to on, the firstmatch and loosescan flags enable finer control over the permitted semi-join strategies. The materialization flag controls whether subquery materialization is used. If semijoin and materialization are both on, semi-joins also use materialization where applicable. These flags are on by default.

The subquery_materialization_cost_based enables control over the choice between subquery materialization and IN -> EXISTS subquery transformation. If the flag is on (the default), the optimizer performs a cost-based choice between subquery materialization and IN -> EXISTS subquery transformation if either method could be used. If the flag is off, the optimizer chooses subquery materialization over IN -> EXISTS subquery transformation.

For more information about individual optimization strategies, see the following sections:

- Section 8.2.1.14, “Block Nested-Loop and Batched Key Access Joins”
- Section 8.2.1.5, “Engine Condition Pushdown Optimization”
- Section 8.2.1.7, “Use of Index Extensions”
- Section 8.2.1.6, “Index Condition Pushdown Optimization”
- Section 8.2.1.4, “Index Merge Optimization”

- Section 8.2.1.13, “Multi-Range Read Optimization”
- Section 8.2.1.18, “Subquery Optimization”

When you assign a value to optimizer_switch, flags that are not mentioned keep their current values. This makes it possible to enable or disable specific optimizer behaviors in a single statement without affecting other behaviors. The statement does not depend on what other optimizer flags exist and what their values are. Suppose that all Index Merge optimizations are enabled:

```
mysql> SELECT @@optimizer_switch\G
*************************** 1. row ***************************
@@optimizer_switch: index_merge=on,index_merge_union=on,
                    index_merge_sort_union=on,
                    index_merge_intersection=on,
                    engine_condition_pushdown=on,
                    index_condition_pushdown=on,
                    mrr=on,mrr_cost_based=on,
                    block_nested_loop=on,batched_key_access=off,
                    materialization=on,semijoin=on,loosescan=on,
                    firstmatch=on,
                    subquery_materialization_cost_based=on,
                    use_index_extensions=on
```

If the server is using the Index Merge Union or Index Merge Sort-Union access methods for certain queries and you want to check whether the optimizer will perform better without them, set the variable value like this:

```
mysql> SET optimizer_switch='index_merge_union=off,index_merge_sort_union=off';

mysql> SELECT @@optimizer_switch\G
*************************** 1. row ***************************
@@optimizer_switch: index_merge=on,index_merge_union=off,
                    index_merge_sort_union=off,
                    index_merge_intersection=on,
                    engine_condition_pushdown=on,
                    index_condition_pushdown=on,
                    mrr=on,mrr_cost_based=on,
                    block_nested_loop=on,batched_key_access=off,
                    materialization=on,semijoin=on,loosescan=on,
                    firstmatch=on,
                    subquery_materialization_cost_based=on,
                    use_index_extensions=on
```

8.9 Buffering and Caching

MySQL uses several strategies that cache information in memory buffers to increase performance.

8.9.1 The InnoDB Buffer Pool

InnoDB maintains a storage area called the buffer pool for caching data and indexes in memory. Knowing how the InnoDB buffer pool works, and taking advantage of it to keep frequently accessed data in memory, is an important aspect of MySQL tuning.

Guidelines

Ideally, you set the size of the buffer pool to as large a value as practical, leaving enough memory for other processes on the server to run without excessive paging. The larger the buffer pool, the more InnoDB acts like an in-memory database, reading data from disk once and then accessing the data from memory during subsequent reads. The buffer pool even caches data changed by insert and update operations, so that disk writes can be grouped together for better performance.

Depending on the typical workload on your system, you might adjust the proportions of the parts within the buffer pool. You can tune the way the buffer pool chooses which blocks to cache once it fills up, to keep frequently accessed data in memory despite sudden spikes of activity for operations such as backups or reporting.

With 64-bit systems with large memory sizes, you can split the buffer pool into multiple parts, to minimize contention for the memory structures among concurrent operations. For details, see Improvements to Performance from Multiple Buffer Pools.

Internal Details

InnoDB manages the pool as a list, using a variation of the least recently used (LRU) algorithm. When room is needed to add a new block to the pool, InnoDB evicts the least recently used block and adds the new block to the middle of the list. This “midpoint insertion strategy” treats the list as two sublists:

- At the head, a sublist of “new” (or “young”) blocks that were accessed recently.
- At the tail, a sublist of “old” blocks that were accessed less recently.

This algorithm keeps blocks that are heavily used by queries in the new sublist. The old sublist contains less-used blocks; these blocks are candidates for eviction.

The LRU algorithm operates as follows by default:

- 3/8 of the buffer pool is devoted to the old sublist.
- The midpoint of the list is the boundary where the tail of the new sublist meets the head of the old sublist.
- When InnoDB reads a block into the buffer pool, it initially inserts it at the midpoint (the head of the old sublist). A block can be read in because it is required for a user-specified operation such as an SQL query, or as part of a read-ahead operation performed automatically by InnoDB.
- Accessing a block in the old sublist makes it “young”, moving it to the head of the buffer pool (the head of the new sublist). If the block was read in because it was required, the first access occurs immediately and the block is made young. If the block was read in due to read-ahead, the first access does not occur immediately (and might not occur at all before the block is evicted).
- As the database operates, blocks in the buffer pool that are not accessed “age” by moving toward the tail of the list. Blocks in both the new and old sublists age as other blocks are made new. Blocks in the old sublist also age as blocks are inserted at the midpoint. Eventually, a block that remains unused for long enough reaches the tail of the old sublist and is evicted.

By default, blocks read by queries immediately move into the new sublist, meaning they will stay in the buffer pool for a long time. A table scan (such as performed for a mysqldump operation, or a SELECT statement with no WHERE clause) can bring a large amount of data into the buffer pool and evict an equivalent amount of older data, even if the new data is never used again. Similarly, blocks that are loaded by the read-ahead background thread and then accessed only once move to the head of the new list. These situations can push frequently used blocks to the old sublist, where they become subject to eviction.

Configuration Options

Several InnoDB system variables control the size of the buffer pool and let you tune the LRU algorithm:

- innodb_buffer_pool_size

  Specifies the size of the buffer pool. If your buffer pool is small and you have sufficient memory, making the pool larger can improve performance by reducing the amount of disk I/O needed as queries access InnoDB tables.

- innodb_buffer_pool_instances

  Divides the buffer pool into a user-specified number of separate regions, each with its own LRU list and related data structures, to reduce contention during concurrent memory read and write operations. This option takes effect only when you set the innodb_buffer_pool_size to a size of 1 gigabyte or more. The total size you specify is divided among all the buffer pools. For best efficiency, specify a combination of innodb_buffer_pool_instances and innodb_buffer_pool_size so that each buffer pool instance is at least 1 gigabyte.

- innodb_old_blocks_pct

  Specifies the approximate percentage of the buffer pool that InnoDB uses for the old block sublist. The range of values is 5 to 95. The default value is 37 (that is, 3/8 of the pool).

- innodb_old_blocks_time

  Specifies how long in milliseconds (ms) a block inserted into the old sublist must stay there after its first access before it can be moved to the new sublist. The default value is 0: A block inserted into the old sublist moves to the new sublist when Innodb has evicted 1/4 of the inserted block's pages from the buffer pool, no matter how soon after insertion the access occurs. If the value is greater than 0, blocks remain in the old sublist until an access occurs at least that many ms after the first access. For example, a value of 1000 causes blocks to stay in the old sublist for 1 second after the first access before they become eligible to move to the new sublist.

Setting innodb_old_blocks_time greater than 0 prevents one-time table scans from flooding the new sublist with blocks used only for the scan. Rows in a block read in for a scan are accessed many times in rapid succession, but the block is unused after that. If innodb_old_blocks_time is set to a value greater than time to process the block, the block remains in the “old” sublist and ages to the tail of the list to be evicted quickly. This way, blocks used only for a one-time scan do not act to the detriment of heavily used blocks in the new sublist.

innodb_old_blocks_time can be set at runtime, so you can change it temporarily while performing operations such as table scans and dumps:

```
SET GLOBAL innodb_old_blocks_time = 1000;
... perform queries that scan tables ...
SET GLOBAL innodb_old_blocks_time = 0;
```

This strategy does not apply if your intent is to “warm up” the buffer pool by filling it with a table's content. For example, benchmark tests often perform a table or index scan at server startup, because that data would normally be in the buffer pool after a period of normal use. In this case, leave innodb_old_blocks_time set to 0, at least until the warmup phase is complete.

Monitoring the Buffer Pool

The output from the InnoDB Standard Monitor contains several fields in the BUFFER POOL AND MEMORY section that pertain to operation of the buffer pool LRU algorithm:

- Old database pages: The number of pages in the old sublist of the buffer pool.

- Pages made young, not young: The number of old pages that were moved to the head of the buffer pool (the new sublist), and the number of pages that have remained in the old sublist without being made new.

- youngs/s non-youngs/s: The number of accesses to old pages that have resulted in making them young or not. This metric differs from that of the previous item in two ways. First, it relates only to old pages. Second, it is based on number of accesses to pages and not the number of pages. (There can be multiple accesses to a given page, all of which are counted.)

- `young-making rate`: Hits that cause blocks to move to the head of the buffer pool.
- `not`: Hits that do not cause blocks to move to the head of the buffer pool (due to the delay not being met).

The `young-making` rate and `not` rate will not normally add up to the overall buffer pool hit rate. Hits for blocks in the old sublist cause them to move to the new sublist, but hits to blocks in the new sublist cause them to move to the head of the list only if they are a certain distance from the head.

The preceding information from the Monitor can help you make LRU tuning decisions:

- If you see very low `youngs/s` values when you do not have large scans going on, that indicates that you might need to either reduce the delay time, or increase the percentage of the buffer pool used for the old sublist. Increasing the percentage makes the old sublist larger, so blocks in that sublist take longer to move to the tail and be evicted. This increases the likelihood that they will be accessed again and be made young.
- If you do not see a lot of `non-youngs/s` when you are doing large table scans (and lots of `youngs/s`), to tune your delay value to be larger.

Note

Per second averages provided in `InnoDB` Monitor output are based on the elapsed time between the current time and the last time `InnoDB` Monitor output was printed.

For more information about InnoDB Monitors, see Section 14.2.12.4, “`InnoDB` Monitors”.

8.9.2 The `MyISAM` Key Cache

To minimize disk I/O, the `MyISAM` storage engine exploits a strategy that is used by many database management systems. It employs a cache mechanism to keep the most frequently accessed table blocks in memory:

- For index blocks, a special structure called the *key cache* (or *key buffer*) is maintained. The structure contains a number of block buffers where the most-used index blocks are placed.
- For data blocks, MySQL uses no special cache. Instead it relies on the native operating system file system cache.

This section first describes the basic operation of the `MyISAM` key cache. Then it discusses features that improve key cache performance and that enable you to better control cache operation:

- Multiple sessions can access the cache concurrently.
- You can set up multiple key caches and assign table indexes to specific caches.

To control the size of the key cache, use the `key_buffer_size` system variable. If this variable is set equal to zero, no key cache is used. The key cache also is not used if the `key_buffer_size` value is too small to allocate the minimal number of block buffers (8).

When the key cache is not operational, index files are accessed using only the native file system buffering provided by the operating system. (In other words, table index blocks are accessed using the same strategy as that employed for table data blocks.)

An index block is a contiguous unit of access to the MyISAM index files. Usually the size of an index block is equal to the size of nodes of the index B-tree. (Indexes are represented on disk using a B-tree data structure. Nodes at the bottom of the tree are leaf nodes. Nodes above the leaf nodes are nonleaf nodes.)

All block buffers in a key cache structure are the same size. This size can be equal to, greater than, or less than the size of a table index block. Usually one these two values is a multiple of the other.

When data from any table index block must be accessed, the server first checks whether it is available in some block buffer of the key cache. If it is, the server accesses data in the key cache rather than on disk. That is, it reads from the cache or writes into it rather than reading from or writing to disk. Otherwise, the server chooses a cache block buffer containing a different table index block (or blocks) and replaces the data there by a copy of required table index block. As soon as the new index block is in the cache, the index data can be accessed.

If it happens that a block selected for replacement has been modified, the block is considered “dirty.” In this case, prior to being replaced, its contents are flushed to the table index from which it came.

Usually the server follows an *LRU (Least Recently Used)* strategy: When choosing a block for replacement, it selects the least recently used index block. To make this choice easier, the key cache module maintains all used blocks in a special list (*LRU chain*) ordered by time of use. When a block is accessed, it is the most recently used and is placed at the end of the list. When blocks need to be replaced, blocks at the beginning of the list are the least recently used and become the first candidates for eviction.

The InnoDB storage engine also uses an LRU algorithm, to manage its buffer pool. See Section 8.9.1, “The InnoDB Buffer Pool”.

8.9.2.1 Shared Key Cache Access

Threads can access key cache buffers simultaneously, subject to the following conditions:

- A buffer that is not being updated can be accessed by multiple sessions.
- A buffer that is being updated causes sessions that need to use it to wait until the update is complete.
- Multiple sessions can initiate requests that result in cache block replacements, as long as they do not interfere with each other (that is, as long as they need different index blocks, and thus cause different cache blocks to be replaced).

Shared access to the key cache enables the server to improve throughput significantly.

8.9.2.2 Multiple Key Caches

Shared access to the key cache improves performance but does not eliminate contention among sessions entirely. They still compete for control structures that manage access to the key cache buffers. To reduce key cache access contention further, MySQL also provides multiple key caches. This feature enables you to assign different table indexes to different key caches.

Where there are multiple key caches, the server must know which cache to use when processing queries for a given MyISAM table. By default, all MyISAM table indexes are cached in the default key cache. To assign table indexes to a specific key cache, use the CACHE INDEX statement (see Section 13.7.6.2, “CACHE INDEX Syntax”). For example, the following statement assigns indexes from the tables t1, t2, and t3 to the key cache named hot_cache:

```
mysql> CACHE INDEX t1, t2, t3 IN hot_cache;
+---------+--------------------+----------+----------+
| Table   | Op                 | Msg_type | Msg_text |
+---------+--------------------+----------+----------+
| test.t1 | assign_to_keycache | status   | OK       |
| test.t2 | assign_to_keycache | status   | OK       |
| test.t3 | assign_to_keycache | status   | OK       |
+---------+--------------------+----------+----------+
```

The key cache referred to in a CACHE INDEX statement can be created by setting its size with a SET GLOBAL parameter setting statement or by using server startup options. For example:

```
mysql> SET GLOBAL keycache1.key_buffer_size=128*1024;
```

To destroy a key cache, set its size to zero:

```
mysql> SET GLOBAL keycache1.key_buffer_size=0;
```

Note that you cannot destroy the default key cache. Any attempt to do this will be ignored:

```
mysql> SET GLOBAL key_buffer_size = 0;

mysql> SHOW VARIABLES LIKE 'key_buffer_size';
+-----------------+---------+
| Variable_name   | Value   |
+-----------------+---------+
| key_buffer_size | 8384512 |
+-----------------+---------+
```

Key cache variables are structured system variables that have a name and components. For keycache1.key_buffer_size, keycache1 is the cache variable name and key_buffer_size is the cache component. See Section 5.1.5.1, “Structured System Variables”, for a description of the syntax used for referring to structured key cache system variables.

By default, table indexes are assigned to the main (default) key cache created at the server startup. When a key cache is destroyed, all indexes assigned to it are reassigned to the default key cache.

For a busy server, you can use a strategy that involves three key caches:

- A “hot” key cache that takes up 20% of the space allocated for all key caches. Use this for tables that are heavily used for searches but that are not updated.

- A “cold” key cache that takes up 20% of the space allocated for all key caches. Use this cache for medium-sized, intensively modified tables, such as temporary tables.

- A “warm” key cache that takes up 60% of the key cache space. Employ this as the default key cache, to be used by default for all other tables.

One reason the use of three key caches is beneficial is that access to one key cache structure does not block access to the others. Statements that access tables assigned to one cache do not compete with statements that access tables assigned to another cache. Performance gains occur for other reasons as well:

- The hot cache is used only for retrieval queries, so its contents are never modified. Consequently, whenever an index block needs to be pulled in from disk, the contents of the cache block chosen for replacement need not be flushed first.

- For an index assigned to the hot cache, if there are no queries requiring an index scan, there is a high probability that the index blocks corresponding to nonleaf nodes of the index B-tree remain in the cache.

- An update operation most frequently executed for temporary tables is performed much faster when the updated node is in the cache and need not be read in from disk first. If the size of the indexes of the temporary tables are comparable with the size of cold key cache, the probability is very high that the updated node is in the cache.

The CACHE INDEX statement sets up an association between a table and a key cache, but the association is lost each time the server restarts. If you want the association to take effect each time the server starts, one way to accomplish this is to use an option file: Include variable settings that configure your key caches, and an init-file option that names a file containing CACHE INDEX statements to be executed. For example:

```
key_buffer_size = 4G
hot_cache.key_buffer_size = 2G
cold_cache.key_buffer_size = 2G
init_file=/path/to/data-directory/mysqld_init.sql
```

The statements in mysqld_init.sql are executed each time the server starts. The file should contain one SQL statement per line. The following example assigns several tables each to hot_cache and cold_cache:

```
CACHE INDEX db1.t1, db1.t2, db2.t3 IN hot_cache
CACHE INDEX db1.t4, db2.t5, db2.t6 IN cold_cache
```

8.9.2.3 Midpoint Insertion Strategy

By default, the key cache management system uses a simple LRU strategy for choosing key cache blocks to be evicted, but it also supports a more sophisticated method called the *midpoint insertion strategy.*

When using the midpoint insertion strategy, the LRU chain is divided into two parts: a hot sublist and a warm sublist. The division point between two parts is not fixed, but the key cache management system takes care that the warm part is not “too short,” always containing at least key_cache_division_limit percent of the key cache blocks. key_cache_division_limit is a component of structured key cache variables, so its value is a parameter that can be set per cache.

When an index block is read from a table into the key cache, it is placed at the end of the warm sublist. After a certain number of hits (accesses of the block), it is promoted to the hot sublist. At present, the number of hits required to promote a block (3) is the same for all index blocks.

A block promoted into the hot sublist is placed at the end of the list. The block then circulates within this sublist. If the block stays at the beginning of the sublist for a long enough time, it is demoted to the warm sublist. This time is determined by the value of the key_cache_age_threshold component of the key cache.

The threshold value prescribes that, for a key cache containing *N* blocks, the block at the beginning of the hot sublist not accessed within the last *N* * key_cache_age_threshold / 100 hits is to be moved to the beginning of the warm sublist. It then becomes the first candidate for eviction, because blocks for replacement always are taken from the beginning of the warm sublist.

The midpoint insertion strategy enables you to keep more-valued blocks always in the cache. If you prefer to use the plain LRU strategy, leave the key_cache_division_limit value set to its default of 100.

The midpoint insertion strategy helps to improve performance when execution of a query that requires an index scan effectively pushes out of the cache all the index blocks corresponding to valuable high-level B-tree nodes. To avoid this, you must use a midpoint insertion strategy with the key_cache_division_limit set to much less than 100. Then valuable frequently hit nodes are preserved in the hot sublist during an index scan operation as well.

8.9.2.4 Index Preloading

If there are enough blocks in a key cache to hold blocks of an entire index, or at least the blocks corresponding to its nonleaf nodes, it makes sense to preload the key cache with index blocks before starting to use it. Preloading enables you to put the table index blocks into a key cache buffer in the most efficient way: by reading the index blocks from disk sequentially.

Without preloading, the blocks are still placed into the key cache as needed by queries. Although the blocks will stay in the cache, because there are enough buffers for all of them, they are fetched from disk in random order, and not sequentially.

To preload an index into a cache, use the LOAD INDEX INTO CACHE statement. For example, the following statement preloads nodes (index blocks) of indexes of the tables t1 and t2:

```
mysql> LOAD INDEX INTO CACHE t1, t2 IGNORE LEAVES;
+---------+--------------+----------+----------+
| Table   | Op           | Msg_type | Msg_text |
+---------+--------------+----------+----------+
| test.t1 | preload_keys | status   | OK       |
| test.t2 | preload_keys | status   | OK       |
+---------+--------------+----------+----------+
```

The IGNORE LEAVES modifier causes only blocks for the nonleaf nodes of the index to be preloaded. Thus, the statement shown preloads all index blocks from t1, but only blocks for the nonleaf nodes from t2.

If an index has been assigned to a key cache using a CACHE INDEX statement, preloading places index blocks into that cache. Otherwise, the index is loaded into the default key cache.

8.9.2.5 Key Cache Block Size

It is possible to specify the size of the block buffers for an individual key cache using the key_cache_block_size variable. This permits tuning of the performance of I/O operations for index files.

The best performance for I/O operations is achieved when the size of read buffers is equal to the size of the native operating system I/O buffers. But setting the size of key nodes equal to the size of the I/O buffer does not always ensure the best overall performance. When reading the big leaf nodes, the server pulls in a lot of unnecessary data, effectively preventing reading other leaf nodes.

To control the size of blocks in the .MYI index file of MyISAM tables, use the --myisam-block-size option at server startup.

8.9.2.6 Restructuring a Key Cache

A key cache can be restructured at any time by updating its parameter values. For example:

```
mysql> SET GLOBAL cold_cache.key_buffer_size=4*1024*1024;
```

If you assign to either the key_buffer_size or key_cache_block_size key cache component a value that differs from the component's current value, the server destroys the cache's old structure and creates a new one based on the new values. If the cache contains any dirty blocks, the server saves them to disk before destroying and re-creating the cache. Restructuring does not occur if you change other key cache parameters.

When restructuring a key cache, the server first flushes the contents of any dirty buffers to disk. After that, the cache contents become unavailable. However, restructuring does not block queries that need to use indexes assigned to the cache. Instead, the server directly accesses the table indexes using native file system caching. File system caching is not as efficient as using a key cache, so although queries execute, a slowdown can be anticipated. After the cache has been restructured, it becomes available again for caching indexes assigned to it, and the use of file system caching for the indexes ceases.

8.9.3 The MySQL Query Cache

The query cache stores the text of a `SELECT` statement together with the corresponding result that was sent to the client. If an identical statement is received later, the server retrieves the results from the query cache rather than parsing and executing the statement again. The query cache is shared among sessions, so a result set generated by one client can be sent in response to the same query issued by another client.

The query cache can be useful in an environment where you have tables that do not change very often and for which the server receives many identical queries. This is a typical situation for many Web servers that generate many dynamic pages based on database content.

The query cache does not return stale data. When tables are modified, any relevant entries in the query cache are flushed.

> **Note**
>
> The query cache does not work in an environment where you have multiple `mysqld` servers updating the same `MyISAM` tables.

The query cache is used for prepared statements under the conditions described in Section 8.9.3.1, “How the Query Cache Operates”.

> **Note**
>
> The query cache is not supported for partitioned tables, and is automatically disabled for queries involving partitioned tables. The query cache cannot be enabled for such queries.

Some performance data for the query cache follows. These results were generated by running the MySQL benchmark suite on a Linux Alpha 2×500MHz system with 2GB RAM and a 64MB query cache.

- If all the queries you are performing are simple (such as selecting a row from a table with one row), but still differ so that the queries cannot be cached, the overhead for having the query cache active is 13%. This could be regarded as the worst case scenario. In real life, queries tend to be much more complicated, so the overhead normally is significantly lower.
- Searches for a single row in a single-row table are 238% faster with the query cache than without it. This can be regarded as close to the minimum speedup to be expected for a query that is cached.

To disable the query cache at server startup, set the `query_cache_size` system variable to 0. By disabling the query cache code, there is no noticeable overhead.

The query cache offers the potential for substantial performance improvement, but do not assume that it will do so under all circumstances. With some query cache configurations or server workloads, you might actually see a performance decrease:

- Be cautious about sizing the query cache excessively large, which increases the overhead required to maintain the cache, possibly beyond the benefit of enabling it. Sizes in tens of megabytes are usually beneficial. Sizes in the hundreds of megabytes might not be.

- Server workload has a significant effect on query cache efficiency. A query mix consisting almost entirely of a fixed set of SELECT statements is much more likely to benefit from enabling the cache than a mix in which frequent INSERT statements cause continual invalidation of results in the cache. In some cases, a workaround is to use the SQL_NO_CACHE option to prevent results from even entering the cache for SELECT statements that use frequently modified tables. (See Section 8.9.3.2, "Query Cache SELECT Options".)

To verify that enabling the query cache is beneficial, test the operation of your MySQL server with the cache enabled and disabled. Then retest periodically because query cache efficiency may change as server workload changes.

8.9.3.1 How the Query Cache Operates

This section describes how the query cache works when it is operational. Section 8.9.3.3, "Query Cache Configuration", describes how to control whether it is operational.

Incoming queries are compared to those in the query cache before parsing, so the following two queries are regarded as different by the query cache:

```
SELECT * FROM tbl_name
Select * from tbl_name
```

Queries must be *exactly* the same (byte for byte) to be seen as identical. In addition, query strings that are identical may be treated as different for other reasons. Queries that use different databases, different protocol versions, or different default character sets are considered different queries and are cached separately.

The cache is not used for queries of the following types:

- Queries that are a subquery of an outer query

- Queries executed within the body of a stored function, trigger, or event

Before a query result is fetched from the query cache, MySQL checks whether the user has SELECT privilege for all databases and tables involved. If this is not the case, the cached result is not used.

If a query result is returned from query cache, the server increments the Qcache_hits status variable, not Com_select. See Section 8.9.3.4, "Query Cache Status and Maintenance".

If a table changes, all cached queries that use the table become invalid and are removed from the cache. This includes queries that use MERGE tables that map to the changed table. A table can be changed by many types of statements, such as INSERT, UPDATE, DELETE, TRUNCATE TABLE, ALTER TABLE, DROP TABLE, or DROP DATABASE.

The query cache also works within transactions when using InnoDB tables.

In MySQL 5.7, the result from a SELECT query on a view is cached.

The query cache works for SELECT SQL_CALC_FOUND_ROWS ... queries and stores a value that is returned by a following SELECT FOUND_ROWS() query. FOUND_ROWS() returns the correct value even if the preceding query was fetched from the cache because the number of found rows is also stored in the cache. The SELECT FOUND_ROWS() query itself cannot be cached.

Prepared statements that are issued using the binary protocol using mysql_stmt_prepare() and mysql_stmt_execute() (see Section 21.8.8, “C API Prepared Statements”), are subject to limitations on caching. Comparison with statements in the query cache is based on the text of the statement after expansion of ? parameter markers. The statement is compared only with other cached statements that were executed using the binary protocol. That is, for query cache purposes, prepared statements issued using the binary protocol are distinct from prepared statements issued using the text protocol (see Section 13.5, “SQL Syntax for Prepared Statements”).

A query cannot be cached if it contains any of the functions shown in the following table.

AES_DECRYPT() (as of 5.7.4)	AES_ENCRYPT() (as of 5.7.4)	BENCHMARK()
CONNECTION_ID()	CONVERT_TZ()	CURDATE()
CURRENT_DATE()	CURRENT_TIME()	CURRENT_TIMESTAMP()
CURTIME()	DATABASE()	ENCRYPT() with one parameter
FOUND_ROWS()	GET_LOCK()	LAST_INSERT_ID()
LOAD_FILE()	MASTER_POS_WAIT()	NOW()
PASSWORD()	RAND()	RANDOM_BYTES()
RELEASE_LOCK()	SLEEP()	SYSDATE()
UNIX_TIMESTAMP() with no parameters	USER()	UUID()
UUID_SHORT()		

A query also is not cached under these conditions:

- It refers to user-defined functions (UDFs) or stored functions.
- It refers to user variables or local stored program variables.
- It refers to tables in the mysql, INFORMATION_SCHEMA, or performance_schema database.
- It refers to any partitioned tables.
- It is of any of the following forms:

  ```
  SELECT ... LOCK IN SHARE MODE
  SELECT ... FOR UPDATE
  SELECT ... INTO OUTFILE ...
  SELECT ... INTO DUMPFILE ...
  SELECT * FROM ... WHERE autoincrement_col IS NULL
  ```

  The last form is not cached because it is used as the ODBC workaround for obtaining the last insert ID value. See the Connector/ODBC section of Chapter 21, *Connectors and APIs*.

  Statements within transactions that use SERIALIZABLE isolation level also cannot be cached because they use LOCK IN SHARE MODE locking.
- It uses TEMPORARY tables.
- It does not use any tables.
- It generates warnings.

- The user has a column-level privilege for any of the involved tables.

8.9.3.2 Query Cache SELECT Options

Two query cache-related options may be specified in SELECT statements:

- SQL_CACHE

  The query result is cached if it is cacheable and the value of the query_cache_type system variable is ON or DEMAND.

- SQL_NO_CACHE

  The server does not use the query cache. It neither checks the query cache to see whether the result is already cached, nor does it cache the query result.

Examples:

```
SELECT SQL_CACHE id, name FROM customer;
SELECT SQL_NO_CACHE id, name FROM customer;
```

8.9.3.3 Query Cache Configuration

The have_query_cache server system variable indicates whether the query cache is available:

```
mysql> SHOW VARIABLES LIKE 'have_query_cache';
+------------------+-------+
| Variable_name    | Value |
+------------------+-------+
| have_query_cache | YES   |
+------------------+-------+
```

When using a standard MySQL binary, this value is always YES, even if query caching is disabled.

Several other system variables control query cache operation. These can be set in an option file or on the command line when starting mysqld. The query cache system variables all have names that begin with query_cache_. They are described briefly in Section 5.1.4, “Server System Variables”, with additional configuration information given here.

To set the size of the query cache, set the query_cache_size system variable. Setting it to 0 disables the query cache, as does setting query_cache_type=0. By default, the query cache is disabled. This is achieved using a default size of 1M, with a default for query_cache_type of 0.

To reduce overhead significantly, also start the server with query_cache_type=0 if you will not be using the query cache.

Note

When using the Windows Configuration Wizard to install or configure MySQL, the default value for query_cache_size will be configured automatically for you based on the different configuration types available. When using the Windows Configuration Wizard, the query cache may be enabled (that is, set to a nonzero value) due to the selected configuration. The query cache is also controlled by the setting of the query_cache_type variable. Check the values of these variables as set in your my.ini file after configuration has taken place.

When you set query_cache_size to a nonzero value, keep in mind that the query cache needs a minimum size of about 40KB to allocate its structures. (The exact size depends on system architecture.) If you set the value too small, you'll get a warning, as in this example:

```
mysql> SET GLOBAL query_cache_size = 40000;
Query OK, 0 rows affected, 1 warning (0.00 sec)

mysql> SHOW WARNINGS\G
*************************** 1. row ***************************
  Level: Warning
   Code: 1282
Message: Query cache failed to set size 39936;
         new query cache size is 0

mysql> SET GLOBAL query_cache_size = 41984;
Query OK, 0 rows affected (0.00 sec)

mysql> SHOW VARIABLES LIKE 'query_cache_size';
+------------------+-------+
| Variable_name    | Value |
+------------------+-------+
| query_cache_size | 41984 |
+------------------+-------+
```

For the query cache to actually be able to hold any query results, its size must be set larger:

```
mysql> SET GLOBAL query_cache_size = 1000000;
Query OK, 0 rows affected (0.04 sec)

mysql> SHOW VARIABLES LIKE 'query_cache_size';
+------------------+--------+
| Variable_name    | Value  |
+------------------+--------+
| query_cache_size | 999424 |
+------------------+--------+
1 row in set (0.00 sec)
```

The query_cache_size value is aligned to the nearest 1024 byte block. The value reported may therefore be different from the value that you assign.

If the query cache size is greater than 0, the query_cache_type variable influences how it works. This variable can be set to the following values:

- A value of 0 or OFF prevents caching or retrieval of cached results.
- A value of 1 or ON enables caching except of those statements that begin with SELECT SQL_NO_CACHE.
- A value of 2 or DEMAND causes caching of only those statements that begin with SELECT SQL_CACHE.

If query_cache_size is 0, you should also set query_cache_type variable to 0. In this case, the server does not acquire the query cache mutex at all, which means that the query cache cannot be enabled at runtime and there is reduced overhead in query execution.

Setting the GLOBAL query_cache_type value determines query cache behavior for all clients that connect after the change is made. Individual clients can control cache behavior for their own connection by setting the SESSION query_cache_type value. For example, a client can disable use of the query cache for its own queries like this:

```
mysql> SET SESSION query_cache_type = OFF;
```

If you set query_cache_type at server startup (rather than at runtime with a SET statement), only the numeric values are permitted.

To control the maximum size of individual query results that can be cached, set the query_cache_limit system variable. The default value is 1MB.

Be careful not to set the size of the cache too large. Due to the need for threads to lock the cache during updates, you may see lock contention issues with a very large cache.

Note

You can set the maximum size that can be specified for the query cache at runtime with the SET statement by using the --maximum-query_cache_size=*32M* option on the command line or in the configuration file.

When a query is to be cached, its result (the data sent to the client) is stored in the query cache during result retrieval. Therefore the data usually is not handled in one big chunk. The query cache allocates blocks for storing this data on demand, so when one block is filled, a new block is allocated. Because memory allocation operation is costly (timewise), the query cache allocates blocks with a minimum size given by the query_cache_min_res_unit system variable. When a query is executed, the last result block is trimmed to the actual data size so that unused memory is freed. Depending on the types of queries your server executes, you might find it helpful to tune the value of query_cache_min_res_unit:

- The default value of query_cache_min_res_unit is 4KB. This should be adequate for most cases.
- If you have a lot of queries with small results, the default block size may lead to memory fragmentation, as indicated by a large number of free blocks. Fragmentation can force the query cache to prune (delete) queries from the cache due to lack of memory. In this case, decrease the value of query_cache_min_res_unit. The number of free blocks and queries removed due to pruning are given by the values of the Qcache_free_blocks and Qcache_lowmem_prunes status variables.
- If most of your queries have large results (check the Qcache_total_blocks and Qcache_queries_in_cache status variables), you can increase performance by increasing query_cache_min_res_unit. However, be careful to not make it too large (see the previous item).

8.9.3.4 Query Cache Status and Maintenance

To check whether the query cache is present in your MySQL server, use the following statement:

```
mysql> SHOW VARIABLES LIKE 'have_query_cache';
+------------------+-------+
| Variable_name    | Value |
+------------------+-------+
| have_query_cache | YES   |
+------------------+-------+
```

You can defragment the query cache to better utilize its memory with the FLUSH QUERY CACHE statement. The statement does not remove any queries from the cache.

The RESET QUERY CACHE statement removes all query results from the query cache. The FLUSH TABLES statement also does this.

To monitor query cache performance, use SHOW STATUS to view the cache status variables:

```
mysql> SHOW STATUS LIKE 'Qcache%';
+-------------------------+--------+
| Variable_name           | Value  |
+-------------------------+--------+
| Qcache_free_blocks      | 36     |
| Qcache_free_memory      | 138488 |
| Qcache_hits             | 79570  |
| Qcache_inserts          | 27087  |
| Qcache_lowmem_prunes    | 3114   |
| Qcache_not_cached       | 22989  |
| Qcache_queries_in_cache | 415    |
| Qcache_total_blocks     | 912    |
+-------------------------+--------+
```

Descriptions of each of these variables are given in Section 5.1.6, “Server Status Variables”. Some uses for them are described here.

The total number of SELECT queries is given by this formula:

```
  Com_select
+ Qcache_hits
+ queries with errors found by parser
```

The Com_select value is given by this formula:

```
  Qcache_inserts
+ Qcache_not_cached
+ queries with errors found during the column-privileges check
```

The query cache uses variable-length blocks, so Qcache_total_blocks and Qcache_free_blocks may indicate query cache memory fragmentation. After FLUSH QUERY CACHE, only a single free block remains.

Every cached query requires a minimum of two blocks (one for the query text and one or more for the query results). Also, every table that is used by a query requires one block. However, if two or more queries use the same table, only one table block needs to be allocated.

The information provided by the Qcache_lowmem_prunes status variable can help you tune the query cache size. It counts the number of queries that have been removed from the cache to free up memory for caching new queries. The query cache uses a least recently used (LRU) strategy to decide which queries to remove from the cache. Tuning information is given in Section 8.9.3.3, “Query Cache Configuration”.

8.9.4 Caching of Prepared Statements and Stored Programs

For certain statements that a client might execute multiple times during a session, the server converts the statement to an internal structure and caches that structure to be used during execution. Caching enables the server to perform more efficiently because it avoids the overhead of reconverting the statement should it be needed again during the session. Conversion and caching occurs for these statements:

- Prepared statements, both those processed at the SQL level (using the PREPARE statement) and those processed using the binary client/server protocol (using the mysql_stmt_prepare() C API function). The max_prepared_stmt_count system variable controls the total number of statements the server caches. (The sum of the number of prepared statements across all sessions.)

- Stored programs (stored procedures and functions, triggers, and events). In this case, the server converts and caches the entire program body. The stored_program_cache system variable indicates the approximate number of stored programs the the server caches per session.

The server maintains caches for prepared statements and stored programs on a per-session basis. Statements cached for one session are not accessible to other sessions. When a session ends, the server discards any statements cached for it.

When the server uses a cached internal statement structure, it must take care that the structure does not go out of date. Metadata changes can occur for an object used by the statement, causing a mismatch between the current object definition and the definition as represented in the internal statement structure. Metadata changes occur for DDL statements such as those that create, drop, alter, rename, or truncate tables, or that analyze, optimize, or repair tables. Table content changes (for example, with INSERT or UPDATE) do not change metadata, nor do SELECT statements.

Here is an illustration of the problem. Suppose that a client prepares this statement:

```
PREPARE s1 FROM 'SELECT * FROM t1';
```

The SELECT * expands in the internal structure to the list of columns in the table. If the set of columns in the table is modified with ALTER TABLE, the prepared statement goes out of date. If the server does not detect this change the next time the client executes s1, the prepared statement will return incorrect results.

To avoid problems caused by metadata changes to tables or views referred to by the prepared statement, the server detects these changes and automatically reprepares the statement when it is next executed. That is, the server reparses the statement and rebuilds the internal structure. Reparsing also occurs after referenced tables or views are flushed from the table definition cache, either implicitly to make room for new entries in the cache, or explicitly due to FLUSH TABLES.

Similarly, if changes occur to objects used by a stored program, the server reparses affected statements within the program.

The server also detects metadata changes for objects in expressions. These might be used in statements specific to stored programs, such as DECLARE CURSOR or flow-control statements such as IF, CASE, and RETURN.

To avoid reparsing entire stored programs, the server reparses affected statements or expressions within a program only as needed. Examples:

- Suppose that metadata for a table or view is changed. Reparsing occurs for a SELECT * within the program that accesses the table or view, but not for a SELECT * that does not access the table or view.
- When a statement is affected, the server reparses it only partially if possible. Consider this CASE statement:

  ```
  CASE case_expr
    WHEN when_expr1 ...
    WHEN when_expr2 ...
    WHEN when_expr3 ...
    ...
  END CASE
  ```

  If a metadata change affects only WHEN *when_expr3*, that expression is reparsed. *case_expr* and the other WHEN expressions are not reparsed.

Reparsing uses the default database and SQL mode that were in effect for the original conversion to internal form.

The server attempts reparsing up to three times. An error occurs if all attempts fail.

Reparsing is automatic, but to the extent that it occurs, diminishes prepared statement and stored program performance.

For prepared statements, the Com_stmt_reprepare status variable tracks the number of repreparations.

8.10 Optimizing Locking Operations

MySQL manages contention for table contents using locking:

- Internal locking is performed within the MySQL server itself to manage contention for table contents by multiple threads. This type of locking is internal because it is performed entirely by the server and involves no other programs. See Section 8.10.1, “Internal Locking Methods”.

- External locking occurs when the server and other programs lock `MyISAM` table files to coordinate among themselves which program can access the tables at which time. See Section 8.10.5, “External Locking”.

8.10.1 Internal Locking Methods

This section discusses internal locking; that is, locking performed within the MySQL server itself to manage contention for table contents by multiple sessions. This type of locking is internal because it is performed entirely by the server and involves no other programs. For locking performed on MySQL files by other programs, see Section 8.10.5, “External Locking”.

Row-Level Locking

MySQL uses row-level locking for `InnoDB` tables to support simultaneous write access by multiple sessions, making them suitable for multi-user, highly concurrent, and OLTP applications.

To avoid deadlocks when performing multiple concurrent write operations on a single `InnoDB` table, acquire necessary locks at the start of the transaction by issuing a `SELECT ... FOR UPDATE` statement for each group of rows expected to be modified, even if the DML statements come later in the transaction. If transactions modify or lock more than one table, issue the applicable statements in the same order within each transaction. Deadlocks affect performance rather than representing a serious error, because `InnoDB` automatically detects deadlock conditions and rolls back one of the affected transactions.

Advantages of row-level locking:

- Fewer lock conflicts when different sessions access different rows.

- Fewer changes for rollbacks.

- Possible to lock a single row for a long time.

Table-Level Locking

MySQL uses table-level locking for `MyISAM`, `MEMORY`, and `MERGE` tables, allowing only one session to update those tables at a time, making them more suitable for read-only, read-mostly, or single-user applications.

These storage engines avoid deadlocks by always requesting all needed locks at once at the beginning of a query and always locking the tables in the same order. The tradeoff is that this strategy reduces concurrency; other sessions that want to modify the table must wait until the current DML statement finishes.

MySQL grants table write locks as follows:

1. If there are no locks on the table, put a write lock on it.

2. Otherwise, put the lock request in the write lock queue.

MySQL grants table read locks as follows:

1. If there are no write locks on the table, put a read lock on it.

2. Otherwise, put the lock request in the read lock queue.

Table updates are given higher priority than table retrievals. Therefore, when a lock is released, the lock is made available to the requests in the write lock queue and then to the requests in the read lock queue. This ensures that updates to a table are not “starved” even if there is heavy SELECT activity for the table. However, if you have many updates for a table, SELECT statements wait until there are no more updates.

For information on altering the priority of reads and writes, see Section 8.10.2, “Table Locking Issues”.

You can analyze the table lock contention on your system by checking the Table_locks_immediate and Table_locks_waited status variables, which indicate the number of times that requests for table locks could be granted immediately and the number that had to wait, respectively:

```
mysql> SHOW STATUS LIKE 'Table%';
+-----------------------+---------+
| Variable_name         | Value   |
+-----------------------+---------+
| Table_locks_immediate | 1151552 |
| Table_locks_waited    | 15324   |
+-----------------------+---------+
```

The MyISAM storage engine supports concurrent inserts to reduce contention between readers and writers for a given table: If a MyISAM table has no free blocks in the middle of the data file, rows are always inserted at the end of the data file. In this case, you can freely mix concurrent INSERT and SELECT statements for a MyISAM table without locks. That is, you can insert rows into a MyISAM table at the same time other clients are reading from it. Holes can result from rows having been deleted from or updated in the middle of the table. If there are holes, concurrent inserts are disabled but are enabled again automatically when all holes have been filled with new data.. This behavior is altered by the concurrent_insert system variable. See Section 8.10.3, “Concurrent Inserts”.

If you acquire a table lock explicitly with LOCK TABLES, you can request a READ LOCAL lock rather than a READ lock to enable other sessions to perform concurrent inserts while you have the table locked.

To perform many INSERT and SELECT operations on a table real_table when concurrent inserts are not possible, you can insert rows into a temporary table temp_table and update the real table with the rows from the temporary table periodically. This can be done with the following code:

```
mysql> LOCK TABLES real_table WRITE, temp_table WRITE;
mysql> INSERT INTO real_table SELECT * FROM temp_table;
mysql> DELETE FROM temp_table;
mysql> UNLOCK TABLES;
```

Advantages of table-level locking:

- Requires relatively little memory.

- Fast when used on a large part of the table because only a single lock is involved.

- Fast if you often do GROUP BY operations on a large part of the data or if you must scan the entire table frequently.

Generally, table locks are suited to the following cases:

- Most statements for the table are reads.

- Statements for the table are a mix of reads and writes, where writes are updates or deletes for a single row that can be fetched with one key read:

```
UPDATE tbl_name SET column=value WHERE unique_key_col=key_value;
DELETE FROM tbl_name WHERE unique_key_col=key_value;
```

- SELECT combined with concurrent INSERT statements, and very few UPDATE or DELETE statements.
- Many scans or GROUP BY operations on the entire table without any writers.

8.10.2 Table Locking Issues

InnoDB tables use row-level locking so that multiple sessions and applications can read from and write to the same table simultaneously, without making each other wait or producing inconsistent results. For this storage engine, avoid using the LOCK TABLES statement, because it does not offer any extra protection, but instead reduces concurrency. The automatic row-level locking makes these tables suitable for your busiest databases with your most important data, while also simplifying application logic since you do not need to lock and unlock tables. Consequently, the InnoDB storage engine is the default in MySQL 5.7.

MySQL uses table locking (instead of page, row, or column locking) for all storage engines except InnoDB. The locking operations themselves do not have much overhead. But because only one session can write to a table at any one time, for best performance with these other storage engines, use them primarily for tables that are queried often and rarely inserted into or updated.

Performance Considerations Favoring InnoDB

When choosing whether to create a table using InnoDB or a different storage engine, keep in mind the following disadvantages of table locking:

- Table locking enables many sessions to read from a table at the same time, but if a session wants to write to a table, it must first get exclusive access, meaning it might have to wait for other sessions to finish with the table first. During the update, all other sessions that want to access this particular table must wait until the update is done.
- Table locking causes problems when a session is waiting because the disk is full and free space needs to become available before the session can proceed. In this case, all sessions that want to access the problem table are also put in a waiting state until more disk space is made available.
- A SELECT statement that takes a long time to run prevents other sessions from updating the table in the meantime, making the other sessions appear slow or unresponsive. While a session is waiting to get exclusive access to the table for updates, other sessions that issue SELECT statements will queue up behind it, reducing concurrency even for read-only sessions.

Workarounds for Locking Performance Issues

The following items describe some ways to avoid or reduce contention caused by table locking:

- Consider switching the table to the InnoDB storage engine, either using CREATE TABLE ... ENGINE=INNODB during setup, or using ALTER TABLE ... ENGINE=INNODB for an existing table. See Section 14.2, “The InnoDB Storage Engine” for more details about this storage engine.
- Optimize SELECT statements to run faster so that they lock tables for a shorter time. You might have to create some summary tables to do this.
- Start mysqld with --low-priority-updates. For storage engines that use only table-level locking (such as MyISAM, MEMORY, and MERGE), this gives all statements that update (modify) a table lower

priority than SELECT statements. In this case, the second SELECT statement in the preceding scenario would execute before the UPDATE statement, and would not wait for the first SELECT to finish.

- To specify that all updates issued in a specific connection should be done with low priority, set the low_priority_updates server system variable equal to 1.
- To give a specific INSERT, UPDATE, or DELETE statement lower priority, use the LOW_PRIORITY attribute.
- To give a specific SELECT statement higher priority, use the HIGH_PRIORITY attribute. See Section 13.2.9, "SELECT Syntax".
- Start mysqld with a low value for the max_write_lock_count system variable to force MySQL to temporarily elevate the priority of all SELECT statements that are waiting for a table after a specific number of inserts to the table occur. This permits READ locks after a certain number of WRITE locks.
- If you have problems with INSERT combined with SELECT, consider switching to MyISAM tables, which support concurrent SELECT and INSERT statements. (See Section 8.10.3, "Concurrent Inserts".)
- If you have problems with mixed SELECT and DELETE statements, the LIMIT option to DELETE may help. See Section 13.2.2, "DELETE Syntax".
- Using SQL_BUFFER_RESULT with SELECT statements can help to make the duration of table locks shorter. See Section 13.2.9, "SELECT Syntax".
- Splitting table contents into separate tables may help, by allowing queries to run against columns in one table, while updates are confined to columns in a different table.
- You could change the locking code in mysys/thr_lock.c to use a single queue. In this case, write locks and read locks would have the same priority, which might help some applications.

8.10.3 Concurrent Inserts

The MyISAM storage engine supports concurrent inserts to reduce contention between readers and writers for a given table: If a MyISAM table has no holes in the data file (deleted rows in the middle), an INSERT statement can be executed to add rows to the end of the table at the same time that SELECT statements are reading rows from the table. If there are multiple INSERT statements, they are queued and performed in sequence, concurrently with the SELECT statements. The results of a concurrent INSERT may not be visible immediately.

The concurrent_insert system variable can be set to modify the concurrent-insert processing. By default, the variable is set to AUTO (or 1) and concurrent inserts are handled as just described. If concurrent_insert is set to NEVER (or 0), concurrent inserts are disabled. If the variable is set to ALWAYS (or 2), concurrent inserts at the end of the table are permitted even for tables that have deleted rows. See also the description of the concurrent_insert system variable.

If you are using the binary log, concurrent inserts are converted to normal inserts for CREATE ... SELECT or INSERT ... SELECT statements. This is done to ensure that you can re-create an exact copy of your tables by applying the log during a backup operation. See Section 5.2.4, "The Binary Log". In addition, for those statements a read lock is placed on the selected-from table such that inserts into that table are blocked. The effect is that concurrent inserts for that table must wait as well.

With LOAD DATA INFILE, if you specify CONCURRENT with a MyISAM table that satisfies the condition for concurrent inserts (that is, it contains no free blocks in the middle), other sessions can retrieve data from the table while LOAD DATA is executing. Use of the CONCURRENT option affects the performance of LOAD DATA a bit, even if no other session is using the table at the same time.

If you specify HIGH_PRIORITY, it overrides the effect of the --low-priority-updates option if the server was started with that option. It also causes concurrent inserts not to be used.

For LOCK TABLE, the difference between READ LOCAL and READ is that READ LOCAL permits nonconflicting INSERT statements (concurrent inserts) to execute while the lock is held. However, this cannot be used if you are going to manipulate the database using processes external to the server while you hold the lock.

8.10.4 Metadata Locking

MySQL uses metadata locking to manage access to objects (tables, triggers, and so forth). Metadata locking is used to ensure data consistency but does involve some overhead, which increases as query volume increases. Metadata contention increases the more that multiple queries attempt to access the same objects.

Metadata locking is not a replacement for the table definition case, and its mutexes and locks differ from the LOCK_open mutex. The following discussion provides some information about how metadata locking works.

To ensure transaction serializability, the server must not permit one session to perform a data definition language (DDL) statement on a table that is used in an uncompleted transaction in another session. The server achieves this by acquiring metadata locks on tables used within a transaction and deferring release of those locks until the transaction ends. A metadata lock on a table prevents changes to the table's structure. This locking approach has the implication that a table that is being used by a transaction within one session cannot be used in DDL statements by other sessions until the transaction ends.

This principle applies not only to transactional tables, but also to nontransactional tables. Suppose that a session begins a transaction that uses transactional table t and nontransactional table nt as follows:

```
START TRANSACTION;
SELECT * FROM t;
SELECT * FROM nt;
```

Metadata locks are held on both t and nt until the transaction ends. If another session attempts a DDL operation on either table, it blocks until metadata lock release at transaction end. For example, a second session blocks if it attempts any of these operations:

```
DROP TABLE t;
ALTER TABLE t ...;
DROP TABLE nt;
ALTER TABLE nt ...;
```

If the server acquires metadata locks for a statement that is syntactically valid but fails during execution, it does not release the locks early. Lock release is still deferred to the end of the transaction because the failed statement is written to the binary log and the locks protect log consistency.

In autocommit mode, each statement is in effect a complete transaction, so metadata locks acquired for the statement are held only to the end of the statement.

Metadata locks acquired during a PREPARE statement are released once the statement has been prepared, even if preparation occurs within a multiple-statement transaction.

Before MySQL 5.5, when a transaction acquired the equivalent of a metadata lock for a table used within a statement, it released the lock at the end of the statement. This approach had the disadvantage that if a DDL statement occurred for a table that was being used by another session in an active transaction, statements could be written to the binary log in the wrong order.

8.10.5 External Locking

External locking is the use of file system locking to manage contention for MyISAM database tables by multiple processes. External locking is used in situations where a single process such as the MySQL server cannot be assumed to be the only process that requires access to tables. Here are some examples:

- If you run multiple servers that use the same database directory (not recommended), each server must have external locking enabled.

- If you use myisamchk to perform table maintenance operations on MyISAM tables, you must either ensure that the server is not running, or that the server has external locking enabled so that it locks table files as necessary to coordinate with myisamchk for access to the tables. The same is true for use of myisampack to pack MyISAM tables.

  If the server is run with external locking enabled, you can use myisamchk at any time for read operations such a checking tables. In this case, if the server tries to update a table that myisamchk is using, the server will wait for myisamchk to finish before it continues.

  If you use myisamchk for write operations such as repairing or optimizing tables, or if you use myisampack to pack tables, you *must* always ensure that the mysqld server is not using the table. If you don't stop mysqld, at least do a mysqladmin flush-tables before you run myisamchk. Your tables *may become corrupted* if the server and myisamchk access the tables simultaneously.

With external locking in effect, each process that requires access to a table acquires a file system lock for the table files before proceeding to access the table. If all necessary locks cannot be acquired, the process is blocked from accessing the table until the locks can be obtained (after the process that currently holds the locks releases them).

External locking affects server performance because the server must sometimes wait for other processes before it can access tables.

External locking is unnecessary if you run a single server to access a given data directory (which is the usual case) and if no other programs such as myisamchk need to modify tables while the server is running. If you only *read* tables with other programs, external locking is not required, although myisamchk might report warnings if the server changes tables while myisamchk is reading them.

With external locking disabled, to use myisamchk, you must either stop the server while myisamchk executes or else lock and flush the tables before running myisamchk. (See Section 8.11.1, “System Factors and Startup Parameter Tuning”.) To avoid this requirement, use the CHECK TABLE and REPAIR TABLE statements to check and repair MyISAM tables.

For mysqld, external locking is controlled by the value of the skip_external_locking system variable. When this variable is enabled, external locking is disabled, and vice versa. From MySQL 4.0 on, external locking is disabled by default.

Use of external locking can be controlled at server startup by using the --external-locking or --skip-external-locking option.

If you do use external locking option to enable updates to MyISAM tables from many MySQL processes, you must ensure that the following conditions are satisfied:

- Do not use the query cache for queries that use tables that are updated by another process.

- Do not start the server with the --delay-key-write=ALL option or use the DELAY_KEY_WRITE=1 table option for any shared tables. Otherwise, index corruption can occur.

The easiest way to satisfy these conditions is to always use `--external-locking` together with `--delay-key-write=OFF` and `--query-cache-size=0`. (This is not done by default because in many setups it is useful to have a mixture of the preceding options.)

8.11 Optimizing the MySQL Server

This section discusses optimization techniques for the database server, primarily dealing with system configuration rather than tuning SQL statements. The information in this section is appropriate for DBAs who want to ensure performance and scalability across the servers they manage; for developers constructing installation scripts that include setting up the database; and people running MySQL themselves for development, testing, and so on who want to maximize their own productivity.

8.11.1 System Factors and Startup Parameter Tuning

We start with system-level factors, because some of these decisions must be made very early to achieve large performance gains. In other cases, a quick look at this section may suffice. However, it is always nice to have a sense of how much can be gained by changing factors that apply at this level.

Before using MySQL in production, we advise you to test it on your intended platform.

Other tips:

- If you have enough RAM, you could remove all swap devices. Some operating systems use a swap device in some contexts even if you have free memory.

- Avoid external locking for `MyISAM` tables. Since MySQL 4.0, the default has been for external locking to be disabled on all systems. The `--external-locking` and `--skip-external-locking` options explicitly enable and disable external locking.

  Note that disabling external locking does not affect MySQL's functionality as long as you run only one server. Just remember to take down the server (or lock and flush the relevant tables) before you run `myisamchk`. On some systems it is mandatory to disable external locking because it does not work, anyway.

  The only case in which you cannot disable external locking is when you run multiple MySQL *servers* (not clients) on the same data, or if you run `myisamchk` to check (not repair) a table without telling the server to flush and lock the tables first. Note that using multiple MySQL servers to access the same data concurrently is generally *not* recommended, except when using MySQL Cluster.

  > **Note**
  >
  > MySQL Cluster is currently not supported in MySQL 5.7. Users wishing to upgrade a MySQL Cluster from MySQL 5.0 or 5.1 should instead migrate to MySQL Cluster NDB 7.0 or 7.1; these are based on MySQL 5.1 but contain the latest improvements and fixes for `NDB`.

  The `LOCK TABLES` and `UNLOCK TABLES` statements use internal locking, so you can use them even if external locking is disabled.

8.11.2 Tuning Server Parameters

You can determine the default buffer sizes used by the `mysqld` server using this command:

```
shell> mysqld --verbose --help
```

This command produces a list of all `mysqld` options and configurable system variables. The output includes the default variable values and looks something like this:

```
abort-slave-event-count           0
allow-suspicious-udfs             FALSE
auto-increment-increment          1
auto-increment-offset             1
automatic-sp-privileges           TRUE
back_log                          50
basedir                           /home/jon/bin/mysql-5.7/
bind-address                      (No default value)
binlog-row-event-max-size         1024
binlog_cache_size                 32768
binlog_format                     (No default value)
bulk_insert_buffer_size           8388608
character-set-client-handshake    TRUE
character-set-filesystem          binary
character-set-server              latin1
character-sets-dir                /home/jon/bin/mysql-5.7/share/mysql/charsets/
chroot                            (No default value)
collation-server                  latin1_swedish_ci
completion-type                   0
concurrent-insert                 1
connect_timeout                   10
console                           FALSE
datadir                           .
datetime_format                   %Y-%m-%d %H:%i:%s
date_format                       %Y-%m-%d
default-storage-engine            MyISAM
default-time-zone                 (No default value)
default_week_format               0
delayed_insert_limit              100
delayed_insert_timeout            300
delayed_queue_size                1000
disconnect-slave-event-count      0
div_precision_increment           4
engine-condition-pushdown         TRUE
expire_logs_days                  0
external-locking                  FALSE
flush_time                        0
ft_max_word_len                   84
ft_min_word_len                   4
ft_query_expansion_limit          20
ft_stopword_file                  (No default value)
gdb                               FALSE
general_log                       FALSE
general_log_file                  (No default value)
group_concat_max_len              1024
help                              TRUE
init-connect                      (No default value)
init-file                         (No default value)
init-slave                        (No default value)
innodb                            TRUE
innodb-adaptive-hash-index        TRUE
innodb-additional-mem-pool-size   1048576
innodb-autoextend-increment       8
innodb-autoinc-lock-mode          1
innodb-buffer-pool-size           8388608
innodb-checksums                  TRUE
innodb-commit-concurrency         0
innodb-concurrency-tickets        500
innodb-data-file-path             (No default value)
innodb-data-home-dir              (No default value)
innodb-doublewrite                TRUE
innodb-fast-shutdown              1
innodb-file-io-threads            4
```

```
innodb-file-per-table             FALSE
innodb-flush-log-at-trx-commit    1
innodb-flush-method               (No default value)
innodb-force-recovery             0
innodb-lock-wait-timeout          50
innodb-locks-unsafe-for-binlog    FALSE
innodb-log-buffer-size            1048576
innodb-log-file-size              5242880
innodb-log-files-in-group         2
innodb-log-group-home-dir         (No default value)
innodb-max-dirty-pages-pct        90
innodb-max-purge-lag              0
innodb-mirrored-log-groups        1
innodb-open-files                 300
innodb-rollback-on-timeout        FALSE
innodb-stats-on-metadata          TRUE
innodb-status-file                FALSE
innodb-support-xa                 TRUE
innodb-sync-spin-loops            20
innodb-table-locks                TRUE
innodb-thread-concurrency         8
innodb-thread-sleep-delay         10000
interactive_timeout               28800
join_buffer_size                  131072
keep_files_on_create              FALSE
key_buffer_size                   8384512
key_cache_age_threshold           300
key_cache_block_size              1024
key_cache_division_limit          100
language                          /home/jon/bin/mysql-5.7/share/mysql/english/
large-pages                       FALSE
lc-time-names                     en_US
local-infile                      TRUE
log                               (No default value)
log-bin                           (No default value)
log-bin-index                     (No default value)
log-bin-trust-function-creators   FALSE
log-error
log-error-verbosity               1
log-isam                          myisam.log
log-output                        FILE
log-queries-not-using-indexes     FALSE
log-short-format                  FALSE
log-slave-updates                 FALSE
log-slow-admin-statements         FALSE
log-slow-slave-statements         FALSE
log-tc                            tc.log
log-tc-size                       24576
log-warnings                      1
log_slow_queries                  (No default value)
long_query_time                   10
low-priority-updates              FALSE
lower_case_table_names            0
master-retry-count                86400
max-binlog-dump-events            0
max_allowed_packet                1048576
max_binlog_cache_size             18446744073709547520
max_binlog_size                   1073741824
max_connections                   151
max_connect_errors                10
max_delayed_threads               20
max_error_count                   64
max_heap_table_size               16777216
max_join_size                     18446744073709551615
max_length_for_sort_data          1024
max_prepared_stmt_count           16382
max_relay_log_size                0
```

```
max_seeks_for_key                 18446744073709551615
max_sort_length                   1024
max_sp_recursion_depth            0
max_tmp_tables                    32
max_user_connections              0
max_write_lock_count              18446744073709551615
memlock                           FALSE
min_examined_row_limit            0
multi_range_count                 256
myisam-recover-options            OFF
myisam_block_size                 1024
myisam_data_pointer_size          6
myisam_max_sort_file_size         9223372036853727232
myisam_repair_threads             1
myisam_sort_buffer_size           8388608
myisam_stats_method               nulls_unequal
myisam_use_mmap                   FALSE
ndb-autoincrement-prefetch-sz     1
ndb-cache-check-time              0
ndb-connectstring                 (No default value)
ndb-extra-logging                 0
ndb-force-send                    TRUE
ndb-index-stat-enable             FALSE
ndb-mgmd-host                     (No default value)
ndb-nodeid                        0
ndb-optimized-node-selection      TRUE
ndb-report-thresh-binlog-epoch-slip 3
ndb-report-thresh-binlog-mem-usage 10
ndb-shm                           FALSE
ndb-use-copying-alter-table       FALSE
ndb-use-exact-count               TRUE
ndb-use-transactions              TRUE
ndb_force_send                    TRUE
ndb_use_exact_count               TRUE
ndb_use_transactions              TRUE
net_buffer_length                 16384
net_read_timeout                  30
net_retry_count                   10
net_write_timeout                 60
new                               FALSE
old                               FALSE
old-alter-table                   FALSE
old-passwords                     FALSE
old-style-user-limits             FALSE
open_files_limit                  1024
optimizer_prune_level             1
optimizer_search_depth            62
pid-file                          /home/jon/bin/mysql-5.7/var/tonfisk.pid
plugin_dir                        /home/jon/bin/mysql-5.7/lib/mysql/plugin
port                              3306
port-open-timeout                 0
preload_buffer_size               32768
profiling_history_size            15
query_alloc_block_size            8192
query_cache_limit                 1048576
query_cache_min_res_unit          4096
query_cache_size                  0
query_cache_type                  1
query_cache_wlock_invalidate      FALSE
query_prealloc_size               8192
range_alloc_block_size            4096
read_buffer_size                  131072
read_only                         FALSE
read_rnd_buffer_size              262144
relay-log                         (No default value)
relay-log-index                   (No default value)
relay-log-info-file               relay-log.info
```

```
relay_log_purge                   TRUE
relay_log_space_limit             0
replicate-same-server-id          FALSE
report-host                       (No default value)
report-password                   (No default value)
report-port                       3306
report-user                       (No default value)
safe-user-create                  FALSE
secure-auth                       TRUE
secure-file-priv                  (No default value)
server-id                         0
show-slave-auth-info              FALSE
skip-grant-tables                 FALSE
skip-slave-start                  FALSE
slave-exec-mode                   STRICT
slave-load-tmpdir                 /tmp
slave_compressed_protocol         FALSE
slave_net_timeout                 3600
slave_transaction_retries         10
slow-query-log                    FALSE
slow_launch_time                  2
slow_query_log_file               (No default value)
socket                            /tmp/mysql.sock
sort_buffer_size                  2097144
sporadic-binlog-dump-fail         FALSE
sql-mode                          OFF
symbolic-links                    TRUE
sync-binlog                       0
sync-frm                          TRUE
sysdate-is-now                    FALSE
table_definition_cache            256
table_open_cache                  400
tc-heuristic-recover              (No default value)
temp-pool                         TRUE
thread_cache_size                 0
thread_concurrency                10
thread_stack                      262144
timed_mutexes                     FALSE
time_format                       %H:%i:%s
tmpdir                            (No default value)
tmp_table_size                    16777216
transaction_alloc_block_size      8192
transaction_prealloc_size         4096
updatable_views_with_limit        1
verbose                           TRUE
wait_timeout                      28800
```

For a `mysqld` server that is currently running, you can see the current values of its system variables by connecting to it and issuing this statement:

```
mysql> SHOW VARIABLES;
```

You can also see some statistical and status indicators for a running server by issuing this statement:

```
mysql> SHOW STATUS;
```

System variable and status information also can be obtained using `mysqladmin`:

```
shell> mysqladmin variables
shell> mysqladmin extended-status
```

For a full description of all system and status variables, see Section 5.1.4, “Server System Variables”, and Section 5.1.6, “Server Status Variables”.

MySQL uses algorithms that are very scalable, so you can usually run with very little memory. However, normally you get better performance by giving MySQL more memory.

When tuning a MySQL server, the two most important variables to configure are `key_buffer_size` and `table_open_cache`. You should first feel confident that you have these set appropriately before trying to change any other variables.

The following examples indicate some typical variable values for different runtime configurations.

- If you have at least 256MB of memory and many tables and want maximum performance with a moderate number of clients, use something like this:

  ```
  shell> mysqld_safe --key_buffer_size=64M --table_open_cache=256 \
             --sort_buffer_size=4M --read_buffer_size=1M &
  ```

- If you have only 128MB of memory and only a few tables, but you still do a lot of sorting, you can use something like this:

  ```
  shell> mysqld_safe --key_buffer_size=16M --sort_buffer_size=1M
  ```

  If there are very many simultaneous connections, swapping problems may occur unless `mysqld` has been configured to use very little memory for each connection. `mysqld` performs better if you have enough memory for all connections.

- With little memory and lots of connections, use something like this:

  ```
  shell> mysqld_safe --key_buffer_size=512K --sort_buffer_size=100K \
             --read_buffer_size=100K &
  ```

  Or even this:

  ```
  shell> mysqld_safe --key_buffer_size=512K --sort_buffer_size=16K \
             --table_open_cache=32 --read_buffer_size=8K \
             --net_buffer_length=1K &
  ```

If you are performing `GROUP BY` or `ORDER BY` operations on tables that are much larger than your available memory, increase the value of `read_rnd_buffer_size` to speed up the reading of rows following sorting operations.

You can make use of the example option files included with your MySQL distribution; see Section 5.1.2, “Server Configuration Defaults”.

If you specify an option on the command line for `mysqld` or `mysqld_safe`, it remains in effect only for that invocation of the server. To use the option every time the server runs, put it in an option file.

To see the effects of a parameter change, do something like this:

```
shell> mysqld --key_buffer_size=32M --verbose --help
```

The variable values are listed near the end of the output. Make sure that the `--verbose` and `--help` options are last. Otherwise, the effect of any options listed after them on the command line are not reflected in the output.

For information on tuning the `InnoDB` storage engine, see Section 14.2.12.1, “`InnoDB` Performance Tuning Tips”.

8.11.3 Optimizing Disk I/O

- Disk seeks are a huge performance bottleneck. This problem becomes more apparent when the amount of data starts to grow so large that effective caching becomes impossible. For large databases where you access data more or less randomly, you can be sure that you need at least one disk seek to read and a couple of disk seeks to write things. To minimize this problem, use disks with low seek times.

- Increase the number of available disk spindles (and thereby reduce the seek overhead) by either symlinking files to different disks or striping the disks:

  - Using symbolic links

    This means that, for `MyISAM` tables, you symlink the index file and data files from their usual location in the data directory to another disk (that may also be striped). This makes both the seek and read times better, assuming that the disk is not used for other purposes as well. See Section 8.11.3.1, “Using Symbolic Links”.

  - Striping

    Striping means that you have many disks and put the first block on the first disk, the second block on the second disk, and the *`N`*-th block on the (*`N`* `MOD` *`number_of_disks`*) disk, and so on. This means if your normal data size is less than the stripe size (or perfectly aligned), you get much better performance. Striping is very dependent on the operating system and the stripe size, so benchmark your application with different stripe sizes. See Section 8.12.3, “Using Your Own Benchmarks”.

    The speed difference for striping is *very* dependent on the parameters. Depending on how you set the striping parameters and number of disks, you may get differences measured in orders of magnitude. You have to choose to optimize for random or sequential access.

- For reliability, you may want to use RAID 0+1 (striping plus mirroring), but in this case, you need 2 × *`N`* drives to hold *`N`* drives of data. This is probably the best option if you have the money for it. However, you may also have to invest in some volume-management software to handle it efficiently.

- A good option is to vary the RAID level according to how critical a type of data is. For example, store semi-important data that can be regenerated on a RAID 0 disk, but store really important data such as host information and logs on a RAID 0+1 or RAID *`N`* disk. RAID *`N`* can be a problem if you have many writes, due to the time required to update the parity bits.

- On Linux, you can get much better performance by using `hdparm` to configure your disk's interface. (Up to 100% under load is not uncommon.) The following `hdparm` options should be quite good for MySQL, and probably for many other applications:

  ```
  hdparm -m 16 -d 1
  ```

  Note that performance and reliability when using this command depend on your hardware, so we strongly suggest that you test your system thoroughly after using `hdparm`. Please consult the `hdparm` manual page for more information. If `hdparm` is not used wisely, file system corruption may result, so back up everything before experimenting!

- You can also set the parameters for the file system that the database uses:

  If you do not need to know when files were last accessed (which is not really useful on a database server), you can mount your file systems with the `-o noatime` option. That skips updates to the last access time in inodes on the file system, which avoids some disk seeks.

On many operating systems, you can set a file system to be updated asynchronously by mounting it with the `-o async` option. If your computer is reasonably stable, this should give you better performance without sacrificing too much reliability. (This flag is on by default on Linux.)

8.11.3.1 Using Symbolic Links

You can move databases or tables from the database directory to other locations and replace them with symbolic links to the new locations. You might want to do this, for example, to move a database to a file system with more free space or increase the speed of your system by spreading your tables to different disks.

For `InnoDB` tables, use the `DATA DIRECTORY` clause on the `CREATE TABLE` statement instead of symbolic links, as explained in Section 14.2.5.4, "Specifying the Location of a Tablespace". This new feature is a supported, cross-platform technique.

The recommended way to do this is to symlink entire database directories to a different disk. Symlink `MyISAM` tables only as a last resort.

To determine the location of your data directory, use this statement:

```
SHOW VARIABLES LIKE 'datadir';
```

Using Symbolic Links for Databases on Unix

On Unix, the way to symlink a database is first to create a directory on some disk where you have free space and then to create a soft link to it from the MySQL data directory.

```
shell> mkdir /dr1/databases/test
shell> ln -s /dr1/databases/test /path/to/datadir
```

MySQL does not support linking one directory to multiple databases. Replacing a database directory with a symbolic link works as long as you do not make a symbolic link between databases. Suppose that you have a database `db1` under the MySQL data directory, and then make a symlink `db2` that points to `db1`:

```
shell> cd /path/to/datadir
shell> ln -s db1 db2
```

The result is that, or any table `tbl_a` in `db1`, there also appears to be a table `tbl_a` in `db2`. If one client updates `db1.tbl_a` and another client updates `db2.tbl_a`, problems are likely to occur.

Using Symbolic Links for `MyISAM` Tables on Unix

Symlinks are fully supported only for `MyISAM` tables. For files used by tables for other storage engines, you may get strange problems if you try to use symbolic links. For `InnoDB` tables, use the alternative technique explained in Section 14.2.5.4, "Specifying the Location of a Tablespace" instead.

Do not symlink tables on systems that do not have a fully operational `realpath()` call. (Linux and Solaris support `realpath()`). To determine whether your system supports symbolic links, check the value of the `have_symlink` system variable using this statement:

```
SHOW VARIABLES LIKE 'have_symlink';
```

The handling of symbolic links for `MyISAM` tables works as follows:

- In the data directory, you always have the table format (.frm) file, the data (.MYD) file, and the index (.MYI) file. The data file and index file can be moved elsewhere and replaced in the data directory by symlinks. The format file cannot.

- You can symlink the data file and the index file independently to different directories.

- To instruct a running MySQL server to perform the symlinking, use the DATA DIRECTORY and INDEX DIRECTORY options to CREATE TABLE. See Section 13.1.14, "CREATE TABLE Syntax". Alternatively, if mysqld is not running, symlinking can be accomplished manually using ln -s from the command line.

  > **Note**
  >
  > The path used with either or both of the DATA DIRECTORY and INDEX DIRECTORY options may not include the MySQL data directory. (Bug #32167)

- myisamchk does not replace a symlink with the data file or index file. It works directly on the file to which the symlink points. Any temporary files are created in the directory where the data file or index file is located. The same is true for the ALTER TABLE, OPTIMIZE TABLE, and REPAIR TABLE statements.

- > **Note**
  >
  > When you drop a table that is using symlinks, *both the symlink and the file to which the symlink points are dropped*. This is an extremely good reason *not* to run mysqld as the system root or permit system users to have write access to MySQL database directories.

- If you rename a table with ALTER TABLE ... RENAME or RENAME TABLE and you do not move the table to another database, the symlinks in the database directory are renamed to the new names and the data file and index file are renamed accordingly.

- If you use ALTER TABLE ... RENAME or RENAME TABLE to move a table to another database, the table is moved to the other database directory. If the table name changed, the symlinks in the new database directory are renamed to the new names and the data file and index file are renamed accordingly.

- If you are not using symlinks, start mysqld with the --skip-symbolic-links option to ensure that no one can use mysqld to drop or rename a file outside of the data directory.

These table symlink operations are not supported:

- ALTER TABLE ignores the DATA DIRECTORY and INDEX DIRECTORY table options.

- As indicated previously, only the data and index files can be symbolic links. The .frm file must *never* be a symbolic link. Attempting to do this (for example, to make one table name a synonym for another) produces incorrect results. Suppose that you have a database db1 under the MySQL data directory, a table tbl1 in this database, and in the db1 directory you make a symlink tbl2 that points to tbl1:

  ```
  shell> cd /path/to/datadir/db1
  shell> ln -s tbl1.frm tbl2.frm
  shell> ln -s tbl1.MYD tbl2.MYD
  shell> ln -s tbl1.MYI tbl2.MYI
  ```

  Problems result if one thread reads db1.tbl1 and another thread updates db1.tbl2:

  - The query cache is "fooled" (it has no way of knowing that tbl1 has not been updated, so it returns outdated results).

  - ALTER statements on tbl2 fail.

Using Symbolic Links for Databases on Windows

On Windows, symbolic links can be used for database directories. This enables you to put a database directory at a different location (for example, on a different disk) by setting up a symbolic link to it. Use of database symlinks on Windows is similar to their use on Unix, although the procedure for setting up the link differs.

Suppose that you want to place the database directory for a database named mydb at D:\data\mydb. To do this, create a symbolic link in the MySQL data directory that points to D:\data\mydb. However, before creating the symbolic link, make sure that the D:\data\mydb directory exists by creating it if necessary. If you already have a database directory named mydb in the data directory, move it to D:\data. Otherwise, the symbolic link will be ineffective. To avoid problems, make sure that the server is not running when you move the database directory.

Windows Vista, Windows Server 2008, or newer have native symbolic link support, so you can create a symlink using the mklink command. This command requires administrative privileges.

1. Change location into the data directory:

```
C:\> cd \path\to\datadir
```

2. In the data directory, create a symlink named mydb that points to the location of the database directory:

```
C:\> mklink /d mydb D:\data\mydb
```

After this, all tables created in the database mydb are created in D:\data\mydb.

8.11.4 Optimizing Memory Use

8.11.4.1 How MySQL Uses Memory

The following list indicates some of the ways that the mysqld server uses memory. Where applicable, the name of the system variable relevant to the memory use is given:

- All threads share the MyISAM key buffer; its size is determined by the key_buffer_size variable. Other buffers used by the server are allocated as needed. See Section 8.11.2, “Tuning Server Parameters”.

- Each thread that is used to manage client connections uses some thread-specific space. The following list indicates these and which variables control their size:

  - A stack (variable thread_stack)

  - A connection buffer (variable net_buffer_length)

  - A result buffer (variable net_buffer_length)

  The connection buffer and result buffer each begin with a size equal to net_buffer_length bytes, but are dynamically enlarged up to max_allowed_packet bytes as needed. The result buffer shrinks to net_buffer_length bytes after each SQL statement. While a statement is running, a copy of the current statement string is also allocated.

- All threads share the same base memory.

- When a thread is no longer needed, the memory allocated to it is released and returned to the system unless the thread goes back into the thread cache. In that case, the memory remains allocated.

- The myisam_use_mmap system variable can be set to 1 to enable memory-mapping for all MyISAM tables.
- Each request that performs a sequential scan of a table allocates a *read buffer* (variable read_buffer_size).
- When reading rows in an arbitrary sequence (for example, following a sort), a *random-read buffer* (variable read_rnd_buffer_size) may be allocated to avoid disk seeks.
- All joins are executed in a single pass, and most joins can be done without even using a temporary table. Most temporary tables are memory-based hash tables. Temporary tables with a large row length (calculated as the sum of all column lengths) or that contain BLOB columns are stored on disk.

  If an internal in-memory temporary table becomes too large, MySQL handles this automatically by changing the table from in-memory to on-disk format, to be handled by the MyISAM storage engine. You can increase the permissible temporary table size as described in Section 8.4.4, "How MySQL Uses Internal Temporary Tables".
- Most requests that perform a sort allocate a sort buffer and zero to two temporary files depending on the result set size. See Section C.5.4.4, "Where MySQL Stores Temporary Files".
- Almost all parsing and calculating is done in thread-local and reusable memory pools. No memory overhead is needed for small items, so the normal slow memory allocation and freeing is avoided. Memory is allocated only for unexpectedly large strings.
- For each MyISAM table that is opened, the index file is opened once; the data file is opened once for each concurrently running thread. For each concurrent thread, a table structure, column structures for each column, and a buffer of size 3 * *N* are allocated (where *N* is the maximum row length, not counting BLOB columns). A BLOB column requires five to eight bytes plus the length of the BLOB data. The MyISAM storage engine maintains one extra row buffer for internal use.
- For each table having BLOB columns, a buffer is enlarged dynamically to read in larger BLOB values. If you scan a table, a buffer as large as the largest BLOB value is allocated.
- Handler structures for all in-use tables are saved in a cache and managed as a FIFO. The initial cache size is taken from the value of the table_open_cache system variable. If a table has been used by two running threads at the same time, the cache contains two entries for the table. See Section 8.4.3.1, "How MySQL Opens and Closes Tables".
- A FLUSH TABLES statement or mysqladmin flush-tables command closes all tables that are not in use at once and marks all in-use tables to be closed when the currently executing thread finishes. This effectively frees most in-use memory. FLUSH TABLES does not return until all tables have been closed.
- The server caches information in memory as a result of GRANT, CREATE USER, CREATE SERVER, and INSTALL PLUGIN statements. This memory is not released by the corresponding REVOKE, DROP USER, DROP SERVER, and UNINSTALL PLUGIN statements, so for a server that executes many instances of the statements that cause caching, there will be an increase in memory use. This cached memory can be freed with FLUSH PRIVILEGES.

ps and other system status programs may report that mysqld uses a lot of memory. This may be caused by thread stacks on different memory addresses. For example, the Solaris version of ps counts the unused memory between stacks as used memory. To verify this, check available swap with swap -s. We test mysqld with several memory-leakage detectors (both commercial and Open Source), so there should be no memory leaks.

8.11.4.2 Enabling Large Page Support

Some hardware/operating system architectures support memory pages greater than the default (usually 4KB). The actual implementation of this support depends on the underlying hardware and operating system. Applications that perform a lot of memory accesses may obtain performance improvements by using large pages due to reduced Translation Lookaside Buffer (TLB) misses.

In MySQL, large pages can be used by InnoDB, to allocate memory for its buffer pool and additional memory pool.

Standard use of large pages in MySQL attempts to use the largest size supported, up to 4MB. Under Solaris, a “super large pages” feature enables uses of pages up to 256MB. This feature is available for recent SPARC platforms. It can be enabled or disabled by using the `--super-large-pages` or `--skip-super-large-pages` option.

MySQL also supports the Linux implementation of large page support (which is called HugeTLB in Linux).

Before large pages can be used on Linux, the kernel must be enabled to support them and it is necessary to configure the HugeTLB memory pool. For reference, the HugeTBL API is documented in the `Documentation/vm/hugetlbpage.txt` file of your Linux sources.

The kernel for some recent systems such as Red Hat Enterprise Linux appear to have the large pages feature enabled by default. To check whether this is true for your kernel, use the following command and look for output lines containing “huge”:

```
shell> cat /proc/meminfo | grep -i huge
HugePages_Total:       0
HugePages_Free:        0
HugePages_Rsvd:        0
HugePages_Surp:        0
Hugepagesize:       4096 kB
```

The nonempty command output indicates that large page support is present, but the zero values indicate that no pages are configured for use.

If your kernel needs to be reconfigured to support large pages, consult the `hugetlbpage.txt` file for instructions.

Assuming that your Linux kernel has large page support enabled, configure it for use by MySQL using the following commands. Normally, you put these in an `rc` file or equivalent startup file that is executed during the system boot sequence, so that the commands execute each time the system starts. The commands should execute early in the boot sequence, before the MySQL server starts. Be sure to change the allocation numbers and the group number as appropriate for your system.

```
# Set the number of pages to be used.
# Each page is normally 2MB, so a value of 20 = 40MB.
# This command actually allocates memory, so this much
# memory must be available.
echo 20 > /proc/sys/vm/nr_hugepages

# Set the group number that is permitted to access this
# memory (102 in this case). The mysql user must be a
# member of this group.
echo 102 > /proc/sys/vm/hugetlb_shm_group

# Increase the amount of shmem permitted per segment
# (12G in this case).
echo 1560281088 > /proc/sys/kernel/shmmax

# Increase total amount of shared memory.  The value
# is the number of pages. At 4KB/page, 4194304 = 16GB.
echo 4194304 > /proc/sys/kernel/shmall
```

For MySQL usage, you normally want the value of shmmax to be close to the value of shmall.

To verify the large page configuration, check /proc/meminfo again as described previously. Now you should see some nonzero values:

```
shell> cat /proc/meminfo | grep -i huge
HugePages_Total:      20
HugePages_Free:       20
HugePages_Rsvd:        0
HugePages_Surp:        0
Hugepagesize:       4096 kB
```

The final step to make use of the hugetlb_shm_group is to give the mysql user an “unlimited” value for the memlock limit. This can by done either by editing /etc/security/limits.conf or by adding the following command to your mysqld_safe script:

```
ulimit -l unlimited
```

Adding the ulimit command to mysqld_safe causes the root user to set the memlock limit to unlimited before switching to the mysql user. (This assumes that mysqld_safe is started by root.)

Large page support in MySQL is disabled by default. To enable it, start the server with the --large-pages option. For example, you can use the following lines in your server's my.cnf file:

```
[mysqld]
large-pages
```

With this option, InnoDB uses large pages automatically for its buffer pool and additional memory pool. If InnoDB cannot do this, it falls back to use of traditional memory and writes a warning to the error log: Warning: Using conventional memory pool

To verify that large pages are being used, check /proc/meminfo again:

```
shell> cat /proc/meminfo | grep -i huge
HugePages_Total:      20
HugePages_Free:       20
HugePages_Rsvd:        2
HugePages_Surp:        0
Hugepagesize:       4096 kB
```

8.11.5 Optimizing Network Use

8.11.5.1 How MySQL Uses Threads for Client Connections

Connection manager threads handle client connection requests on the network interfaces that the server listens to. On all platforms, one manager thread handles TCP/IP connection requests. On Unix, this manager thread also handles Unix socket file connection requests. On Windows, a manager thread handles shared-memory connection requests, and another handles named-pipe connection requests. The server does not create threads to handle interfaces that it does not listen to. For example, a Windows server that does not have support for named-pipe connections enabled does not create a thread to handle them.

Connection manager threads associate each client connection with a thread dedicated to it that handles authentication and request processing for that connection. Manager threads create a new thread when necessary but try to avoid doing so by consulting the thread cache first to see whether it contains a thread that can be used for the connection. When a connection ends, its thread is returned to the thread cache if the cache is not full.

In this connection thread model, there are as many threads as there are clients currently connected, which has some disadvantages when server workload must scale to handle large numbers of connections. For example, thread creation and disposal becomes expensive. Also, each thread requires server and kernel resources, such as stack space. To accommodate a large number of simultaneous connections, the stack size per thread must be kept small, leading to a situation where it is either too small or the server consumes large amounts of memory. Exhaustion of other resources can occur as well, and scheduling overhead can become significant.

To control and monitor how the server manages threads that handle client connections, several system and status variables are relevant. (See Section 5.1.4, “Server System Variables”, and Section 5.1.6, “Server Status Variables”.)

The thread cache has a size determined by the `thread_cache_size` system variable. The default value is 0 (no caching), which causes a thread to be set up for each new connection and disposed of when the connection terminates. Set `thread_cache_size` to *`N`* to enable *`N`* inactive connection threads to be cached. `thread_cache_size` can be set at server startup or changed while the server runs. A connection thread becomes inactive when the client connection with which it was associated terminates.

To monitor the number of threads in the cache and how many threads have been created because a thread could not be taken from the cache, monitor the `Threads_cached` and `Threads_created` status variables.

You can set `max_connections` at server startup or at runtime to control the maximum number of clients that can connect simultaneously.

When the thread stack is too small, this limits the complexity of the SQL statements which the server can handle, the recursion depth of stored procedures, and other memory-consuming actions. To set a stack size of *`N`* bytes for each thread, start the server with `--thread_stack=N`.

8.11.5.2 DNS Lookup Optimization and the Host Cache

The MySQL server maintains a host cache in memory that contains information about clients: IP address, host name, and error information. The server uses this cache for nonlocal TCP connections. It does not use the cache for TCP connections established using a loopback interface address (`127.0.0.1` or `::1`), or for connections established using a Unix socket file, named pipe, or shared memory.

For each new client connection, the server uses the client IP address to check whether the client host name is in the host cache. If not, the server attempts to resolve the host name. First, it resolves the IP address to a host name and resolves that host name back to an IP address. Then it compares the result to the original IP address to ensure that they are the same. The server stores information about the result of this operation in the host cache. If the cache is full, the least recently used entry is discarded.

The `host_cache` Performance Schema table exposes the contents of the host cache so that it can be examined using `SELECT` statements. This may help you diagnose the causes of connection problems. See Section 20.9.13.1, “The `host_cache` Table”.

The server handles entries in the host cache like this:

1. When the first TCP client connection reaches the server from a given IP address, a new entry is created to record the client IP, host name, and client lookup validation flag. Initially, the host name is

set to NULL and the flag is false. This entry is also used for subsequent client connections from the same originating IP.

2. If the validation flag for the client IP entry is false, the server attempts an IP-to-host name DNS resolution. If that is successful, the host name is updated with the resolved host name and the validation flag is set to true. If resolution is unsuccessful, the action taken depends on whether the error is permanent or transient. For permanent failures, the host name remains NULL and the validation flag is set to true. For transient failures, the host name and validation flag remain unchanged. (Another DNS resolution attempt occurs the next time a client connects from this IP.)

3. If an error occurs while processing an incoming client connection from a given IP address, the server updates the corresponding error counters in the entry for that IP. For a description of the errors recorded, see Section 20.9.13.1, “The host_cache Table”.

The server performs host name resolution using the thread-safe gethostbyaddr_r() and gethostbyname_r() calls if the operating system supports them. Otherwise, the thread performing the lookup locks a mutex and calls gethostbyaddr() and gethostbyname() instead. In this case, no other thread can resolve host names that are not in the host cache until the thread holding the mutex lock releases it.

The server uses the host cache for several purposes:

- By caching the results of IP-to-host name lookups, the server avoids doing a DNS lookup for each client connection. Instead, for a given host, it needs to perform a lookup only for the first connection from that host.

- The cache contains information about errors that occur during the connection process. Some errors are considered “blocking.” If too many of these occur successively from a given host without a successful connection, the server blocks further connections from that host. The max_connect_errors system variable determines the number of permitted errors before blocking occurs. See Section C.5.2.6, “Host '*host_name*' is blocked”.

To unblock blocked hosts, flush the host cache by issuing a FLUSH HOSTS statement or executing a mysqladmin flush-hosts command.

It is possible for a blocked host to become unblocked even without FLUSH HOSTS if activity from other hosts has occurred since the last connection attempt from the blocked host. This can occur because the server discards the least recently used cache entry to make room for a new entry if the cache is full when a connection arrives from a client IP not in the cache. If the discarded entry is for a blocked host, that host becomes unblocked.

The host cache is enabled by default. To disable it, set the host_cache_size system variable to 0, either at server startup or at runtime.

To disable DNS host name lookups, start the server with the --skip-name-resolve option. In this case, the server uses only IP addresses and not host names to match connecting hosts to rows in the MySQL grant tables. Only accounts specified in those tables using IP addresses can be used.

If you have a very slow DNS and many hosts, you might be able to improve performance either by disabling DNS lookups with --skip-name-resolve or by increasing the value of host_cache_size to make the host cache larger.

To disallow TCP/IP connections entirely, start the server with the --skip-networking option.

Some connection errors are not associated with TCP connections, occur very early in the connection process (even before an IP address is known), or are not specific to any particular IP address (such as out-of-memory conditions). For information about these errors, check the Connection_errors_*xxx* status variables (see Section 5.1.6, “Server Status Variables”).

8.12 Measuring Performance (Benchmarking)

To measure performance, consider the following factors:

- Whether you are measuring the speed of a single operation on a quiet system, or how a set of operations (a “workload”) works over a period of time. With simple tests, you usually test how changing one aspect (a configuration setting, the set of indexes on a table, the SQL clauses in a query) affects performance. Benchmarks are typically long-running and elaborate performance tests, where the results could dictate high-level choices such as hardware and storage configuration, or how soon to upgrade to a new MySQL version.

- For benchmarking, sometimes you must simulate a heavy database workload to get an accurate picture.

- Performance can vary depending on so many different factors that a difference of a few percentage points might not be a decisive victory. The results might shift the opposite way when you test in a different environment.

- Certain MySQL features help or do not help performance depending on the workload. For completeness, always test performance with those features turned on and turned off. The two most important features to try with each workload are the MySQL query cache, and the adaptive hash index for InnoDB tables.

This section progresses from simple and direct measurement techniques that a single developer can do, to more complicated ones that require additional expertise to perform and interpret the results.

8.12.1 Measuring the Speed of Expressions and Functions

To measure the speed of a specific MySQL expression or function, invoke the BENCHMARK() function using the mysql client program. Its syntax is BENCHMARK(*loop_count*,*expression*). The return value is always zero, but mysql prints a line displaying approximately how long the statement took to execute. For example:

```
mysql> SELECT BENCHMARK(1000000,1+1);
+------------------------+
| BENCHMARK(1000000,1+1) |
+------------------------+
|                      0 |
+------------------------+
1 row in set (0.32 sec)
```

This result was obtained on a Pentium II 400MHz system. It shows that MySQL can execute 1,000,000 simple addition expressions in 0.32 seconds on that system.

The built-in MySQL functions are typically highly optimized, but there may be some exceptions. BENCHMARK() is an excellent tool for finding out if some function is a problem for your queries.

8.12.2 The MySQL Benchmark Suite

This benchmark suite is meant to tell any user what operations a given SQL implementation performs well or poorly. You can get a good idea for how the benchmarks work by looking at the code and results in the sql-bench directory in any MySQL source distribution.

Note that this benchmark is single-threaded, so it measures the minimum time for the operations performed. We plan to add multi-threaded tests to the benchmark suite in the future.

To use the benchmark suite, the following requirements must be satisfied:

- The benchmark suite is provided with MySQL source distributions. You can either download a released distribution from http://dev.mysql.com/downloads/, or use the current development source tree. (See Section 2.8.3, “Installing MySQL Using a Development Source Tree”.)

- The benchmark scripts are written in Perl and use the Perl DBI module to access database servers, so DBI must be installed. You also need the server-specific DBD drivers for each of the servers you want to test. For example, to test MySQL, PostgreSQL, and DB2, you must have the `DBD::mysql`, `DBD::Pg`, and `DBD::DB2` modules installed. See Section 2.12, “Perl Installation Notes”.

After you obtain a MySQL source distribution, you can find the benchmark suite located in its `sql-bench` directory. To run the benchmark tests, build MySQL, and then change location into the `sql-bench` directory and execute the `run-all-tests` script:

```
shell> cd sql-bench
shell> perl run-all-tests --server=server_name
```

`server_name` should be the name of one of the supported servers. To get a list of all options and supported servers, invoke this command:

```
shell> perl run-all-tests --help
```

The `crash-me` script also is located in the `sql-bench` directory. `crash-me` tries to determine what features a database system supports and what its capabilities and limitations are by actually running queries. For example, it determines:

- What data types are supported

- How many indexes are supported

- What functions are supported

- How big a query can be

- How big a `VARCHAR` column can be

For more information about benchmark results, visit http://www.mysql.com/why-mysql/benchmarks/.

8.12.3 Using Your Own Benchmarks

Benchmark your application and database to find out where the bottlenecks are. After fixing one bottleneck (or by replacing it with a “dummy” module), you can proceed to identify the next bottleneck. Even if the overall performance for your application currently is acceptable, you should at least make a plan for each bottleneck and decide how to solve it if someday you really need the extra performance.

For examples of portable benchmark programs, look at those in the MySQL benchmark suite. See Section 8.12.2, “The MySQL Benchmark Suite”. You can take any program from this suite and modify it for your own needs. By doing this, you can try different solutions to your problem and test which really is fastest for you.

Another free benchmark suite is the Open Source Database Benchmark, available at http://osdb.sourceforge.net/.

It is very common for a problem to occur only when the system is very heavily loaded. We have had many customers who contact us when they have a (tested) system in production and have encountered load problems. In most cases, performance problems turn out to be due to issues of basic database design (for example, table scans are not good under high load) or problems with the operating system or libraries. Most of the time, these problems would be much easier to fix if the systems were not already in production.

To avoid problems like this, benchmark your whole application under the worst possible load:

- The mysqlslap program can be helpful for simulating a high load produced by multiple clients issuing queries simultaneously. See Section 4.5.7, "mysqlslap — Load Emulation Client".
- You can also try benchmarking packages such as SysBench and DBT2, available at http://sourceforge.net/projects/sysbench/, and http://osdldbt.sourceforge.net/#dbt2.

These programs or packages can bring a system to its knees, so be sure to use them only on your development systems.

8.12.4 Measuring Performance with performance_schema

You can query the tables in the performance_schema database to see real-time information about the performance characteristics of your server and the applications it is running. See Chapter 20, *MySQL Performance Schema* for details.

8.12.5 Examining Thread Information

When you are attempting to ascertain what your MySQL server is doing, it can be helpful to examine the process list, which is the set of threads currently executing within the server. Process list information is available from these sources:

- The SHOW [FULL] PROCESSLIST statement: Section 13.7.5.28, "SHOW PROCESSLIST Syntax"
- The SHOW PROFILE statement: Section 13.7.5.30, "SHOW PROFILES Syntax"
- The INFORMATION_SCHEMA PROCESSLIST table: Section 19.16, "The INFORMATION_SCHEMA PROCESSLIST Table"
- The mysqladmin processlist command: Section 4.5.2, "mysqladmin — Client for Administering a MySQL Server"
- The performance_schema.threads table: Section 20.9.13, "Performance Schema Miscellaneous Tables"

Access to threads does not require a mutex and has minimal impact on server performance. INFORMATION_SCHEMA.PROCESSLIST and SHOW PROCESSLIST have negative performance consequences because they require a mutex. threads also shows information about background threads, which INFORMATION_SCHEMA.PROCESSLIST and SHOW PROCESSLIST do not. This means that threads can be used to monitor activity the other thread information sources cannot.

You can always view information about your own threads. To view information about threads being executed for other accounts, you must have the PROCESS privilege.

Each process list entry contains several pieces of information:

- Id is the connection identifier for the client associated with the thread.
- User and Host indicate the account associated with the thread.
- db is the default database for the thread, or NULL if none is selected.

- `Command` and `State` indicate what the thread is doing.

  Most states correspond to very quick operations. If a thread stays in a given state for many seconds, there might be a problem that needs to be investigated.

- `Time` indicates how long the thread has been in its current state. The thread's notion of the current time may be altered in some cases: The thread can change the time with `SET TIMESTAMP = value`. For a thread running on a slave that is processing events from the master, the thread time is set to the time found in the events and thus reflects current time on the master and not the slave.

- `Info` contains the text of the statement being executed by the thread, or `NULL` if it is not executing one. By default, this value contains only the first 100 characters of the statement. To see the complete statements, use `SHOW FULL PROCESSLIST`.

The following sections list the possible `Command` values, and `State` values grouped by category. The meaning for some of these values is self-evident. For others, additional description is provided.

8.12.5.1 Thread Command Values

A thread can have any of the following `Command` values:

- `Binlog Dump`

  This is a thread on a master server for sending binary log contents to a slave server.

- `Change user`

  The thread is executing a change-user operation.

- `Close stmt`

  The thread is closing a prepared statement.

- `Connect`

  A replication slave is connected to its master.

- `Connect Out`

  A replication slave is connecting to its master.

- `Create DB`

  The thread is executing a create-database operation.

- `Daemon`

  This thread is internal to the server, not a thread that services a client connection.

- `Debug`

  The thread is generating debugging information.

- `Delayed insert`

  The thread is a delayed-insert handler.

- `Drop DB`

The thread is executing a drop-database operation.

- `Error`
- `Execute`

  The thread is executing a prepared statement.
- `Fetch`

  The thread is fetching the results from executing a prepared statement.
- `Field List`

  The thread is retrieving information for table columns.
- `Init DB`

  The thread is selecting a default database.
- `Kill`

  The thread is killing another thread.
- `Long Data`

  The thread is retrieving long data in the result of executing a prepared statement.
- `Ping`

  The thread is handling a server-ping request.
- `Prepare`

  The thread is preparing a prepared statement.
- `Processlist`

  The thread is producing information about server threads.
- `Query`

  The thread is executing a statement.
- `Quit`

  The thread is terminating.
- `Refresh`

  The thread is flushing table, logs, or caches, or resetting status variable or replication server information.
- `Register Slave`

  The thread is registering a slave server.
- `Reset stmt`

  The thread is resetting a prepared statement.

- `Set option`

  The thread is setting or resetting a client statement-execution option.

- `Shutdown`

  The thread is shutting down the server.

- `Sleep`

  The thread is waiting for the client to send a new statement to it.

- `Statistics`

  The thread is producing server-status information.

- `Table Dump`

  The thread is sending table contents to a slave server.

- `Time`

  Unused.

8.12.5.2 General Thread States

The following list describes thread `State` values that are associated with general query processing and not more specialized activities such as replication. Many of these are useful only for finding bugs in the server.

- `After create`

  This occurs when the thread creates a table (including internal temporary tables), at the end of the function that creates the table. This state is used even if the table could not be created due to some error.

- `Analyzing`

  The thread is calculating a `MyISAM` table key distributions (for example, for `ANALYZE TABLE`).

- `checking permissions`

  The thread is checking whether the server has the required privileges to execute the statement.

- `Checking table`

  The thread is performing a table check operation.

- `cleaning up`

  The thread has processed one command and is preparing to free memory and reset certain state variables.

- `closing tables`

  The thread is flushing the changed table data to disk and closing the used tables. This should be a fast operation. If not, verify that you do not have a full disk and that the disk is not in very heavy use.

- `converting HEAP to MyISAM`

The thread is converting an internal temporary table from a MEMORY table to an on-disk MyISAM table.

- copy to tmp table

  The thread is processing an ALTER TABLE statement. This state occurs after the table with the new structure has been created but before rows are copied into it.

- Copying to group table

  If a statement has different ORDER BY and GROUP BY criteria, the rows are sorted by group and copied to a temporary table.

- Copying to tmp table

  The server is copying to a temporary table in memory.

- altering table

  The server is in the process of executing an in-place ALTER TABLE.

- Copying to tmp table on disk

  The server is copying to a temporary table on disk. The temporary result set has become too large (see Section 8.4.4, “How MySQL Uses Internal Temporary Tables”). Consequently, the thread is changing the temporary table from in-memory to disk-based format to save memory.

- Creating index

  The thread is processing ALTER TABLE ... ENABLE KEYS for a MyISAM table.

- Creating sort index

  The thread is processing a SELECT that is resolved using an internal temporary table.

- creating table

  The thread is creating a table. This includes creation of temporary tables.

- Creating tmp table

  The thread is creating a temporary table in memory or on disk. If the table is created in memory but later is converted to an on-disk table, the state during that operation will be Copying to tmp table on disk.

- committing alter table to storage engine

  The server has finished an in-place ALTER TABLE and is committing the result.

- deleting from main table

  The server is executing the first part of a multiple-table delete. It is deleting only from the first table, and saving columns and offsets to be used for deleting from the other (reference) tables.

- deleting from reference tables

  The server is executing the second part of a multiple-table delete and deleting the matched rows from the other tables.

- discard_or_import_tablespace

The thread is processing an `ALTER TABLE ... DISCARD TABLESPACE` or `ALTER TABLE ... IMPORT TABLESPACE` statement.

- `end`

  This occurs at the end but before the cleanup of `ALTER TABLE`, `CREATE VIEW`, `DELETE`, `INSERT`, `SELECT`, or `UPDATE` statements.

- `executing`

  The thread has begun executing a statement.

- `Execution of init_command`

  The thread is executing statements in the value of the `init_command` system variable.

- `freeing items`

  The thread has executed a command. Some freeing of items done during this state involves the query cache. This state is usually followed by `cleaning up`.

- `Flushing tables`

  The thread is executing `FLUSH TABLES` and is waiting for all threads to close their tables.

- `FULLTEXT initialization`

  The server is preparing to perform a natural-language full-text search.

- `init`

  This occurs before the initialization of `ALTER TABLE`, `DELETE`, `INSERT`, `SELECT`, or `UPDATE` statements. Actions taken by the server in this state include flushing the binary log, the `InnoDB` log, and some query cache cleanup operations.

  For the `end` state, the following operations could be happening:

  - Removing query cache entries after data in a table is changed
  - Writing an event to the binary log
  - Freeing memory buffers, including for blobs

- `Killed`

  Someone has sent a `KILL` statement to the thread and it should abort next time it checks the kill flag. The flag is checked in each major loop in MySQL, but in some cases it might still take a short time for the thread to die. If the thread is locked by some other thread, the kill takes effect as soon as the other thread releases its lock.

- `logging slow query`

  The thread is writing a statement to the slow-query log.

- `NULL`

  This state is used for the `SHOW PROCESSLIST` state.

- `login`

The initial state for a connection thread until the client has been authenticated successfully.

- `manage keys`

  The server is enabling or disabling a table index.

- `Opening tables`, `Opening table`

  The thread is trying to open a table. This is should be very fast procedure, unless something prevents opening. For example, an `ALTER TABLE` or a `LOCK TABLE` statement can prevent opening a table until the statement is finished. It is also worth checking that your `table_open_cache` value is large enough.

- `optimizing`

  The server is performing initial optimizations for a query.

- `preparing`

  This state occurs during query optimization.

- `Purging old relay logs`

  The thread is removing unneeded relay log files.

- `query end`

  This state occurs after processing a query but before the `freeing items` state.

- `Reading from net`

  The server is reading a packet from the network.

- `Removing duplicates`

  The query was using `SELECT DISTINCT` in such a way that MySQL could not optimize away the distinct operation at an early stage. Because of this, MySQL requires an extra stage to remove all duplicated rows before sending the result to the client.

- `removing tmp table`

  The thread is removing an internal temporary table after processing a `SELECT` statement. This state is not used if no temporary table was created.

- `rename`

  The thread is renaming a table.

- `rename result table`

  The thread is processing an `ALTER TABLE` statement, has created the new table, and is renaming it to replace the original table.

- `Reopen tables`

  The thread got a lock for the table, but noticed after getting the lock that the underlying table structure changed. It has freed the lock, closed the table, and is trying to reopen it.

- `Repair by sorting`

The repair code is using a sort to create indexes.

- `preparing for alter table`

  The server is preparing to execute an in-place `ALTER TABLE`.

- `Repair done`

  The thread has completed a multi-threaded repair for a `MyISAM` table.

- `Repair with keycache`

  The repair code is using creating keys one by one through the key cache. This is much slower than `Repair by sorting`.

- `Rolling back`

  The thread is rolling back a transaction.

- `Saving state`

  For `MyISAM` table operations such as repair or analysis, the thread is saving the new table state to the `.MYI` file header. State includes information such as number of rows, the `AUTO_INCREMENT` counter, and key distributions.

- `Searching rows for update`

  The thread is doing a first phase to find all matching rows before updating them. This has to be done if the `UPDATE` is changing the index that is used to find the involved rows.

- `Sending data`

  The thread is reading and processing rows for a `SELECT` statement, and sending data to the client. Because operations occurring during this this state tend to perform large amounts of disk access (reads), it is often the longest-running state over the lifetime of a given query.

- `setup`

  The thread is beginning an `ALTER TABLE` operation.

- `Sorting for group`

  The thread is doing a sort to satisfy a `GROUP BY`.

- `Sorting for order`

  The thread is doing a sort to satisfy a `ORDER BY`.

- `Sorting index`

  The thread is sorting index pages for more efficient access during a `MyISAM` table optimization operation.

- `Sorting result`

  For a `SELECT` statement, this is similar to `Creating sort index`, but for nontemporary tables.

- `statistics`

The server is calculating statistics to develop a query execution plan. If a thread is in this state for a long time, the server is probably disk-bound performing other work.

- System lock

  The thread is going to request or is waiting for an internal or external system lock for the table. If this state is being caused by requests for external locks and you are not using multiple mysqld servers that are accessing the same MyISAM tables, you can disable external system locks with the --skip-external-locking option. However, external locking is disabled by default, so it is likely that this option will have no effect. For SHOW PROFILE, this state means the thread is requesting the lock (not waiting for it).

- update

  The thread is getting ready to start updating the table.

- Updating

  The thread is searching for rows to update and is updating them.

- updating main table

  The server is executing the first part of a multiple-table update. It is updating only the first table, and saving columns and offsets to be used for updating the other (reference) tables.

- updating reference tables

  The server is executing the second part of a multiple-table update and updating the matched rows from the other tables.

- User lock

  The thread is going to request or is waiting for an advisory lock requested with a GET_LOCK() call. For SHOW PROFILE, this state means the thread is requesting the lock (not waiting for it).

- User sleep

  The thread has invoked a SLEEP() call.

- Waiting for commit lock

  FLUSH TABLES WITH READ LOCK is waiting for a commit lock.

- Waiting for global read lock

  FLUSH TABLES WITH READ LOCK is waiting for a global read lock or the global read_lock system variable is being set.

- Waiting for tables, Waiting for table flush

  The thread got a notification that the underlying structure for a table has changed and it needs to reopen the table to get the new structure. However, to reopen the table, it must wait until all other threads have closed the table in question.

  This notification takes place if another thread has used FLUSH TABLES or one of the following statements on the table in question: FLUSH TABLES *tbl_name*, ALTER TABLE, RENAME TABLE, REPAIR TABLE, ANALYZE TABLE, or OPTIMIZE TABLE.

- `Waiting for lock_type lock`

  The server is waiting to acquire a lock, where *`lock_type`* indicates the type of lock:

  - `Waiting for event metadata lock`
  - `Waiting for global read lock`
  - `Waiting for schema metadata lock`
  - `Waiting for stored function metadata lock`
  - `Waiting for stored procedure metadata lock`
  - `Waiting for table level lock`
  - `Waiting for table metadata lock`
  - `Waiting for trigger metadata lock`

- `Waiting on cond`

  A generic state in which the thread is waiting for a condition to become true. No specific state information is available.

- `Writing to net`

  The server is writing a packet to the network.

8.12.5.3 Query Cache Thread States

These thread states are associated with the query cache (see Section 8.9.3, “The MySQL Query Cache”).

- `checking privileges on cached query`

  The server is checking whether the user has privileges to access a cached query result.

- `checking query cache for query`

  The server is checking whether the current query is present in the query cache.

- `invalidating query cache entries`

  Query cache entries are being marked invalid because the underlying tables have changed.

- `sending cached result to client`

  The server is taking the result of a query from the query cache and sending it to the client.

- `storing result in query cache`

  The server is storing the result of a query in the query cache.

- `Waiting for query cache lock`

  This state occurs while a session is waiting to take the query cache lock. This can happen for any statement that needs to perform some query cache operation, such as an `INSERT` or `DELETE` that invalidates the query cache, a `SELECT` that looks for a cached entry, `RESET QUERY CACHE`, and so forth.

8.12.5.4 Replication Master Thread States

The following list shows the most common states you may see in the `State` column for the master's `Binlog Dump` thread. If you see no `Binlog Dump` threads on a master server, this means that replication is not running—that is, that no slaves are currently connected.

- `Sending binlog event to slave`

  Binary logs consist of *events*, where an event is usually an update plus some other information. The thread has read an event from the binary log and is now sending it to the slave.

- `Finished reading one binlog; switching to next binlog`

  The thread has finished reading a binary log file and is opening the next one to send to the slave.

- `Master has sent all binlog to slave; waiting for binlog to be updated`

  The thread has read all outstanding updates from the binary logs and sent them to the slave. The thread is now idle, waiting for new events to appear in the binary log resulting from new updates occurring on the master.

- `Waiting to finalize termination`

  A very brief state that occurs as the thread is stopping.

8.12.5.5 Replication Slave I/O Thread States

The following list shows the most common states you see in the `State` column for a slave server I/O thread. This state also appears in the `Slave_IO_State` column displayed by `SHOW SLAVE STATUS`, so you can get a good view of what is happening by using that statement.

- `Waiting for master update`

  The initial state before `Connecting to master`.

- `Connecting to master`

  The thread is attempting to connect to the master.

- `Checking master version`

  A state that occurs very briefly, after the connection to the master is established.

- `Registering slave on master`

  A state that occurs very briefly after the connection to the master is established.

- `Requesting binlog dump`

  A state that occurs very briefly, after the connection to the master is established. The thread sends to the master a request for the contents of its binary logs, starting from the requested binary log file name and position.

- `Waiting to reconnect after a failed binlog dump request`

  If the binary log dump request failed (due to disconnection), the thread goes into this state while it sleeps, then tries to reconnect periodically. The interval between retries can be specified using the `CHANGE MASTER TO` statement.

- `Reconnecting after a failed binlog dump request`

  The thread is trying to reconnect to the master.

- `Waiting for master to send event`

  The thread has connected to the master and is waiting for binary log events to arrive. This can last for a long time if the master is idle. If the wait lasts for `slave_net_timeout` seconds, a timeout occurs. At that point, the thread considers the connection to be broken and makes an attempt to reconnect.

- `Queueing master event to the relay log`

  The thread has read an event and is copying it to the relay log so that the SQL thread can process it.

- `Waiting to reconnect after a failed master event read`

  An error occurred while reading (due to disconnection). The thread is sleeping for the number of seconds set by the `CHANGE MASTER TO` statement (default 60) before attempting to reconnect.

- `Reconnecting after a failed master event read`

  The thread is trying to reconnect to the master. When connection is established again, the state becomes `Waiting for master to send event`.

- `Waiting for the slave SQL thread to free enough relay log space`

  You are using a nonzero `relay_log_space_limit` value, and the relay logs have grown large enough that their combined size exceeds this value. The I/O thread is waiting until the SQL thread frees enough space by processing relay log contents so that it can delete some relay log files.

- `Waiting for slave mutex on exit`

  A state that occurs briefly as the thread is stopping.

8.12.5.6 Replication Slave SQL Thread States

The following list shows the most common states you may see in the `State` column for a slave server SQL thread:

- `Waiting for the next event in relay log`

  The initial state before `Reading event from the relay log`.

- `Reading event from the relay log`

  The thread has read an event from the relay log so that the event can be processed.

- `Making temp file`

  The thread is executing a `LOAD DATA INFILE` statement and is creating a temporary file containing the data from which the slave will read rows.

- `Slave has read all relay log; waiting for the slave I/O thread to update it`

  The thread has processed all events in the relay log files, and is now waiting for the I/O thread to write new events to the relay log.

- `Waiting for slave mutex on exit`

A very brief state that occurs as the thread is stopping.

- `Waiting until MASTER_DELAY seconds after master executed event`

  The SQL thread has read an event but is waiting for the slave delay to lapse. This delay is set with the `MASTER_DELAY` option of `CHANGE MASTER TO`.

The `State` column for the I/O thread may also show the text of a statement. This indicates that the thread has read an event from the relay log, extracted the statement from it, and is executing it.

8.12.5.7 Replication Slave Connection Thread States

These thread states occur on a replication slave but are associated with connection threads, not with the I/O or SQL threads.

- `Changing master`

  The thread is processing a `CHANGE MASTER TO` statement.

- `Killing slave`

  The thread is processing a `STOP SLAVE` statement.

- `Opening master dump table`

  This state occurs after `Creating table from master dump`.

- `Reading master dump table data`

  This state occurs after `Opening master dump table`.

- `Rebuilding the index on master dump table`

  This state occurs after `Reading master dump table data`.

8.12.5.8 Event Scheduler Thread States

These states occur for the Event Scheduler thread, threads that are created to execute scheduled events, or threads that terminate the scheduler.

- `Clearing`

  The scheduler thread or a thread that was executing an event is terminating and is about to end.

- `Initialized`

  The scheduler thread or a thread that will execute an event has been initialized.

- `Waiting for next activation`

  The scheduler has a nonempty event queue but the next activation is in the future.

- `Waiting for scheduler to stop`

  The thread issued `SET GLOBAL event_scheduler=OFF` and is waiting for the scheduler to stop.

- `Waiting on empty queue`

The scheduler's event queue is empty and it is sleeping.

Chapter 9 Language Structure

Table of Contents

This chapter discusses the rules for writing the following elements of SQL statements when using MySQL:

- Literal values such as strings and numbers
- Identifiers such as database, table, and column names
- Reserved words
- User-defined and system variables
- Comments

9.1 Literal Values

This section describes how to write literal values in MySQL. These include strings, numbers, hexadecimal values, boolean values, and `NULL`. The section also covers the various nuances and “gotchas” that you may run into when dealing with these basic types in MySQL.

9.1.1 String Literals

A string is a sequence of bytes or characters, enclosed within either single quote (“`'`”) or double quote (“`"`”) characters. Examples:

```
'a string'
"another string"
```

Quoted strings placed next to each other are concatenated to a single string. The following lines are equivalent:

```
'a string'
'a' ' ' 'string'
```

If the ANSI_QUOTES SQL mode is enabled, string literals can be quoted only within single quotation marks because a string quoted within double quotation marks is interpreted as an identifier.

A *binary string* is a string of bytes that has no character set or collation. A *nonbinary string* is a string of characters that has a character set and collation. For both types of strings, comparisons are based on the numeric values of the string unit. For binary strings, the unit is the byte. For nonbinary strings the unit is the character and some character sets support multi-byte characters. Character value ordering is a function of the string collation.

String literals may have an optional character set introducer and COLLATE clause:

```
[_charset_name]'string' [COLLATE collation_name]
```

Examples:

```
SELECT _latin1'string';
SELECT _latin1'string' COLLATE latin1_danish_ci;
```

You can use N'*literal*' (or n'*literal*') to create a string in the national character set. These statements are equivalent:

```
SELECT N'some text';
SELECT n'some text';
SELECT _utf8'some text';
```

For more information about these forms of string syntax, see Section 10.1.3.5, “Character String Literal Character Set and Collation”, and Section 10.1.3.6, “National Character Set”.

Within a string, certain sequences have special meaning unless the NO_BACKSLASH_ESCAPES SQL mode is enabled. Each of these sequences begins with a backslash (“\”), known as the *escape character*. MySQL recognizes the escape sequences shown in Table 9.1, “Special Character Escape Sequences”. For all other escape sequences, backslash is ignored. That is, the escaped character is interpreted as if it was not escaped. For example, “\x” is just “x”. These sequences are case sensitive. For example, “\b” is interpreted as a backspace, but “\B” is interpreted as “B”. Escape processing is done according to the character set indicated by the character_set_connection system variable. This is true even for strings that are preceded by an introducer that indicates a different character set, as discussed in Section 10.1.3.5, “Character String Literal Character Set and Collation”.

Table 9.1 Special Character Escape Sequences

Escape Sequence	Character Represented by Sequence
\0	An ASCII NUL (0x00) character.
\'	A single quote (“'”) character.
\"	A double quote (“"”) character.
\b	A backspace character.
\n	A newline (linefeed) character.
\r	A carriage return character.
\t	A tab character.
\Z	ASCII 26 (Control+Z). See note following the table.
\\	A backslash (“\”) character.
\%	A “%” character. See note following the table.

Escape Sequence	Character Represented by Sequence
`\_`	A "`_`" character. See note following the table.

The ASCII 26 character can be encoded as "`\Z`" to enable you to work around the problem that ASCII 26 stands for END-OF-FILE on Windows. ASCII 26 within a file causes problems if you try to use `mysql` *`db_name < file_name`*.

The "`\%`" and "`\_`" sequences are used to search for literal instances of "`%`" and "`_`" in pattern-matching contexts where they would otherwise be interpreted as wildcard characters. See the description of the `LIKE` operator in Section 12.5.1, "String Comparison Functions". If you use "`\%`" or "`\_`" outside of pattern-matching contexts, they evaluate to the strings "`\%`" and "`\_`", not to "`%`" and "`_`".

There are several ways to include quote characters within a string:

- A "`'`" inside a string quoted with "`'`" may be written as "`''`".
- A "`"`" inside a string quoted with "`"`" may be written as "`""`".
- Precede the quote character by an escape character ("`\`").
- A "`'`" inside a string quoted with "`"`" needs no special treatment and need not be doubled or escaped. In the same way, "`"`" inside a string quoted with "`'`" needs no special treatment.

The following `SELECT` statements demonstrate how quoting and escaping work:

```
mysql> SELECT 'hello', '"hello"', '""hello""', 'hel''lo', '\'hello';
+-------+---------+-----------+--------+--------+
| hello | "hello" | ""hello"" | hel'lo | 'hello |
+-------+---------+-----------+--------+--------+

mysql> SELECT "hello", "'hello'", "''hello''", "hel""lo", "\"hello";
+-------+---------+-----------+--------+--------+
| hello | 'hello' | ''hello'' | hel"lo | "hello |
+-------+---------+-----------+--------+--------+

mysql> SELECT 'This\nIs\nFour\nLines';
+--------------------+
| This
Is
Four
Lines |
+--------------------+

mysql> SELECT 'disappearing\ backslash';
+------------------------+
| disappearing backslash |
+------------------------+
```

If you want to insert binary data into a string column (such as a `BLOB` column), you should represent certain characters by escape sequences. Backslash ("`\`") and the quote character used to quote the string must be escaped. In certain client environments, it may also be necessary to escape `NUL` or Control+Z. The `mysql` client truncates quoted strings containing `NUL` characters if they are not escaped, and Control+Z may be taken for END-OF-FILE on Windows if not escaped. For the escape sequences that represent each of these characters, see Table 9.1, "Special Character Escape Sequences".

When writing application programs, any string that might contain any of these special characters must be properly escaped before the string is used as a data value in an SQL statement that is sent to the MySQL server. You can do this in two ways:

- Process the string with a function that escapes the special characters. In a C program, you can use the `mysql_real_escape_string()` C API function to escape characters. See Section 21.8.7.55, “`mysql_real_escape_string()`”. Within SQL statements that construct other SQL statements, you can use the `QUOTE()` function. The Perl DBI interface provides a `quote` method to convert special characters to the proper escape sequences. See Section 21.10, “MySQL Perl API”. Other language interfaces may provide a similar capability.

- As an alternative to explicitly escaping special characters, many MySQL APIs provide a placeholder capability that enables you to insert special markers into a statement string, and then bind data values to them when you issue the statement. In this case, the API takes care of escaping special characters in the values for you.

9.1.2 Number Literals

Number literals include exact-value (integer and `DECIMAL`) literals and approximate-value (floating-point) literals.

Integers are represented as a sequence of digits. Numbers may include “`.`” as a decimal separator. Numbers may be preceded by “`-`” or “`+`” to indicate a negative or positive value, respectively. Numbers represented in scientific notation with a mantissa and exponent are approximate-value numbers.

Exact-value numeric literals have an integer part or fractional part, or both. They may be signed. Examples: `1`, `.2`, `3.4`, `-5`, `-6.78`, `+9.10`.

Approximate-value numeric literals are represented in scientific notation with a mantissa and exponent. Either or both parts may be signed. Examples: `1.2E3`, `1.2E-3`, `-1.2E3`, `-1.2E-3`.

Two numbers that look similar may be treated differently. For example, `2.34` is an exact-value (fixed-point) number, whereas `2.34E0` is an approximate-value (floating-point) number.

The `DECIMAL` data type is a fixed-point type and calculations are exact. In MySQL, the `DECIMAL` type has several synonyms: `NUMERIC`, `DEC`, `FIXED`. The integer types also are exact-value types. For more information about exact-value calculations, see Section 12.19, “Precision Math”.

The `FLOAT` and `DOUBLE` data types are floating-point types and calculations are approximate. In MySQL, types that are synonymous with `FLOAT` or `DOUBLE` are `DOUBLE PRECISION` and `REAL`.

An integer may be used in a floating-point context; it is interpreted as the equivalent floating-point number.

9.1.3 Date and Time Literals

Date and time values can be represented in several formats, such as quoted strings or as numbers, depending on the exact type of the value and other factors. For example, in contexts where MySQL expects a date, it interprets any of `'2015-07-21'`, `'20150721'`, and `20150721` as a date.

This section describes the acceptable formats for date and time literals. For more information about the temporal data types, such as the range of permitted values, consult these sections:

- Section 11.1.2, “Date and Time Type Overview”

- Section 11.3, “Date and Time Types”

Standard SQL and ODBC Date and Time Literals. Standard SQL permits temporal literals to be specified using a type keyword and a string. The space between the keyword and string is optional.

```
DATE 'str'
TIME 'str'
TIMESTAMP 'str'
```

MySQL recognizes those constructions and also the corresponding ODBC syntax:

```
{ d 'str' }
{ t 'str' }
{ ts 'str' }
```

MySQL uses the type keyword and these constructions produce `DATE`, `TIME`, and `DATETIME` values, respectively, including a trailing fractional seconds part if specified. The `TIMESTAMP` syntax produces a `DATETIME` value in MySQL because `DATETIME` has a range that more closely corresponds to the standard SQL `TIMESTAMP` type, which has a year range from `0001` to `9999`. (The MySQL `TIMESTAMP` year range is `1970` to `2038`.)

String and Numeric Literals in Date and Time Context. MySQL recognizes `DATE` values in these formats:

- As a string in either `'YYYY-MM-DD'` or `'YY-MM-DD'` format. A “relaxed” syntax is permitted: Any punctuation character may be used as the delimiter between date parts. For example, `'2012-12-31'`, `'2012/12/31'`, `'2012^12^31'`, and `'2012@12@31'` are equivalent.
- As a string with no delimiters in either `'YYYYMMDD'` or `'YYMMDD'` format, provided that the string makes sense as a date. For example, `'20070523'` and `'070523'` are interpreted as `'2007-05-23'`, but `'071332'` is illegal (it has nonsensical month and day parts) and becomes `'0000-00-00'`.
- As a number in either `YYYYMMDD` or `YYMMDD` format, provided that the number makes sense as a date. For example, `19830905` and `830905` are interpreted as `'1983-09-05'`.

MySQL recognizes `DATETIME` and `TIMESTAMP` values in these formats:

- As a string in either `'YYYY-MM-DD HH:MM:SS'` or `'YY-MM-DD HH:MM:SS'` format. A “relaxed” syntax is permitted here, too: Any punctuation character may be used as the delimiter between date parts or time parts. For example, `'2012-12-31 11:30:45'`, `'2012^12^31 11+30+45'`, `'2012/12/31 11*30*45'`, and `'2012@12@31 11^30^45'` are equivalent.

  The only delimiter recognized between a date and time part and a fractional seconds part is the decimal point.

  The date and time parts can be separated by `T` rather than a space. For example, `'2012-12-31 11:30:45'` `'2012-12-31T11:30:45'` are equivalent.
- As a string with no delimiters in either `'YYYYMMDDHHMMSS'` or `'YYMMDDHHMMSS'` format, provided that the string makes sense as a date. For example, `'20070523091528'` and `'070523091528'` are interpreted as `'2007-05-23 09:15:28'`, but `'071122129015'` is illegal (it has a nonsensical minute part) and becomes `'0000-00-00 00:00:00'`.
- As a number in either `YYYYMMDDHHMMSS` or `YYMMDDHHMMSS` format, provided that the number makes sense as a date. For example, `19830905132800` and `830905132800` are interpreted as `'1983-09-05 13:28:00'`.

A `DATETIME` or `TIMESTAMP` value can include a trailing fractional seconds part in up to microseconds (6 digits) precision. The fractional part should always be separated from the rest of the time by a decimal point; no other fractional seconds delimiter is recognized. For information about fractional seconds support in MySQL, see Section 11.3.6, “Fractional Seconds in Time Values”.

Dates containing two-digit year values are ambiguous because the century is unknown. MySQL interprets two-digit year values using these rules:

- Year values in the range `70-99` are converted to `1970-1999`.

- Year values in the range 00-69 are converted to 2000-2069.

See also Section 11.3.8, “Two-Digit Years in Dates”.

For values specified as strings that include date part delimiters, it is unnecessary to specify two digits for month or day values that are less than 10. '2015-6-9' is the same as '2015-06-09'. Similarly, for values specified as strings that include time part delimiters, it is unnecessary to specify two digits for hour, minute, or second values that are less than 10. '2015-10-30 1:2:3' is the same as '2015-10-30 01:02:03'.

Values specified as numbers should be 6, 8, 12, or 14 digits long. If a number is 8 or 14 digits long, it is assumed to be in YYYYMMDD or YYYYMMDDHHMMSS format and that the year is given by the first 4 digits. If the number is 6 or 12 digits long, it is assumed to be in YYMMDD or YYMMDDHHMMSS format and that the year is given by the first 2 digits. Numbers that are not one of these lengths are interpreted as though padded with leading zeros to the closest length.

Values specified as nondelimited strings are interpreted according their length. For a string 8 or 14 characters long, the year is assumed to be given by the first 4 characters. Otherwise, the year is assumed to be given by the first 2 characters. The string is interpreted from left to right to find year, month, day, hour, minute, and second values, for as many parts as are present in the string. This means you should not use strings that have fewer than 6 characters. For example, if you specify '9903', thinking that represents March, 1999, MySQL converts it to the “zero” date value. This occurs because the year and month values are 99 and 03, but the day part is completely missing. However, you can explicitly specify a value of zero to represent missing month or day parts. For example, to insert the value '1999-03-00', use '990300'.

MySQL recognizes TIME values in these formats:

- As a string in 'D HH:MM:SS' format. You can also use one of the following “relaxed” syntaxes: 'HH:MM:SS', 'HH:MM', 'D HH:MM', 'D HH', or 'SS'. Here D represents days and can have a value from 0 to 34.
- As a string with no delimiters in 'HHMMSS' format, provided that it makes sense as a time. For example, '101112' is understood as '10:11:12', but '109712' is illegal (it has a nonsensical minute part) and becomes '00:00:00'.
- As a number in HHMMSS format, provided that it makes sense as a time. For example, 101112 is understood as '10:11:12'. The following alternative formats are also understood: SS, MMSS, or HHMMSS.

A trailing fractional seconds part is recognized in the 'D HH:MM:SS.fraction', 'HH:MM:SS.fraction', 'HHMMSS.fraction', and HHMMSS.fraction time formats, where fraction is the fractional part in up to microseconds (6 digits) precision. The fractional part should always be separated from the rest of the time by a decimal point; no other fractional seconds delimiter is recognized. For information about fractional seconds support in MySQL, see Section 11.3.6, “Fractional Seconds in Time Values”.

For TIME values specified as strings that include a time part delimiter, it is unnecessary to specify two digits for hours, minutes, or seconds values that are less than 10. '8:3:2' is the same as '08:03:02'.

9.1.4 Hexadecimal Literals

MySQL supports hexadecimal values, written using X'*val*', x'*val*', or 0x*val* format, where *val* contains hexadecimal digits (0..9, A..F). Lettercase of the digits does not matter. For values written using X'*val*' or x'*val*' format, *val* must contain an even number of digits. For values written using 0x*val* syntax, values that contain an odd number of digits are treated as having an extra leading 0. For example, 0x0a and 0xaaa are interpreted as 0x0a and 0x0aaa.

In numeric contexts, hexadecimal values act like integers (64-bit precision). In string contexts, they act like binary strings, where each pair of hex digits is converted to a character:

```
mysql> SELECT X'4D7953514C';
        -> 'MySQL'
mysql> SELECT 0x0a+0;
        -> 10
mysql> SELECT 0x5061756c;
        -> 'Paul'
```

The default type of a hexadecimal value is a string. If you want to ensure that the value is treated as a number, you can use `CAST(... AS UNSIGNED)`:

```
mysql> SELECT 0x41, CAST(0x41 AS UNSIGNED);
        -> 'A', 65
```

The `X'`*`hexstring`*`'` syntax is based on standard SQL. The `0x` syntax is based on ODBC. Hexadecimal strings are often used by ODBC to supply values for `BLOB` columns.

You can convert a string or a number to a string in hexadecimal format with the `HEX()` function:

```
mysql> SELECT HEX('cat');
        -> '636174'
mysql> SELECT 0x636174;
        -> 'cat'
```

9.1.5 Boolean Literals

The constants `TRUE` and `FALSE` evaluate to `1` and `0`, respectively. The constant names can be written in any lettercase.

```
mysql> SELECT TRUE, true, FALSE, false;
        -> 1, 1, 0, 0
```

9.1.6 Bit-Field Literals

Bit-field values can be written using `b'`*`value`*`'` or `0b`*`value`* notation. *`value`* is a binary value written using zeros and ones.

Bit-field notation is convenient for specifying values to be assigned to `BIT` columns:

```
mysql> CREATE TABLE t (b BIT(8));
mysql> INSERT INTO t SET b = b'11111111';
mysql> INSERT INTO t SET b = b'1010';
mysql> INSERT INTO t SET b = b'0101';
```

Bit values are returned as binary values. To display them in printable form, add 0 or use a conversion function such as `BIN()`. High-order 0 bits are not displayed in the converted value.

```
mysql> SELECT b+0, BIN(b+0), OCT(b+0), HEX(b+0) FROM t;
+------+----------+----------+----------+
| b+0  | BIN(b+0) | OCT(b+0) | HEX(b+0) |
+------+----------+----------+----------+
|  255 | 11111111 | 377      | FF       |
|   10 | 1010     | 12       | A        |
|    5 | 101      | 5        | 5        |
+------+----------+----------+----------+
```

Bit values assigned to user variables are treated as binary strings. To assign a bit value as a number to a user variable, use `CAST()` or `+0`:

```
mysql> SET @v1 = 0b1000001;
mysql> SET @v2 = CAST(0b1000001 AS UNSIGNED), @v3 = 0b1000001+0;
mysql> SELECT @v1, @v2, @v3;
+------+------+------+
| @v1  | @v2  | @v3  |
+------+------+------+
| A    |   65 |   65 |
+------+------+------+
```

9.1.7 `NULL` Values

The `NULL` value means “no data.” `NULL` can be written in any lettercase. A synonym is `\N` (case sensitive).

For text file import or export operations performed with `LOAD DATA INFILE` or `SELECT ... INTO OUTFILE`, `NULL` is represented by the `\N` sequence. See Section 13.2.6, “`LOAD DATA INFILE` Syntax”.

Be aware that the `NULL` value is different from values such as `0` for numeric types or the empty string for string types. For more information, see Section C.5.5.3, “Problems with `NULL` Values”.

9.2 Schema Object Names

Certain objects within MySQL, including database, table, index, column, alias, view, stored procedure, partition, tablespace, and other object names are known as identifiers. This section describes the permissible syntax for identifiers in MySQL. Section 9.2.2, “Identifier Case Sensitivity”, describes which types of identifiers are case sensitive and under what conditions.

An identifier may be quoted or unquoted. If an identifier contains special characters or is a reserved word, you *must* quote it whenever you refer to it. (Exception: A reserved word that follows a period in a qualified name must be an identifier, so it need not be quoted.) Reserved words are listed at Section 9.3, “Reserved Words”.

Identifiers are converted to Unicode internally. They may contain these characters:

- Permitted characters in unquoted identifiers:
  - ASCII: [0-9,a-z,A-Z$_] (basic Latin letters, digits 0-9, dollar, underscore)
  - Extended: U+0080 .. U+FFFF
- Permitted characters in quoted identifiers include the full Unicode Basic Multilingual Plane (BMP), except U+0000:
  - ASCII: U+0001 .. U+007F
  - Extended: U+0080 .. U+FFFF
- ASCII NUL (U+0000) and supplementary characters (U+10000 and higher) are not permitted in quoted or unquoted identifiers.
- Identifiers may begin with a digit but unless quoted may not consist solely of digits.
- Database, table, and column names cannot end with space characters.

The identifier quote character is the backtick (“`` ` ``”):

```
mysql> SELECT * FROM `select` WHERE `select`.id > 100;
```

If the `ANSI_QUOTES` SQL mode is enabled, it is also permissible to quote identifiers within double quotation marks:

```
mysql> CREATE TABLE "test" (col INT);
ERROR 1064: You have an error in your SQL syntax...
mysql> SET sql_mode='ANSI_QUOTES';
mysql> CREATE TABLE "test" (col INT);
Query OK, 0 rows affected (0.00 sec)
```

The `ANSI_QUOTES` mode causes the server to interpret double-quoted strings as identifiers. Consequently, when this mode is enabled, string literals must be enclosed within single quotation marks. They cannot be enclosed within double quotation marks. The server SQL mode is controlled as described in Section 5.1.7, “Server SQL Modes”.

Identifier quote characters can be included within an identifier if you quote the identifier. If the character to be included within the identifier is the same as that used to quote the identifier itself, then you need to double the character. The following statement creates a table named ``a`b`` that contains a column named `c"d`:

```
mysql> CREATE TABLE `a``b` (`c"d` INT);
```

In the select list of a query, a quoted column alias can be specified using identifier or string quoting characters:

```
mysql> SELECT 1 AS `one`, 2 AS 'two';
+-----+-----+
| one | two |
+-----+-----+
|   1 |   2 |
+-----+-----+
```

Elsewhere in the statement, quoted references to the alias must use identifier quoting or the reference is treated as a string literal.

It is recommended that you do not use names that begin with *`Me`* or *`MeN`*, where *`M`* and *`N`* are integers. For example, avoid using `1e` as an identifier, because an expression such as `1e+3` is ambiguous. Depending on context, it might be interpreted as the expression `1e + 3` or as the number `1e+3`.

Be careful when using `MD5()` to produce table names because it can produce names in illegal or ambiguous formats such as those just described.

A user variable cannot be used directly in an SQL statement as an identifier or as part of an identifier. See Section 9.4, “User-Defined Variables”, for more information and examples of workarounds.

Special characters in database and table names are encoded in the corresponding file system names as described in Section 9.2.3, “Mapping of Identifiers to File Names”. If you have databases or tables from an older version of MySQL that contain special characters and for which the underlying directory names or file names have not been updated to use the new encoding, the server displays their names with a prefix of `#mysql50#`. For information about referring to such names or converting them to the newer encoding, see that section.

The following table describes the maximum length for each type of identifier.

Identifier	Maximum Length (characters)
Database	64
Table	64
Column	64
Index	64
Constraint	64
Stored Program	64
View	64
Tablespace	64
Server	64
Log File Group	64
Alias	256 (see exception following table)
Compound Statement Label	16

Aliases for column names in `CREATE VIEW` statements are checked against the maximum column length of 64 characters (not the maximum alias length of 256 characters).

Identifiers are stored using Unicode (UTF-8). This applies to identifiers in table definitions that are stored in `.frm` files and to identifiers stored in the grant tables in the `mysql` database. The sizes of the identifier string columns in the grant tables are measured in characters. You can use multi-byte characters without reducing the number of characters permitted for values stored in these columns, something not true prior to MySQL 4.1. As indicated earlier, the permissible Unicode characters are those in the Basic Multilingual Plane (BMP). Supplementary characters are not permitted.

9.2.1 Identifier Qualifiers

MySQL permits names that consist of a single identifier or multiple identifiers. The components of a multiple-part name must be separated by period (“`.`”) characters. The initial parts of a multiple-part name act as qualifiers that affect the context within which the final identifier is interpreted.

In MySQL, you can refer to a table column using any of the following forms.

Column Reference	Meaning
`col_name`	The column *`col_name`* from whichever table used in the statement contains a column of that name.
`tbl_name.col_name`	The column *`col_name`* from table *`tbl_name`* of the default database.
`db_name.tbl_name.col_name`	The column *`col_name`* from table *`tbl_name`* of the database *`db_name`*.

The qualifier character is a separate token and need not be contiguous with the associated identifiers. For example, *`tbl_name.col_name`* and *`tbl_name . col_name`* are equivalent.

If any components of a multiple-part name require quoting, quote them individually rather than quoting the name as a whole. For example, write `` `my-table`.`my-column` ``, not `` `my-table.my-column` ``.

A reserved word that follows a period in a qualified name must be an identifier, so in that context it need not be quoted.

You need not specify a *`tbl_name`* or *`db_name.tbl_name`* prefix for a column reference in a statement unless the reference would be ambiguous. Suppose that tables `t1` and `t2` each contain a column `c`, and you retrieve `c` in a `SELECT` statement that uses both `t1` and `t2`. In this case, `c` is ambiguous because it is not unique among the tables used in the statement. You must qualify it with a table name as `t1.c` or `t2.c` to indicate which table you mean. Similarly, to retrieve from a table `t` in database `db1` and from a table `t` in database `db2` in the same statement, you must refer to columns in those tables as `db1.t.col_name` and `db2.t.col_name`.

The syntax `.tbl_name` means the table `tbl_name` in the default database. This syntax is accepted for ODBC compatibility because some ODBC programs prefix table names with a "`.`" character.

9.2.2 Identifier Case Sensitivity

In MySQL, databases correspond to directories within the data directory. Each table within a database corresponds to at least one file within the database directory (and possibly more, depending on the storage engine). Triggers also correspond to files. Consequently, the case sensitivity of the underlying operating system plays a part in the case sensitivity of database, table, and trigger names. This means such names are not case sensitive in Windows, but are case sensitive in most varieties of Unix. One notable exception is Mac OS X, which is Unix-based but uses a default file system type (HFS+) that is not case sensitive. However, Mac OS X also supports UFS volumes, which are case sensitive just as on any Unix. See Section 1.8.1, "MySQL Extensions to Standard SQL". The `lower_case_table_names` system variable also affects how the server handles identifier case sensitivity, as described later in this section.

Note

Although database, table, and trigger names are not case sensitive on some platforms, you should not refer to one of these using different cases within the same statement. The following statement would not work because it refers to a table both as `my_table` and as `MY_TABLE`:

```
mysql> SELECT * FROM my_table WHERE MY_TABLE.col=1;
```

Column, index, stored routine, and event names are not case sensitive on any platform, nor are column aliases.

However, names of logfile groups are case sensitive. This differs from standard SQL.

By default, table aliases are case sensitive on Unix, but not so on Windows or Mac OS X. The following statement would not work on Unix, because it refers to the alias both as `a` and as `A`:

```
mysql> SELECT col_name FROM tbl_name AS a
    -> WHERE a.col_name = 1 OR A.col_name = 2;
```

However, this same statement is permitted on Windows. To avoid problems caused by such differences, it is best to adopt a consistent convention, such as always creating and referring to databases and tables using lowercase names. This convention is recommended for maximum portability and ease of use.

How table and database names are stored on disk and used in MySQL is affected by the `lower_case_table_names` system variable, which you can set when starting `mysqld`. `lower_case_table_names` can take the values shown in the following table. This variable does *not* affect case sensitivity of trigger identifiers. On Unix, the default value of `lower_case_table_names` is 0. On Windows the default value is 1. On Mac OS X, the default value is 2.

Value	Meaning
`0`	Table and database names are stored on disk using the lettercase specified in the `CREATE TABLE` or `CREATE DATABASE` statement. Name comparisons are case sensitive. You should *not* set this variable to 0 if you are running MySQL on a system that has case-insensitive file names (such as Windows or Mac OS X). If you force this variable to 0 with `--lower-case-`

Value	Meaning
	table-names=0 on a case-insensitive file system and access MyISAM tablenames using different lettercases, index corruption may result.
1	Table names are stored in lowercase on disk and name comparisons are not case sensitive. MySQL converts all table names to lowercase on storage and lookup. This behavior also applies to database names and table aliases.
2	Table and database names are stored on disk using the lettercase specified in the CREATE TABLE or CREATE DATABASE statement, but MySQL converts them to lowercase on lookup. Name comparisons are not case sensitive. This works *only* on file systems that are not case sensitive! InnoDB table names are stored in lowercase, as for lower_case_table_names=1.

If you are using MySQL on only one platform, you do not normally have to change the lower_case_table_names variable from its default value. However, you may encounter difficulties if you want to transfer tables between platforms that differ in file system case sensitivity. For example, on Unix, you can have two different tables named my_table and MY_TABLE, but on Windows these two names are considered identical. To avoid data transfer problems arising from lettercase of database or table names, you have two options:

- Use lower_case_table_names=1 on all systems. The main disadvantage with this is that when you use SHOW TABLES or SHOW DATABASES, you do not see the names in their original lettercase.
- Use lower_case_table_names=0 on Unix and lower_case_table_names=2 on Windows. This preserves the lettercase of database and table names. The disadvantage of this is that you must ensure that your statements always refer to your database and table names with the correct lettercase on Windows. If you transfer your statements to Unix, where lettercase is significant, they do not work if the lettercase is incorrect.

  Exception: If you are using InnoDB tables and you are trying to avoid these data transfer problems, you should set lower_case_table_names to 1 on all platforms to force names to be converted to lowercase.

If you plan to set the lower_case_table_names system variable to 1 on Unix, you must first convert your old database and table names to lowercase before stopping mysqld and restarting it with the new variable setting. To do this for an individual table, use RENAME TABLE:

```
RENAME TABLE T1 TO t1;
```

To convert one or more entire databases, dump them before setting lower_case_table_names, then drop the databases, and reload them after setting lower_case_table_names:

1. Use mysqldump to dump each database:

   ```
   mysqldump --databases db1 > db1.sql
   mysqldump --databases db2 > db2.sql
   ...
   ```

   Do this for each database that must be recreated.

2. Use DROP DATABASE to drop each database.

3. Stop the server, set lower_case_table_names, and restart the server.

4. Reload the dump file for each database. Because lower_case_table_names is set, each database and table name will be converted to lowercase as it is recreated:

```
mysql < db1.sql
mysql < db2.sql
...
```

Object names may be considered duplicates if their uppercase forms are equal according to a binary collation. That is true for names of cursors, conditions, procedures, functions, savepoints, stored routine parameters, stored program local variables, and plugins. It is not true for names of columns, constraints, databases, partitions, statements prepared with `PREPARE`, tables, triggers, users, and user-defined variables.

File system case sensitivity can affect searches in string columns of `INFORMATION_SCHEMA` tables. For more information, see Section 10.1.7.9, "Collation and `INFORMATION_SCHEMA` Searches".

9.2.3 Mapping of Identifiers to File Names

There is a correspondence between database and table identifiers and names in the file system. For the basic structure, MySQL represents each database as a directory in the data directory, and each table by one or more files in the appropriate database directory. For the table format files (`.FRM`), the data is always stored in this structure and location.

For the data and index files, the exact representation on disk is storage engine specific. These files may be stored in the same location as the `FRM` files, or the information may be stored in a separate file. `InnoDB` data is stored in the InnoDB data files. If you are using tablespaces with `InnoDB`, then the specific tablespace files you create are used instead.

Any character is legal in database or table identifiers except ASCII NUL (`0x00`). MySQL encodes any characters that are problematic in the corresponding file system objects when it creates database directories or table files:

- Basic Latin letters (`a..zA..Z`), digits (`0..9`) and underscore (`_`) are encoded as is. Consequently, their case sensitivity directly depends on file system features.
- All other national letters from alphabets that have uppercase/lowercase mapping are encoded as shown in the following table. Values in the Code Range column are UCS-2 values.

Code Range	Pattern	Number	Used	Unused	Blocks
00C0..017F	[@][0..4][g..z]	5*20= 100	97	3	Latin-1 Supplement + Latin Extended-A
0370..03FF	[@][5..9][g..z]	5*20= 100	88	12	Greek and Coptic
0400..052F	[@][g..z][0..6]	20*7= 140	137	3	Cyrillic + Cyrillic Supplement
0530..058F	[@][g..z][7..8]	20*2= 40	38	2	Armenian
2160..217F	[@][g..z][9]	20*1= 20	16	4	Number Forms
0180..02AF	[@][g..z][a..k]	20*11=220	203	17	Latin Extended-B + IPA Extensions
1E00..1EFF	[@][g..z][l..r]	20*7= 140	136	4	Latin Extended Additional
1F00..1FFF	[@][g..z][s..z]	20*8= 160	144	16	Greek Extended
....	[@][a..f][g..z]	6*20= 120	0	120	RESERVED
24B6..24E9	[@][@][a..z]	26	26	0	Enclosed Alphanumerics

Code Range	Pattern	Number	Used	Unused	Blocks
FF21..FF5A	[@][a..z][@]	26	26	0	Halfwidth and Fullwidth forms

One of the bytes in the sequence encodes lettercase. For example: LATIN CAPITAL LETTER A WITH GRAVE is encoded as @0G, whereas LATIN SMALL LETTER A WITH GRAVE is encoded as @0g. Here the third byte (G or g) indicates lettercase. (On a case-insensitive file system, both letters will be treated as the same.)

For some blocks, such as Cyrillic, the second byte determines lettercase. For other blocks, such as Latin1 Supplement, the third byte determines lettercase. If two bytes in the sequence are letters (as in Greek Extended), the leftmost letter character stands for lettercase. All other letter bytes must be in lowercase.

- All nonletter characters except underscore (_), as well as letters from alphabets that do not have uppercase/lowercase mapping (such as Hebrew) are encoded using hexadecimal representation using lowercase letters for hex digits a..f:

```
0x003F -> @003f
0xFFFF -> @ffff
```

The hexadecimal values correspond to character values in the ucs2 double-byte character set.

On Windows, some names such as nul, prn, and aux are encoded by appending @@@ to the name when the server creates the corresponding file or directory. This occurs on all platforms for portability of the corresponding database object between platforms.

If you have databases or tables from a version of MySQL older than 5.1.6 that contain special characters and for which the underlying directory names or file names have not been updated to use the new encoding, the server displays their names with a prefix of #mysql50# in the output from INFORMATION_SCHEMA tables or SHOW statements. For example, if you have a table named a@b and its name encoding has not been updated, SHOW TABLES displays it like this:

```
mysql> SHOW TABLES;
+----------------+
| Tables_in_test |
+----------------+
| #mysql50#a@b   |
+----------------+
```

To refer to such a name for which the encoding has not been updated, you must supply the #mysql50# prefix:

```
mysql> SHOW COLUMNS FROM `a@b`;
ERROR 1146 (42S02): Table 'test.a@b' doesn't exist

mysql> SHOW COLUMNS FROM `#mysql50#a@b`;
+-------+---------+------+-----+---------+-------+
| Field | Type    | Null | Key | Default | Extra |
+-------+---------+------+-----+---------+-------+
| i     | int(11) | YES  |     | NULL    |       |
+-------+---------+------+-----+---------+-------+
```

To update old names to eliminate the need to use the special prefix to refer to them, re-encode them with mysqlcheck. The following command updates all names to the new encoding:

```
shell> mysqlcheck --check-upgrade --fix-db-names --fix-table-names --all-databases
```

To check only specific databases or tables, omit `--all-databases` and provide the appropriate database or table arguments. For information about `mysqlcheck` invocation syntax, see Section 4.5.3, "`mysqlcheck` — A Table Maintenance Program".

> **Note**
>
> The `#mysql50#` prefix is intended only to be used internally by the server. You should not create databases or tables with names that use this prefix.
>
> Also, `mysqlcheck` cannot fix names that contain literal instances of the `@` character that is used for encoding special characters. If you have databases or tables that contain this character, use `mysqldump` to dump them before upgrading to MySQL 5.1.6 or later, and then reload the dump file after upgrading.

9.2.4 Function Name Parsing and Resolution

MySQL 5.7 supports built-in (native) functions, user-defined functions (UDFs), and stored functions. This section describes how the server recognizes whether the name of a built-in function is used as a function call or as an identifier, and how the server determines which function to use in cases when functions of different types exist with a given name.

Built-In Function Name Parsing

The parser uses default rules for parsing names of built-in functions. These rules can be changed by enabling the `IGNORE_SPACE` SQL mode.

When the parser encounters a word that is the name of a built-in function, it must determine whether the name signifies a function call or is instead a nonexpression reference to an identifier such as a table or column name. For example, in the following statements, the first reference to `count` is a function call, whereas the second reference is a table name:

```
SELECT COUNT(*) FROM mytable;
CREATE TABLE count (i INT);
```

The parser should recognize the name of a built-in function as indicating a function call only when parsing what is expected to be an expression. That is, in nonexpression context, function names are permitted as identifiers.

However, some built-in functions have special parsing or implementation considerations, so the parser uses the following rules by default to distinguish whether their names are being used as function calls or as identifiers in nonexpression context:

- To use the name as a function call in an expression, there must be no whitespace between the name and the following "`(`" parenthesis character.
- Conversely, to use the function name as an identifier, it must not be followed immediately by a parenthesis.

The requirement that function calls be written with no whitespace between the name and the parenthesis applies only to the built-in functions that have special considerations. `COUNT` is one such name. The exact list of function names for which following whitespace determines their interpretation are those listed in the `sql_functions[]` array of the `sql/lex.h` source file. Before MySQL 5.1, these are rather numerous (about 200), so you may find it easiest to treat the no-whitespace requirement as applying to all function calls. In MySQL 5.1 and later, parser improvements reduce to about 30 the number of affected function names.

For functions not listed in the `sql_functions[]`) array, whitespace does not matter. They are interpreted as function calls only when used in expression context and may be used freely as identifiers otherwise. `ASCII` is one such name. However, for these nonaffected function names, interpretation may vary in expression context: *`func_name`* `()` is interpreted as a built-in function if there is one with the given name; if not, *`func_name`* `()` is interpreted as a user-defined function or stored function if one exists with that name.

The `IGNORE_SPACE` SQL mode can be used to modify how the parser treats function names that are whitespace-sensitive:

- With `IGNORE_SPACE` disabled, the parser interprets the name as a function call when there is no whitespace between the name and the following parenthesis. This occurs even when the function name is used in nonexpression context:

  ```
  mysql> CREATE TABLE count(i INT);
  ERROR 1064 (42000): You have an error in your SQL syntax ...
  near 'count(i INT)'
  ```

  To eliminate the error and cause the name to be treated as an identifier, either use whitespace following the name or write it as a quoted identifier (or both):

  ```
  CREATE TABLE count (i INT);
  CREATE TABLE `count`(i INT);
  CREATE TABLE `count` (i INT);
  ```

- With `IGNORE_SPACE` enabled, the parser loosens the requirement that there be no whitespace between the function name and the following parenthesis. This provides more flexibility in writing function calls. For example, either of the following function calls are legal:

  ```
  SELECT COUNT(*) FROM mytable;
  SELECT COUNT (*) FROM mytable;
  ```

  However, enabling `IGNORE_SPACE` also has the side effect that the parser treats the affected function names as reserved words (see Section 9.3, “Reserved Words”). This means that a space following the name no longer signifies its use as an identifier. The name can be used in function calls with or without following whitespace, but causes a syntax error in nonexpression context unless it is quoted. For example, with `IGNORE_SPACE` enabled, both of the following statements fail with a syntax error because the parser interprets `count` as a reserved word:

  ```
  CREATE TABLE count(i INT);
  CREATE TABLE count (i INT);
  ```

  To use the function name in nonexpression context, write it as a quoted identifier:

  ```
  CREATE TABLE `count`(i INT);
  CREATE TABLE `count` (i INT);
  ```

To enable the `IGNORE_SPACE` SQL mode, use this statement:

```
SET sql_mode = 'IGNORE_SPACE';
```

`IGNORE_SPACE` is also enabled by certain other composite modes such as `ANSI` that include it in their value:

```
SET sql_mode = 'ANSI';
```

Check Section 5.1.7, “Server SQL Modes”, to see which composite modes enable IGNORE_SPACE.

To minimize the dependency of SQL code on the IGNORE_SPACE setting, use these guidelines:

- Avoid creating UDFs or stored functions that have the same name as a built-in function.
- Avoid using function names in nonexpression context. For example, these statements use count (one of the affected function names affected by IGNORE_SPACE), so they fail with or without whitespace following the name if IGNORE_SPACE is enabled:

  ```
  CREATE TABLE count(i INT);
  CREATE TABLE count (i INT);
  ```

  If you must use a function name in nonexpression context, write it as a quoted identifier:

  ```
  CREATE TABLE `count`(i INT);
  CREATE TABLE `count` (i INT);
  ```

The number of function names affected by IGNORE_SPACE was reduced significantly in MySQL 5.1.13, from about 200 to about 30. As of MySQL 5.1.13, only the following functions are still affected by the IGNORE_SPACE setting.

ADDDATE	BIT_AND	BIT_OR	BIT_XOR
CAST	COUNT	CURDATE	CURTIME
DATE_ADD	DATE_SUB	EXTRACT	GROUP_CONCAT
MAX	MID	MIN	NOW
POSITION	SESSION_USER	STD	STDDEV
STDDEV_POP	STDDEV_SAMP	SUBDATE	SUBSTR
SUBSTRING	SUM	SYSDATE	SYSTEM_USER
TRIM	VARIANCE	VAR_POP	VAR_SAMP

For earlier versions of MySQL, check the contents of the sql_functions[] array in the sql/lex.h source file to see which functions are affected by IGNORE_SPACE.

Incompatibility warning: The change in MySQL 5.1.13 that reduces the number of function names affected by IGNORE_SPACE improves the consistency of parser operation. However, it also introduces the possibility of incompatibility for old SQL code that relies on the following conditions:

- IGNORE_SPACE is disabled.
- The presence or absence of whitespace following a function name is used to distinguish between a built-in function and stored function that have the same name, such as PI() versus PI ().

For functions that are no longer affected by IGNORE_SPACE as of MySQL 5.1.13, that strategy no longer works. Either of the following approaches can be used if you have code that is subject to the preceding incompatibility:

- If a stored function has a name that conflicts with a built-in function, refer to the stored function with a schema name qualifier, regardless of whether whitespace is present. For example, write *schema_name*.PI() or *schema_name*.PI ().
- Alternatively, rename the stored function to use a nonconflicting name and change invocations of the function to use the new name.

Function Name Resolution

The following rules describe how the server resolves references to function names for function creation and invocation:

- Built-in functions and user-defined functions

  An error occurs if you try to create a UDF with the same name as a built-in function.

- Built-in functions and stored functions

  It is possible to create a stored function with the same name as a built-in function, but to invoke the stored function it is necessary to qualify it with a schema name. For example, if you create a stored function named `PI` in the `test` schema, you invoke it as `test.PI()` because the server resolves `PI()` as a reference to the built-in function. The server creates a warning if the stored function name collides with a built-in function name. The warning can be displayed with `SHOW WARNINGS`.

- User-defined functions and stored functions

  User-defined functions and stored functions share the same namespace, so you cannot create a UDF and a stored function with the same name.

The preceding function name resolution rules have implications for upgrading to versions of MySQL that implement new built-in functions:

- If you have already created a user-defined function with a given name and upgrade MySQL to a version that implements a new built-in function with the same name, the UDF becomes inaccessible. To correct this, use `DROP FUNCTION` to drop the UDF, and then use `CREATE FUNCTION` to re-create the UDF with a different nonconflicting name.

- If a new version of MySQL implements a built-in function with the same name as an existing stored function, you have two choices: Rename the stored function to use a nonconflicting name, or change calls to the function so that they use a schema qualifier (that is, use *`schema_name`*`.`*`func_name`*`()` syntax).

9.3 Reserved Words

Certain words such as `SELECT`, `DELETE`, or `BIGINT` are reserved and require special treatment for use as identifiers such as table and column names. This may also be true for the names of built-in functions.

Reserved words are permitted as identifiers if you quote them as described in Section 9.2, “Schema Object Names”:

```
mysql> CREATE TABLE interval (begin INT, end INT);
ERROR 1064 (42000): You have an error in your SQL syntax ...
near 'interval (begin INT, end INT)'

mysql> CREATE TABLE `interval` (begin INT, end INT);
Query OK, 0 rows affected (0.01 sec)
```

Exception: A word that follows a period in a qualified name must be an identifier, so it need not be quoted even if it is reserved:

```
mysql> CREATE TABLE mydb.interval (begin INT, end INT);
Query OK, 0 rows affected (0.01 sec)
```

Names of built-in functions are permitted as identifiers but may require care to be used as such. For example, `COUNT` is acceptable as a column name. However, by default, no whitespace is permitted in function invocations between the function name and the following "(" character. This requirement enables the parser to distinguish whether the name is used in a function call or in nonfunction context. For further detail on recognition of function names, see Section 9.2.4, "Function Name Parsing and Resolution".

The words in the following table are explicitly reserved in MySQL 5.7. In addition, _FILENAME is reserved. At some point, you might upgrade to a higher version, so it is a good idea to have a look at future reserved words, too. You can find these in the manuals that cover higher versions of MySQL. Most of the words in the table are forbidden by standard SQL as column or table names (for example, GROUP). A few are reserved because MySQL needs them and uses a yacc parser. A reserved word can be used as an identifier if you quote it.

For a more detailed list of reserved words, including differences between versions, see Reserved Words in MySQL 5.7.

Table 9.2 Reserved Words in MySQL 5.7.5

ACCESSIBLE	ADD	ALL
ALTER	ANALYZE	AND
AS	ASC	ASENSITIVE
BEFORE	BETWEEN	BIGINT
BINARY	BLOB	BOTH
BY	CALL	CASCADE
CASE	CHANGE	CHAR
CHARACTER	CHECK	COLLATE
COLUMN	CONDITION	CONSTRAINT
CONTINUE	CONVERT	CREATE
CROSS	CURRENT_DATE	CURRENT_TIME
CURRENT_TIMESTAMP	CURRENT_USER	CURSOR
DATABASE	DATABASES	DAY_HOUR
DAY_MICROSECOND	DAY_MINUTE	DAY_SECOND
DEC	DECIMAL	DECLARE
DEFAULT	DELAYED	DELETE
DESC	DESCRIBE	DETERMINISTIC
DISTINCT	DISTINCTROW	DIV
DOUBLE	DROP	DUAL
EACH	ELSE	ELSEIF
ENCLOSED	ESCAPED	EXISTS
EXIT	EXPLAIN	FALSE
FETCH	FLOAT	FLOAT4
FLOAT8	FOR	FORCE
FOREIGN	FROM	FULLTEXT
GET	GRANT	GROUP
HAVING	HIGH_PRIORITY	HOUR_MICROSECOND
HOUR_MINUTE	HOUR_SECOND	IF
IGNORE	IN	INDEX

INFILE	INNER	INOUT
INSENSITIVE	INSERT	INT
INT1	INT2	INT3
INT4	INT8	INTEGER
INTERVAL	INTO	IO_AFTER_GTIDS
IO_BEFORE_GTIDS	IS	ITERATE
JOIN	KEY	KEYS
KILL	LEADING	LEAVE
LEFT	LIKE	LIMIT
LINEAR	LINES	LOAD
LOCALTIME	LOCALTIMESTAMP	LOCK
LONG	LONGBLOB	LONGTEXT
LOOP	LOW_PRIORITY	MASTER_BIND
MASTER_SSL_VERIFY_SERVER_CERT	MATCH	MAXVALUE
MEDIUMBLOB	MEDIUMINT	MEDIUMTEXT
MIDDLEINT	MINUTE_MICROSECOND	MINUTE_SECOND
MOD	MODIFIES	NATURAL
NONBLOCKING	NOT	NO_WRITE_TO_BINLOG
NULL	NUMERIC	ON
OPTIMIZE	OPTION	OPTIONALLY
OR	ORDER	OUT
OUTER	OUTFILE	PARTITION
PRECISION	PRIMARY	PROCEDURE
PURGE	RANGE	READ
READS	READ_WRITE	REAL
REFERENCES	REGEXP	RELEASE
RENAME	REPEAT	REPLACE
REQUIRE	RESIGNAL	RESTRICT
RETURN	REVOKE	RIGHT
RLIKE	SCHEMA	SCHEMAS
SECOND_MICROSECOND	SELECT	SENSITIVE
SEPARATOR	SET	SHOW
SIGNAL	SMALLINT	SPATIAL
SPECIFIC	SQL	SQLEXCEPTION
SQLSTATE	SQLWARNING	SQL_BIG_RESULT
SQL_CALC_FOUND_ROWS	SQL_SMALL_RESULT	SSL
STARTING	STRAIGHT_JOIN	TABLE
TERMINATED	THEN	TINYBLOB
TINYINT	TINYTEXT	TO

TRAILING	TRIGGER	TRUE
UNDO	UNION	UNIQUE
UNLOCK	UNSIGNED	UPDATE
USAGE	USE	USING
UTC_DATE	UTC_TIME	UTC_TIMESTAMP
VALUES	VARBINARY	VARCHAR
VARCHARACTER	VARYING	WHEN
WHERE	WHILE	WITH
WRITE	XOR	YEAR_MONTH
ZEROFILL		

Table 9.3 New Reserved Words in MySQL 5.7

NONBLOCKING		

MySQL permits some keywords to be used as unquoted identifiers because many people previously used them. Examples are those in the following list:

- ACTION
- BIT
- DATE
- ENUM
- NO
- TEXT
- TIME
- TIMESTAMP

9.4 User-Defined Variables

You can store a value in a user-defined variable in one statement and then refer to it later in another statement. This enables you to pass values from one statement to another. *User-defined variables are session-specific*. That is, a user variable defined by one client cannot be seen or used by other clients. All variables for a given client session are automatically freed when that client exits.

User variables are written as @*var_name*, where the variable name *var_name* consists of alphanumeric characters, ".", "_", and "$". A user variable name can contain other characters if you quote it as a string or identifier (for example, @'my-var', @"my-var", or @`my-var`).

User variable names are not case sensitive in MySQL 5.0 and up.

One way to set a user-defined variable is by issuing a SET statement:

```
SET @var_name = expr [, @var_name = expr] ...
```

For `SET`, either `=` or `:=` can be used as the assignment operator.

You can also assign a value to a user variable in statements other than `SET`. In this case, the assignment operator must be `:=` and not `=` because the latter is treated as the comparison operator `=` in non-`SET` statements:

```
mysql> SET @t1=1, @t2=2, @t3:=4;
mysql> SELECT @t1, @t2, @t3, @t4 := @t1+@t2+@t3;
+------+------+------+--------------------+
| @t1  | @t2  | @t3  | @t4 := @t1+@t2+@t3 |
+------+------+------+--------------------+
|    1 |    2 |    4 |                  7 |
+------+------+------+--------------------+
```

User variables can be assigned a value from a limited set of data types: integer, decimal, floating-point, binary or nonbinary string, or `NULL` value. Assignment of decimal and real values does not preserve the precision or scale of the value. A value of a type other than one of the permissible types is converted to a permissible type. For example, a value having a temporal or spatial data type is converted to a binary string.

If a user variable is assigned a nonbinary (character) string value, it has the same character set and collation as the string. The coercibility of user variables is implicit. (This is the same coercibility as for table column values.)

Bit values assigned to user variables are treated as binary strings. To assign a bit value as a number to a user variable, use `CAST()` or `+0`:

```
mysql> SET @v1 = b'1000001';
mysql> SET @v2 = CAST(b'1000001' AS UNSIGNED), @v3 = b'1000001'+0;
mysql> SELECT @v1, @v2, @v3;
+------+------+------+
| @v1  | @v2  | @v3  |
+------+------+------+
| A    |   65 |   65 |
+------+------+------+
```

If the value of a user variable is selected in a result set, it is returned to the client as a string.

If you refer to a variable that has not been initialized, it has a value of `NULL` and a type of string.

User variables may be used in most contexts where expressions are permitted. This does not currently include contexts that explicitly require a literal value, such as in the `LIMIT` clause of a `SELECT` statement, or the `IGNORE N LINES` clause of a `LOAD DATA` statement.

As a general rule, other than in `SET` statements, you should never assign a value to a user variable and read the value within the same statement. For example, to increment a variable, this is okay:

```
SET @a = @a + 1;
```

For other statements, such as `SELECT`, you might get the results you expect, but this is not guaranteed. In the following statement, you might think that MySQL will evaluate `@a` first and then do an assignment second:

```
SELECT @a, @a:=@a+1, ...;
```

However, the order of evaluation for expressions involving user variables is undefined.

Another issue with assigning a value to a variable and reading the value within the same non-SET statement is that the default result type of a variable is based on its type at the start of the statement. The following example illustrates this:

```
mysql> SET @a='test';
mysql> SELECT @a,(@a:=20) FROM tbl_name;
```

For this SELECT statement, MySQL reports to the client that column one is a string and converts all accesses of @a to strings, even though @a is set to a number for the second row. After the SELECT statement executes, @a is regarded as a number for the next statement.

To avoid problems with this behavior, either do not assign a value to and read the value of the same variable within a single statement, or else set the variable to 0, 0.0, or '' to define its type before you use it.

In a SELECT statement, each select expression is evaluated only when sent to the client. This means that in a HAVING, GROUP BY, or ORDER BY clause, referring to a variable that is assigned a value in the select expression list does *not* work as expected:

```
mysql> SELECT (@aa:=id) AS a, (@aa+3) AS b FROM tbl_name HAVING b=5;
```

The reference to b in the HAVING clause refers to an alias for an expression in the select list that uses @aa. This does not work as expected: @aa contains the value of id from the previous selected row, not from the current row.

User variables are intended to provide data values. They cannot be used directly in an SQL statement as an identifier or as part of an identifier, such as in contexts where a table or database name is expected, or as a reserved word such as SELECT. This is true even if the variable is quoted, as shown in the following example:

```
mysql> SELECT c1 FROM t;
+----+
| c1 |
+----+
|  0 |
+----+
|  1 |
+----+
2 rows in set (0.00 sec)

mysql> SET @col = "c1";
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @col FROM t;
+------+
| @col |
+------+
| c1   |
+------+
1 row in set (0.00 sec)

mysql> SELECT `@col` FROM t;
ERROR 1054 (42S22): Unknown column '@col' in 'field list'

mysql> SET @col = "`c1`";
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @col FROM t;
+------+
| @col |
+------+
| `c1` |
+------+
1 row in set (0.00 sec)
```

An exception to this principle that user variables cannot be used to provide identifiers, is when you are constructing a string for use as a prepared statement to execute later. In this case, user variables can be used to provide any part of the statement. The following example illustrates how this can be done:

```
mysql> SET @c = "c1";
Query OK, 0 rows affected (0.00 sec)

mysql> SET @s = CONCAT("SELECT ", @c, " FROM t");
Query OK, 0 rows affected (0.00 sec)

mysql> PREPARE stmt FROM @s;
Query OK, 0 rows affected (0.04 sec)
Statement prepared

mysql> EXECUTE stmt;
+----+
| c1 |
+----+
|  0 |
+----+
|  1 |
+----+
2 rows in set (0.00 sec)

mysql> DEALLOCATE PREPARE stmt;
Query OK, 0 rows affected (0.00 sec)
```

See Section 13.5, “SQL Syntax for Prepared Statements”, for more information.

A similar technique can be used in application programs to construct SQL statements using program variables, as shown here using PHP 5:

```
<?php
  $mysqli = new mysqli("localhost", "user", "pass", "test");

  if( mysqli_connect_errno() )
    die("Connection failed: %s\n", mysqli_connect_error());

  $col = "c1";

  $query = "SELECT $col FROM t";

  $result = $mysqli->query($query);

  while($row = $result->fetch_assoc())
  {
    echo "<p>" . $row["$col"] . "</p>\n";
  }

  $result->close();

  $mysqli->close();
?>
```

Assembling an SQL statement in this fashion is sometimes known as “Dynamic SQL”.

9.5 Expression Syntax

The following rules define expression syntax in MySQL. The grammar shown here is based on that given in the `sql/sql_yacc.yy` file of MySQL source distributions. See the notes after the grammar for additional information about some of the terms.

```
expr:
    expr OR expr
  | expr || expr
  | expr XOR expr
  | expr AND expr
  | expr && expr
  | NOT expr
  | ! expr
  | boolean_primary IS [NOT] {TRUE | FALSE | UNKNOWN}
  | boolean_primary

boolean_primary:
    boolean_primary IS [NOT] NULL
  | boolean_primary <=> predicate
  | boolean_primary comparison_operator predicate
  | boolean_primary comparison_operator {ALL | ANY} (subquery)
  | predicate

comparison_operator: = | >= | > | <= | < | <> | !=

predicate:
    bit_expr [NOT] IN (subquery)
  | bit_expr [NOT] IN (expr [, expr] ...)
  | bit_expr [NOT] BETWEEN bit_expr AND predicate
  | bit_expr SOUNDS LIKE bit_expr
  | bit_expr [NOT] LIKE simple_expr [ESCAPE simple_expr]
  | bit_expr [NOT] REGEXP bit_expr
  | bit_expr

bit_expr:
    bit_expr | bit_expr
  | bit_expr & bit_expr
  | bit_expr << bit_expr
  | bit_expr >> bit_expr
  | bit_expr + bit_expr
  | bit_expr - bit_expr
  | bit_expr * bit_expr
  | bit_expr / bit_expr
  | bit_expr DIV bit_expr
  | bit_expr MOD bit_expr
  | bit_expr % bit_expr
  | bit_expr ^ bit_expr
  | bit_expr + interval_expr
  | bit_expr - interval_expr
  | simple_expr

simple_expr:
    literal
  | identifier
  | function_call
  | simple_expr COLLATE collation_name
  | param_marker
  | variable
  | simple_expr || simple_expr
  | + simple_expr
  | - simple_expr
  | ~ simple_expr
  | ! simple_expr
  | BINARY simple_expr
  | (expr [, expr] ...)
  | ROW (expr, expr [, expr] ...)
  | (subquery)
  | EXISTS (subquery)
  | {identifier expr}
  | match_expr
  | case_expr
  | interval_expr
```

Notes:

For operator precedence, see in Section 12.3.1, “Operator Precedence”.

For literal value syntax, see Section 9.1, “Literal Values”.

For identifier syntax, see Section 9.2, “Schema Object Names”.

Variables can be user variables, system variables, or stored program local variables or parameters:

- User variables: Section 9.4, “User-Defined Variables”
- System variables: Section 5.1.5, “Using System Variables”
- Local variables: Section 13.6.4.1, “Local Variable `DECLARE` Syntax”
- Parameters: Section 13.1.12, “`CREATE PROCEDURE` and `CREATE FUNCTION` Syntax”

`param_marker` is `?` as used in prepared statements for placeholders. See Section 13.5.1, “`PREPARE` Syntax”.

`(`*`subquery`*`)` indicates a subquery that returns a single value; that is, a scalar subquery. See Section 13.2.10.1, “The Subquery as Scalar Operand”.

`{`*`identifier expr`*`}` is ODBC escape syntax and is accepted for ODBC compatibility. The value is *`expr`*. The curly braces in the syntax should be written literally; they are not metasyntax as used elsewhere in syntax descriptions.

`match_expr` indicates a `MATCH` [1271] expression. See Section 12.9, “Full-Text Search Functions”.

`case_expr` indicates a `CASE` expression. See Section 12.4, “Control Flow Functions”.

`interval_expr` represents a time interval. The syntax is `INTERVAL` *`expr unit`*, where *`unit`* is a specifier such as `HOUR`, `DAY`, or `WEEK`. For the full list of *`unit`* specifiers, see the description of the `DATE_ADD()` function in Section 12.7, “Date and Time Functions”.

The meaning of some operators depends on the SQL mode:

- By default, `||` is a logical `OR` operator. With `PIPES_AS_CONCAT` enabled, `||` is string concatenation, with a precedence between `^` and the unary operators.
- By default, `!` has a higher precedence than `NOT`. With `HIGH_NOT_PRECEDENCE` enabled, `!` and `NOT` have the same precedence.

See Section 5.1.7, “Server SQL Modes”.

9.6 Comment Syntax

MySQL Server supports three comment styles:

- From a “`#`” character to the end of the line.

- From a “-- ” sequence to the end of the line. In MySQL, the “-- ” (double-dash) comment style requires the second dash to be followed by at least one whitespace or control character (such as a space, tab, newline, and so on). This syntax differs slightly from standard SQL comment syntax, as discussed in Section 1.8.2.5, “'--' as the Start of a Comment”.
- From a /* sequence to the following */ sequence, as in the C programming language. This syntax enables a comment to extend over multiple lines because the beginning and closing sequences need not be on the same line.

The following example demonstrates all three comment styles:

```
mysql> SELECT 1+1;     # This comment continues to the end of line
mysql> SELECT 1+1;     -- This comment continues to the end of line
mysql> SELECT 1 /* this is an in-line comment */ + 1;
mysql> SELECT 1+
/*
this is a
multiple-line comment
*/
1;
```

Nested comments are not supported. (Under some conditions, nested comments might be permitted, but usually are not, and users should avoid them.)

MySQL Server supports some variants of C-style comments. These enable you to write code that includes MySQL extensions, but is still portable, by using comments of the following form:

```
/*! MySQL-specific code */
```

In this case, MySQL Server parses and executes the code within the comment as it would any other SQL statement, but other SQL servers will ignore the extensions. For example, MySQL Server recognizes the STRAIGHT_JOIN keyword in the following statement, but other servers will not:

```
SELECT /*! STRAIGHT_JOIN */ col1 FROM table1,table2 WHERE ...
```

If you add a version number after the “!” character, the syntax within the comment is executed only if the MySQL version is greater than or equal to the specified version number. The TEMPORARY keyword in the following comment is executed only by servers from MySQL 3.23.02 or higher:

```
CREATE /*!32302 TEMPORARY */ TABLE t (a INT);
```

The comment syntax just described applies to how the mysqld server parses SQL statements. The mysql client program also performs some parsing of statements before sending them to the server. (It does this to determine statement boundaries within a multiple-statement input line.)

Comments in this format, /*!12345 ... */, are not stored on the server. If this format is used to comment stored routines, the comments will not be retained on the server.

The use of short-form mysql commands such as \C within multi-line /* ... */ comments is not supported.

Chapter 10 Globalization

Table of Contents

This chapter covers issues of globalization, which includes internationalization (MySQL's capabilities for adapting to local use) and localization (selecting particular local conventions):

- MySQL support for character sets in SQL statements.
- How to configure the server to support different character sets.
- Selecting the language for error messages.
- How to set the server's time zone and enable per-connection time zone support.
- Selecting the locale for day and month names.

10.1 Character Set Support

MySQL includes character set support that enables you to store data using a variety of character sets and perform comparisons according to a variety of collations. You can specify character sets at the server, database, table, and column level. MySQL supports the use of character sets for the MyISAM, MEMORY, and InnoDB storage engines.

This chapter discusses the following topics:

- What are character sets and collations?
- The multiple-level default system for character set assignment.
- Syntax for specifying character sets and collations.
- Affected functions and operations.
- Unicode support.
- The character sets and collations that are available, with notes.

Character set issues affect not only data storage, but also communication between client programs and the MySQL server. If you want the client program to communicate with the server using a character set different from the default, you'll need to indicate which one. For example, to use the utf8 Unicode character set, issue this statement after connecting to the server:

```
SET NAMES 'utf8';
```

For more information about configuring character sets for application use and character set-related issues in client/server communication, see Section 10.1.5, “Configuring the Character Set and Collation for Applications”, and Section 10.1.4, “Connection Character Sets and Collations”.

10.1.1 Character Sets and Collations in General

A *character set* is a set of symbols and encodings. A *collation* is a set of rules for comparing characters in a character set. Let's make the distinction clear with an example of an imaginary character set.

Suppose that we have an alphabet with four letters: “A”, “B”, “a”, “b”. We give each letter a number: “A” = 0, “B” = 1, “a” = 2, “b” = 3. The letter “A” is a symbol, the number 0 is the **encoding** for “A”, and the combination of all four letters and their encodings is a **character set**.

Suppose that we want to compare two string values, “A” and “B”. The simplest way to do this is to look at the encodings: 0 for “A” and 1 for “B”. Because 0 is less than 1, we say “A” is less than “B”. What we've just done is apply a collation to our character set. The collation is a set of rules (only one rule in this case): “compare the encodings.” We call this simplest of all possible collations a *binary* collation.

But what if we want to say that the lowercase and uppercase letters are equivalent? Then we would have at least two rules: (1) treat the lowercase letters “a” and “b” as equivalent to “A” and “B”; (2) then compare the encodings. We call this a *case-insensitive* collation. It is a little more complex than a binary collation.

In real life, most character sets have many characters: not just “A” and “B” but whole alphabets, sometimes multiple alphabets or eastern writing systems with thousands of characters, along with many special symbols and punctuation marks. Also in real life, most collations have many rules, not just for whether to distinguish lettercase, but also for whether to distinguish accents (an “accent” is a mark attached to a character as in German “Ö”), and for multiple-character mappings (such as the rule that “Ö” = “OE” in one of the two German collations).

MySQL can do these things for you:

- Store strings using a variety of character sets.
- Compare strings using a variety of collations.

- Mix strings with different character sets or collations in the same server, the same database, or even the same table.
- Enable specification of character set and collation at any level.

In these respects, MySQL is far ahead of most other database management systems. However, to use these features effectively, you need to know what character sets and collations are available, how to change the defaults, and how they affect the behavior of string operators and functions.

10.1.2 Character Sets and Collations in MySQL

The MySQL server can support multiple character sets. To list the available character sets, use the SHOW CHARACTER SET statement. A partial listing follows. For more complete information, see Section 10.1.14, “Character Sets and Collations That MySQL Supports”.

```
mysql> SHOW CHARACTER SET;
+----------+-----------------------------+---------------------+--------+
| Charset  | Description                 | Default collation   | Maxlen |
+----------+-----------------------------+---------------------+--------+
| big5     | Big5 Traditional Chinese    | big5_chinese_ci     |      2 |
| dec8     | DEC West European           | dec8_swedish_ci     |      1 |
| cp850    | DOS West European           | cp850_general_ci    |      1 |
| hp8      | HP West European            | hp8_english_ci      |      1 |
| koi8r    | KOI8-R Relcom Russian       | koi8r_general_ci    |      1 |
| latin1   | cp1252 West European        | latin1_swedish_ci   |      1 |
| latin2   | ISO 8859-2 Central European | latin2_general_ci   |      1 |
| swe7     | 7bit Swedish                | swe7_swedish_ci     |      1 |
| ascii    | US ASCII                    | ascii_general_ci    |      1 |
| ujis     | EUC-JP Japanese             | ujis_japanese_ci    |      3 |
| sjis     | Shift-JIS Japanese          | sjis_japanese_ci    |      2 |
| hebrew   | ISO 8859-8 Hebrew           | hebrew_general_ci   |      1 |
| tis620   | TIS620 Thai                 | tis620_thai_ci      |      1 |
| euckr    | EUC-KR Korean               | euckr_korean_ci     |      2 |
| koi8u    | KOI8-U Ukrainian            | koi8u_general_ci    |      1 |
| gb2312   | GB2312 Simplified Chinese   | gb2312_chinese_ci   |      2 |
| greek    | ISO 8859-7 Greek            | greek_general_ci    |      1 |
| cp1250   | Windows Central European    | cp1250_general_ci   |      1 |
| gbk      | GBK Simplified Chinese      | gbk_chinese_ci      |      2 |
| latin5   | ISO 8859-9 Turkish          | latin5_turkish_ci   |      1 |
...
```

Any given character set always has at least one collation. It may have several collations. To list the collations for a character set, use the SHOW COLLATION statement. For example, to see the collations for the latin1 (cp1252 West European) character set, use this statement to find those collation names that begin with latin1:

```
mysql> SHOW COLLATION LIKE 'latin1%';
+---------------------+---------+----+---------+----------+---------+
| Collation           | Charset | Id | Default | Compiled | Sortlen |
+---------------------+---------+----+---------+----------+---------+
| latin1_german1_ci   | latin1  |  5 |         |          |       0 |
| latin1_swedish_ci   | latin1  |  8 | Yes     | Yes      |       1 |
| latin1_danish_ci    | latin1  | 15 |         |          |       0 |
| latin1_german2_ci   | latin1  | 31 |         | Yes      |       2 |
| latin1_bin          | latin1  | 47 |         | Yes      |       1 |
| latin1_general_ci   | latin1  | 48 |         |          |       0 |
| latin1_general_cs   | latin1  | 49 |         |          |       0 |
| latin1_spanish_ci   | latin1  | 94 |         |          |       0 |
+---------------------+---------+----+---------+----------+---------+
```

The latin1 collations have the following meanings.

Collation	Meaning
`latin1_german1_ci`	German DIN-1
`latin1_swedish_ci`	Swedish/Finnish
`latin1_danish_ci`	Danish/Norwegian
`latin1_german2_ci`	German DIN-2
`latin1_bin`	Binary according to `latin1` encoding
`latin1_general_ci`	Multilingual (Western European)
`latin1_general_cs`	Multilingual (ISO Western European), case sensitive
`latin1_spanish_ci`	Modern Spanish

Collations have these general characteristics:

- Two different character sets cannot have the same collation.
- Each character set has one collation that is the *default collation*. For example, the default collation for `latin1` is `latin1_swedish_ci`. The output for `SHOW CHARACTER SET` indicates which collation is the default for each displayed character set.
- There is a convention for collation names: They start with the name of the character set with which they are associated, they usually include a language name, and they end with `_ci` (case insensitive), `_cs` (case sensitive), or `_bin` (binary).

In cases where a character set has multiple collations, it might not be clear which collation is most suitable for a given application. To avoid choosing the wrong collation, it can be helpful to perform some comparisons with representative data values to make sure that a given collation sorts values the way you expect.

Collation-Charts.Org is a useful site for information that shows how one collation compares to another.

10.1.3 Specifying Character Sets and Collations

There are default settings for character sets and collations at four levels: server, database, table, and column. The description in the following sections may appear complex, but it has been found in practice that multiple-level defaulting leads to natural and obvious results.

`CHARACTER SET` is used in clauses that specify a character set. `CHARSET` can be used as a synonym for `CHARACTER SET`.

Character set issues affect not only data storage, but also communication between client programs and the MySQL server. If you want the client program to communicate with the server using a character set different from the default, you'll need to indicate which one. For example, to use the `utf8` Unicode character set, issue this statement after connecting to the server:

```
SET NAMES 'utf8';
```

For more information about character set-related issues in client/server communication, see Section 10.1.4, “Connection Character Sets and Collations”.

10.1.3.1 Server Character Set and Collation

MySQL Server has a server character set and a server collation. These can be set at server startup on the command line or in an option file and changed at runtime.

Initially, the server character set and collation depend on the options that you use when you start mysqld. You can use --character-set-server for the character set. Along with it, you can add --collation-server for the collation. If you don't specify a character set, that is the same as saying --character-set-server=latin1. If you specify only a character set (for example, latin1) but not a collation, that is the same as saying --character-set-server=latin1 --collation-server=latin1_swedish_ci because latin1_swedish_ci is the default collation for latin1. Therefore, the following three commands all have the same effect:

```
shell> mysqld
shell> mysqld --character-set-server=latin1
shell> mysqld --character-set-server=latin1 \
           --collation-server=latin1_swedish_ci
```

One way to change the settings is by recompiling. To change the default server character set and collation when building from sources, use the DEFAULT_CHARSET and DEFAULT_COLLATION options for CMake. For example:

```
shell> cmake . -DDEFAULT_CHARSET=latin1
```

Or:

```
shell> cmake . -DDEFAULT_CHARSET=latin1 \
           -DDEFAULT_COLLATION=latin1_german1_ci
```

Both mysqld and CMake verify that the character set/collation combination is valid. If not, each program displays an error message and terminates.

The server character set and collation are used as default values if the database character set and collation are not specified in CREATE DATABASE statements. They have no other purpose.

The current server character set and collation can be determined from the values of the character_set_server and collation_server system variables. These variables can be changed at runtime.

10.1.3.2 Database Character Set and Collation

Every database has a database character set and a database collation. The CREATE DATABASE and ALTER DATABASE statements have optional clauses for specifying the database character set and collation:

```
CREATE DATABASE db_name
    [[DEFAULT] CHARACTER SET charset_name]
    [[DEFAULT] COLLATE collation_name]

ALTER DATABASE db_name
    [[DEFAULT] CHARACTER SET charset_name]
    [[DEFAULT] COLLATE collation_name]
```

The keyword SCHEMA can be used instead of DATABASE.

All database options are stored in a text file named db.opt that can be found in the database directory.

The CHARACTER SET and COLLATE clauses make it possible to create databases with different character sets and collations on the same MySQL server.

Example:

```
CREATE DATABASE db_name CHARACTER SET latin1 COLLATE latin1_swedish_ci;
```

MySQL chooses the database character set and database collation in the following manner:

- If both `CHARACTER SET X` and `COLLATE Y` are specified, character set *X* and collation *Y* are used.

- If `CHARACTER SET X` is specified without `COLLATE`, character set *X* and its default collation are used. To see the default collation for each character set, use the `SHOW COLLATION` statement.

- If `COLLATE Y` is specified without `CHARACTER SET`, the character set associated with *Y* and collation *Y* are used.

- Otherwise, the server character set and server collation are used.

The database character set and collation are used as default values for table definitions if the table character set and collation are not specified in `CREATE TABLE` statements. The database character set also is used by `LOAD DATA INFILE`. The character set and collation have no other purposes.

The character set and collation for the default database can be determined from the values of the `character_set_database` and `collation_database` system variables. The server sets these variables whenever the default database changes. If there is no default database, the variables have the same value as the corresponding server-level system variables, `character_set_server` and `collation_server`.

10.1.3.3 Table Character Set and Collation

Every table has a table character set and a table collation. The `CREATE TABLE` and `ALTER TABLE` statements have optional clauses for specifying the table character set and collation:

```
CREATE TABLE tbl_name (column_list)
    [[DEFAULT] CHARACTER SET charset_name]
    [COLLATE collation_name]]

ALTER TABLE tbl_name
    [[DEFAULT] CHARACTER SET charset_name]
    [COLLATE collation_name]
```

Example:

```
CREATE TABLE t1 ( ... )
CHARACTER SET latin1 COLLATE latin1_danish_ci;
```

MySQL chooses the table character set and collation in the following manner:

- If both `CHARACTER SET X` and `COLLATE Y` are specified, character set *X* and collation *Y* are used.

- If `CHARACTER SET X` is specified without `COLLATE`, character set *X* and its default collation are used. To see the default collation for each character set, use the `SHOW COLLATION` statement.

- If `COLLATE Y` is specified without `CHARACTER SET`, the character set associated with *Y* and collation *Y* are used.

- Otherwise, the database character set and collation are used.

The table character set and collation are used as default values for column definitions if the column character set and collation are not specified in individual column definitions. The table character set and collation are MySQL extensions; there are no such things in standard SQL.

10.1.3.4 Column Character Set and Collation

Every “character” column (that is, a column of type CHAR, VARCHAR, or TEXT) has a column character set and a column collation. Column definition syntax for CREATE TABLE and ALTER TABLE has optional clauses for specifying the column character set and collation:

```
col_name {CHAR | VARCHAR | TEXT} (col_length)
    [CHARACTER SET charset_name]
    [COLLATE collation_name]
```

These clauses can also be used for ENUM and SET columns:

```
col_name {ENUM | SET} (val_list)
    [CHARACTER SET charset_name]
    [COLLATE collation_name]
```

Examples:

```
CREATE TABLE t1
(
    col1 VARCHAR(5)
      CHARACTER SET latin1
      COLLATE latin1_german1_ci
);

ALTER TABLE t1 MODIFY
    col1 VARCHAR(5)
      CHARACTER SET latin1
      COLLATE latin1_swedish_ci;
```

MySQL chooses the column character set and collation in the following manner:

- If both CHARACTER SET *X* and COLLATE *Y* are specified, character set *X* and collation *Y* are used.

  ```
  CREATE TABLE t1
  (
      col1 CHAR(10) CHARACTER SET utf8 COLLATE utf8_unicode_ci
  ) CHARACTER SET latin1 COLLATE latin1_bin;
  ```

  The character set and collation are specified for the column, so they are used. The column has character set utf8 and collation utf8_unicode_ci.

- If CHARACTER SET *X* is specified without COLLATE, character set *X* and its default collation are used.

  ```
  CREATE TABLE t1
  (
      col1 CHAR(10) CHARACTER SET utf8
  ) CHARACTER SET latin1 COLLATE latin1_bin;
  ```

  The character set is specified for the column, but the collation is not. The column has character set utf8 and the default collation for utf8, which is utf8_general_ci. To see the default collation for each character set, use the SHOW COLLATION statement.

- If COLLATE *Y* is specified without CHARACTER SET, the character set associated with *Y* and collation *Y* are used.

  ```
  CREATE TABLE t1
  (
  ```

```
    col1 CHAR(10) COLLATE utf8_polish_ci
) CHARACTER SET latin1 COLLATE latin1_bin;
```

The collation is specified for the column, but the character set is not. The column has collation `utf8_polish_ci` and the character set is the one associated with the collation, which is `utf8`.

- Otherwise, the table character set and collation are used.

```
CREATE TABLE t1
(
    col1 CHAR(10)
) CHARACTER SET latin1 COLLATE latin1_bin;
```

Neither the character set nor collation are specified for the column, so the table defaults are used. The column has character set `latin1` and collation `latin1_bin`.

The `CHARACTER SET` and `COLLATE` clauses are standard SQL.

If you use `ALTER TABLE` to convert a column from one character set to another, MySQL attempts to map the data values, but if the character sets are incompatible, there may be data loss.

10.1.3.5 Character String Literal Character Set and Collation

Every character string literal has a character set and a collation.

A character string literal may have an optional character set introducer and `COLLATE` clause:

```
[_charset_name]'string' [COLLATE collation_name]
```

Examples:

```
SELECT 'string';
SELECT _latin1'string';
SELECT _latin1'string' COLLATE latin1_danish_ci;
```

For the simple statement `SELECT 'string'`, the string has the character set and collation defined by the `character_set_connection` and `collation_connection` system variables.

The *`_charset_name`* expression is formally called an *introducer*. It tells the parser, “the string that is about to follow uses character set *`X`*.” Because this has confused people in the past, we emphasize that an introducer does not change the string to the introducer character set like `CONVERT()` would do. It does not change the string's value, although padding may occur. The introducer is just a signal. An introducer is also legal before standard hex literal and numeric hex literal notation (`x'literal'` and `0xnnnn`), or before bit-field literal notation (`b'literal'` and `0bnnnn`).

Examples:

```
SELECT _latin1 x'AABBCC';
SELECT _latin1 0xAABBCC;
SELECT _latin1 b'1100011';
SELECT _latin1 0b1100011;
```

MySQL determines a literal's character set and collation in the following manner:

- If both *`_X`* and `COLLATE Y` are specified, character set *`X`* and collation *`Y`* are used.
- If *`_X`* is specified but `COLLATE` is not specified, character set *`X`* and its default collation are used. To see the default collation for each character set, use the `SHOW COLLATION` statement.

- Otherwise, the character set and collation given by the character_set_connection and collation_connection system variables are used.

Examples:

- A string with latin1 character set and latin1_german1_ci collation:

```
SELECT _latin1'Müller' COLLATE latin1_german1_ci;
```

- A string with latin1 character set and its default collation (that is, latin1_swedish_ci):

```
SELECT _latin1'Müller';
```

- A string with the connection default character set and collation:

```
SELECT 'Müller';
```

Character set introducers and the COLLATE clause are implemented according to standard SQL specifications.

An introducer indicates the character set for the following string, but does not change now how the parser performs escape processing within the string. Escapes are always interpreted by the parser according to the character set given by character_set_connection.

The following examples show that escape processing occurs using character_set_connection even in the presence of an introducer. The examples use SET NAMES (which changes character_set_connection, as discussed in Section 10.1.4, “Connection Character Sets and Collations”), and display the resulting strings using the HEX() function so that the exact string contents can be seen.

Example 1:

```
mysql> SET NAMES latin1;
Query OK, 0 rows affected (0.01 sec)

mysql> SELECT HEX('à\n'), HEX(_sjis'à\n');
+------------+-----------------+
| HEX('à\n') | HEX(_sjis'à\n') |
+------------+-----------------+
| E00A       | E00A            |
+------------+-----------------+
1 row in set (0.00 sec)
```

Here, “à” (hex value E0) is followed by “\n”, the escape sequence for newline. The escape sequence is interpreted using the character_set_connection value of latin1 to produce a literal newline (hex value 0A). This happens even for the second string. That is, the introducer of _sjis does not affect the parser's escape processing.

Example 2:

```
mysql> SET NAMES sjis;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT HEX('à\n'), HEX(_latin1'à\n');
+------------+-------------------+
| HEX('à\n') | HEX(_latin1'à\n') |
+------------+-------------------+
| E05C6E     | E05C6E             |
+------------+--------------------+
1 row in set (0.04 sec)
```

Here, `character_set_connection` is `sjis`, a character set in which the sequence of “`à`” followed by “`\`” (hex values `05` and `5C`) is a valid multi-byte character. Hence, the first two bytes of the string are interpreted as a single `sjis` character, and the “`\`” is not interpreted as an escape character. The following “`n`” (hex value `6E`) is not interpreted as part of an escape sequence. This is true even for the second string; the introducer of `_latin1` does not affect escape processing.

10.1.3.6 National Character Set

Standard SQL defines `NCHAR` or `NATIONAL CHAR` as a way to indicate that a `CHAR` column should use some predefined character set. MySQL 5.7 uses `utf8` as this predefined character set. For example, these data type declarations are equivalent:

```
CHAR(10) CHARACTER SET utf8
NATIONAL CHARACTER(10)
NCHAR(10)
```

As are these:

```
VARCHAR(10) CHARACTER SET utf8
NATIONAL VARCHAR(10)
NCHAR VARCHAR(10)
NATIONAL CHARACTER VARYING(10)
NATIONAL CHAR VARYING(10)
```

You can use `N'`*`literal`*`'` (or `n'`*`literal`*`'`) to create a string in the national character set. These statements are equivalent:

```
SELECT N'some text';
SELECT n'some text';
SELECT _utf8'some text';
```

For information on upgrading character sets to MySQL 5.7 from versions prior to 4.1, see the *MySQL 3.23, 4.0, 4.1 Reference Manual*.

10.1.3.7 Examples of Character Set and Collation Assignment

The following examples show how MySQL determines default character set and collation values.

Example 1: Table and Column Definition

```
CREATE TABLE t1
(
    c1 CHAR(10) CHARACTER SET latin1 COLLATE latin1_german1_ci
) DEFAULT CHARACTER SET latin2 COLLATE latin2_bin;
```

Here we have a column with a `latin1` character set and a `latin1_german1_ci` collation. The definition is explicit, so that is straightforward. Notice that there is no problem with storing a `latin1` column in a `latin2` table.

Example 2: Table and Column Definition

```
CREATE TABLE t1
(
    c1 CHAR(10) CHARACTER SET latin1
) DEFAULT CHARACTER SET latin1 COLLATE latin1_danish_ci;
```

This time we have a column with a latin1 character set and a default collation. Although it might seem natural, the default collation is not taken from the table level. Instead, because the default collation for latin1 is always latin1_swedish_ci, column c1 has a collation of latin1_swedish_ci (not latin1_danish_ci).

Example 3: Table and Column Definition

```
CREATE TABLE t1
(
    c1 CHAR(10)
) DEFAULT CHARACTER SET latin1 COLLATE latin1_danish_ci;
```

We have a column with a default character set and a default collation. In this circumstance, MySQL checks the table level to determine the column character set and collation. Consequently, the character set for column c1 is latin1 and its collation is latin1_danish_ci.

Example 4: Database, Table, and Column Definition

```
CREATE DATABASE d1
    DEFAULT CHARACTER SET latin2 COLLATE latin2_czech_ci;
USE d1;
CREATE TABLE t1
(
    c1 CHAR(10)
);
```

We create a column without specifying its character set and collation. We're also not specifying a character set and a collation at the table level. In this circumstance, MySQL checks the database level to determine the table settings, which thereafter become the column settings.) Consequently, the character set for column c1 is latin2 and its collation is latin2_czech_ci.

10.1.3.8 Compatibility with Other DBMSs

For MaxDB compatibility these two statements are the same:

```
CREATE TABLE t1 (f1 CHAR(N) UNICODE);
CREATE TABLE t1 (f1 CHAR(N) CHARACTER SET ucs2);
```

10.1.4 Connection Character Sets and Collations

Several character set and collation system variables relate to a client's interaction with the server. Some of these have been mentioned in earlier sections:

- The server character set and collation are the values of the character_set_server and collation_server system variables.

- The character set and collation of the default database are the values of the character_set_database and collation_database system variables.

Additional character set and collation system variables are involved in handling traffic for the connection between a client and the server. Every client has connection-related character set and collation system variables.

A “connection” is what you make when you connect to the server. The client sends SQL statements, such as queries, over the connection to the server. The server sends responses, such as result sets or error messages, over the connection back to the client. This leads to several questions about character set and collation handling for client connections, each of which can be answered in terms of system variables:

- What character set is the statement in when it leaves the client?

  The server takes the `character_set_client` system variable to be the character set in which statements are sent by the client.

- What character set should the server translate a statement to after receiving it?

  For this, the server uses the `character_set_connection` and `collation_connection` system variables. It converts statements sent by the client from `character_set_client` to `character_set_connection` (except for string literals that have an introducer such as `_latin1` or `_utf8`). `collation_connection` is important for comparisons of literal strings. For comparisons of strings with column values, `collation_connection` does not matter because columns have their own collation, which has a higher collation precedence.

- What character set should the server translate to before shipping result sets or error messages back to the client?

  The `character_set_results` system variable indicates the character set in which the server returns query results to the client. This includes result data such as column values, and result metadata such as column names and error messages.

Clients can fine-tune the settings for these variables, or depend on the defaults (in which case, you can skip the rest of this section). If you do not use the defaults, you must change the character settings *for each connection to the server.*

Two statements affect the connection-related character set variables as a group:

- `SET NAMES 'charset_name' [COLLATE 'collation_name']`

  `SET NAMES` indicates what character set the client will use to send SQL statements to the server. Thus, `SET NAMES 'cp1251'` tells the server, “future incoming messages from this client are in character set `cp1251`.” It also specifies the character set that the server should use for sending results back to the client. (For example, it indicates what character set to use for column values if you use a `SELECT` statement.)

  A `SET NAMES 'charset_name'` statement is equivalent to these three statements:

  ```
  SET character_set_client = charset_name;
  SET character_set_results = charset_name;
  SET character_set_connection = charset_name;
  ```

  Setting `character_set_connection` to *`charset_name`* also implicitly sets `collation_connection` to the default collation for *`charset_name`*. It is unnecessary to set that collation explicitly. To specify a particular collation, use the optional `COLLATE` clause:

  ```
  SET NAMES 'charset_name' COLLATE 'collation_name'
  ```

- `SET CHARACTER SET charset_name`

  `SET CHARACTER SET` is similar to `SET NAMES` but sets `character_set_connection` and `collation_connection` to `character_set_database` and `collation_database`. A `SET CHARACTER SET charset_name` statement is equivalent to these three statements:

```
SET character_set_client = charset_name;
SET character_set_results = charset_name;
SET collation_connection = @@collation_database;
```

Setting collation_connection also implicitly sets character_set_connection to the character set associated with the collation (equivalent to executing SET character_set_connection = @@character_set_database). It is unnecessary to set character_set_connection explicitly.

Note

ucs2, utf16, utf16le, and utf32 cannot be used as a client character set, which means that they do not work for SET NAMES or SET CHARACTER SET.

The MySQL client programs mysql, mysqladmin, mysqlcheck, mysqlimport, and mysqlshow determine the default character set to use as follows:

- In the absence of other information, the programs use the compiled-in default character set, usually latin1.

- The programs can autodetect which character set to use based on the operating system setting, such as the value of the LANG or LC_ALL locale environment variable on Unix systems or the code page setting on Windows systems. For systems on which the locale is available from the OS, the client uses it to set the default character set rather than using the compiled-in default. For example, setting LANG to ru_RU.KOI8-R causes the koi8r character set to be used. Thus, users can configure the locale in their environment for use by MySQL clients.

  The OS character set is mapped to the closest MySQL character set if there is no exact match. If the client does not support the matching character set, it uses the compiled-in default. For example, ucs2 is not supported as a connection character set.

  C applications can use character set autodetection based on the OS setting by invoking mysql_options() as follows before connecting to the server:

```
mysql_options(mysql,
              MYSQL_SET_CHARSET_NAME,
              MYSQL_AUTODETECT_CHARSET_NAME);
```

- The programs support a --default-character-set option, which enables users to specify the character set explicitly to override whatever default the client otherwise determines.

When a client connects to the server, it sends the name of the character set that it wants to use. The server uses the name to set the character_set_client, character_set_results, and character_set_connection system variables. In effect, the server performs a SET NAMES operation using the character set name.

With the mysql client, to use a character set different from the default, you could explicitly execute SET NAMES every time you start up. To accomplish the same result more easily, add the --default-character-set option setting to your mysql command line or in your option file. For example, the following option file setting changes the three connection-related character set variables set to koi8r each time you invoke mysql:

```
[mysql]
default-character-set=koi8r
```

If you are using the mysql client with auto-reconnect enabled (which is not recommended), it is preferable to use the charset command rather than SET NAMES. For example:

```
mysql> charset utf8
Charset changed
```

The charset command issues a SET NAMES statement, and also changes the default character set that mysql uses when it reconnects after the connection has dropped.

Example: Suppose that column1 is defined as CHAR(5) CHARACTER SET latin2. If you do not say SET NAMES or SET CHARACTER SET, then for SELECT column1 FROM t, the server sends back all the values for column1 using the character set that the client specified when it connected. On the other hand, if you say SET NAMES 'latin1' or SET CHARACTER SET latin1 before issuing the SELECT statement, the server converts the latin2 values to latin1 just before sending results back. Conversion may be lossy if there are characters that are not in both character sets.

If you want the server to perform no conversion of result sets or error messages, set character_set_results to NULL or binary:

```
SET character_set_results = NULL;
```

To see the values of the character set and collation system variables that apply to your connection, use these statements:

```
SHOW VARIABLES LIKE 'character_set%';
SHOW VARIABLES LIKE 'collation%';
```

You must also consider the environment within which your MySQL applications execute. See Section 10.1.5, “Configuring the Character Set and Collation for Applications”.

For more information about character sets and error messages, see Section 10.1.6, “Character Set for Error Messages”.

10.1.5 Configuring the Character Set and Collation for Applications

For applications that store data using the default MySQL character set and collation (latin1, latin1_swedish_ci), no special configuration should be needed. If applications require data storage using a different character set or collation, you can configure character set information several ways:

- Specify character settings per database. For example, applications that use one database might require utf8, whereas applications that use another database might require sjis.
- Specify character settings at server startup. This causes the server to use the given settings for all applications that do not make other arrangements.
- Specify character settings at configuration time, if you build MySQL from source. This causes the server to use the given settings for all applications, without having to specify them at server startup.

When different applications require different character settings, the per-database technique provides a good deal of flexibility. If most or all applications use the same character set, specifying character settings at server startup or configuration time may be most convenient.

For the per-database or server-startup techniques, the settings control the character set for data storage. Applications must also tell the server which character set to use for client/server communications, as described in the following instructions.

The examples shown here assume use of the utf8 character set and utf8_general_ci collation.

Specify character settings per database. To create a database such that its tables will use a given default character set and collation for data storage, use a CREATE DATABASE statement like this:

```
CREATE DATABASE mydb
  DEFAULT CHARACTER SET utf8
  DEFAULT COLLATE utf8_general_ci;
```

Tables created in the database will use utf8 and utf8_general_ci by default for any character columns.

Applications that use the database should also configure their connection to the server each time they connect. This can be done by executing a SET NAMES 'utf8' statement after connecting. The statement can be used regardless of connection method: The mysql client, PHP scripts, and so forth.

In some cases, it may be possible to configure the connection to use the desired character set some other way. For example, for connections made using mysql, you can specify the --default-character-set=utf8 command-line option to achieve the same effect as SET NAMES 'utf8'.

For more information about configuring client connections, see Section 10.1.4, “Connection Character Sets and Collations”.

If you change the default character set or collation for a database, stored routines that use the database defaults must be dropped and recreated so that they use the new defaults. (In a stored routine, variables with character data types use the database defaults if the character set or collation are not specified explicitly. See Section 13.1.12, “CREATE PROCEDURE and CREATE FUNCTION Syntax”.)

Specify character settings at server startup. To select a character set and collation at server startup, use the --character-set-server and --collation-server options. For example, to specify the options in an option file, include these lines:

```
[mysqld]
character-set-server=utf8
collation-server=utf8_general_ci
```

These settings apply server-wide and apply as the defaults for databases created by any application, and for tables created in those databases.

It is still necessary for applications to configure their connection using SET NAMES or equivalent after they connect, as described previously. You might be tempted to start the server with the --init_connect="SET NAMES 'utf8'" option to cause SET NAMES to be executed automatically for each client that connects. However, this will yield inconsistent results because the init_connect value is not executed for users who have the SUPER privilege.

Specify character settings at MySQL configuration time. To select a character set and collation when you configure and build MySQL from source, use the DEFAULT_CHARSET and DEFAULT_COLLATION options for CMake:

```
shell> cmake . -DDEFAULT_CHARSET=utf8 \
           -DDEFAULT_COLLATION=utf8_general_ci
```

The resulting server uses utf8 and utf8_general_ci as the default for databases and tables and for client connections. It is unnecessary to use --character-set-server and --collation-server to specify those defaults at server startup. It is also unnecessary for applications to configure their connection using SET NAMES or equivalent after they connect to the server.

Regardless of how you configure the MySQL character set for application use, you must also consider the environment within which those applications execute. If you will send statements using UTF-8 text taken from a file that you create in an editor, you should edit the file with the locale of your environment set to UTF-8 so that the file encoding is correct and so that the operating system handles it correctly. If you use the `mysql` client from within a terminal window, the window must be configured to use UTF-8 or characters may not display properly. For a script that executes in a Web environment, the script must handle character encoding properly for its interaction with the MySQL server, and it must generate pages that correctly indicate the encoding so that browsers know how to display the content of the pages. For example, you can include this `<meta>` tag within your `<head>` element:

```
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
```

10.1.6 Character Set for Error Messages

This section describes how the server uses character sets for constructing error messages and returning them to clients. For information about the language of error messages (rather than the character set), see Section 10.2, “Setting the Error Message Language”.

In MySQL 5.7, the server constructs error messages using UTF-8 and returns them to clients in the character set specified by the `character_set_results` system variable.

The server constructs error messages as follows:

- The message template uses UTF-8.
- Parameters in the message template are replaced with values that apply to a specific error occurrence:
  - Identifiers such as table or column names use UTF-8 internally so they are copied as is.
  - Character (nonbinary) string values are converted from their character set to UTF-8.
  - Binary string values are copied as is for bytes in the range `0x20` to `0x7E`, and using `\x` hex encoding for bytes outside that range. For example, if a duplicate-key error occurs for an attempt to insert `0x41CF9F` into a `VARBINARY` unique column, the resulting error message uses UTF-8 with some bytes hex encoded:

    ```
    Duplicate entry 'A\xC3\x9F' for key 1
    ```

To return a message to the client after it has been constructed, the server converts it from UTF-8 to the character set specified by the `character_set_results` system variable. If `character_set_results` has a value of `NULL` or `binary`, no conversion occurs. No conversion occurs if the variable value is `utf8`, either, because that matches the original error message character set.

For characters that cannot be represented in `character_set_results`, some encoding may occur during the conversion. The encoding uses Unicode code point values:

- Characters in the Basic Multilingual Plane (BMP) range (`0x0000` to `0xFFFF`) are written using `\`*`nnnn`* notation.
- Characters outside the BMP range (`0x01000` to `0x10FFFF`) are written using `\+`*`nnnnnn`* notation.

Clients can set `character_set_results` to control the character set in which they receive error messages. The variable can be set directly, or indirectly by means such as `SET NAMES`. For more information about `character_set_results`, see Section 10.1.4, “Connection Character Sets and Collations”.

The encoding that occurs during the conversion to `character_set_results` before returning error messages to clients can result in different message content compared to earlier versions (before MySQL 5.5). For example, if an error occurs for an attempt to drop a table named ペ (KATAKANA LETTER PE) and `character_set_results` is a character set such as `latin1` that does not contain that character, the resulting message sent to the client has an encoded table name:

```
ERROR 1051 (42S02): Unknown table '\30DA'
```

Before MySQL 5.5, the name is not encoded:

```
ERROR 1051 (42S02): Unknown table 'ペ'
```

10.1.7 Collation Issues

The following sections discuss various aspects of character set collations.

10.1.7.1 Collation Names

MySQL collation names follow these rules:

- A name ending in `_ci` indicates a case-insensitive collation.
- A name ending in `_cs` indicates a case-sensitive collation.
- A name ending in `_bin` indicates a binary collation. Character comparisons are based on character binary code values.
- Unicode collation names may include a version number to indicate the version of the Unicode Collation Algorithm (UCA) on which the collation is based. UCA-based collations without a version number in the name use the version-4.0.0 UCA weight keys: http://www.unicode.org/Public/UCA/4.0.0/allkeys-4.0.0.txt. A collation name such as `utf8_unicode_520_ci` is based on UCA 5.2.0 weight keys: http://www.unicode.org/Public/UCA/5.2.0/allkeys.txt.

10.1.7.2 Using `COLLATE` in SQL Statements

With the `COLLATE` clause, you can override whatever the default collation is for a comparison. `COLLATE` may be used in various parts of SQL statements. Here are some examples:

- With `ORDER BY`:

```
SELECT k
FROM t1
ORDER BY k COLLATE latin1_german2_ci;
```

- With `AS`:

```
SELECT k COLLATE latin1_german2_ci AS k1
FROM t1
ORDER BY k1;
```

- With `GROUP BY`:

```
SELECT k
FROM t1
GROUP BY k COLLATE latin1_german2_ci;
```

- With aggregate functions:

```
SELECT MAX(k COLLATE latin1_german2_ci)
FROM t1;
```

- With DISTINCT:

```
SELECT DISTINCT k COLLATE latin1_german2_ci
FROM t1;
```

- With WHERE:

```
     SELECT *
     FROM t1
     WHERE _latin1 'Müller' COLLATE latin1_german2_ci = k;
```

```
     SELECT *
     FROM t1
     WHERE k LIKE _latin1 'Müller' COLLATE latin1_german2_ci;
```

- With HAVING:

```
SELECT k
FROM t1
GROUP BY k
HAVING k = _latin1 'Müller' COLLATE latin1_german2_ci;
```

10.1.7.3 COLLATE Clause Precedence

The COLLATE clause has high precedence (higher than ||), so the following two expressions are equivalent:

```
x || y COLLATE z
x || (y COLLATE z)
```

10.1.7.4 Collations Must Be for the Right Character Set

Each character set has one or more collations, but each collation is associated with one and only one character set. Therefore, the following statement causes an error message because the latin2_bin collation is not legal with the latin1 character set:

```
mysql> SELECT _latin1 'x' COLLATE latin2_bin;
ERROR 1253 (42000): COLLATION 'latin2_bin' is not valid
for CHARACTER SET 'latin1'
```

10.1.7.5 Collation of Expressions

In the great majority of statements, it is obvious what collation MySQL uses to resolve a comparison operation. For example, in the following cases, it should be clear that the collation is the collation of column charset_name:

```
SELECT x FROM T ORDER BY x;
SELECT x FROM T WHERE x = x;
SELECT DISTINCT x FROM T;
```

However, with multiple operands, there can be ambiguity. For example:

```
SELECT x FROM T WHERE x = 'Y';
```

Should the comparison use the collation of the column x, or of the string literal 'Y'? Both x and 'Y' have collations, so which collation takes precedence?

Standard SQL resolves such questions using what used to be called “coercibility” rules. MySQL assigns coercibility values as follows:

- An explicit COLLATE clause has a coercibility of 0. (Not coercible at all.)
- The concatenation of two strings with different collations has a coercibility of 1.
- The collation of a column or a stored routine parameter or local variable has a coercibility of 2.
- A “system constant” (the string returned by functions such as USER() or VERSION()) has a coercibility of 3.
- The collation of a literal has a coercibility of 4.
- NULL or an expression that is derived from NULL has a coercibility of 5.

MySQL uses coercibility values with the following rules to resolve ambiguities:

- Use the collation with the lowest coercibility value.
- If both sides have the same coercibility, then:
  - If both sides are Unicode, or both sides are not Unicode, it is an error.
  - If one of the sides has a Unicode character set, and another side has a non-Unicode character set, the side with Unicode character set wins, and automatic character set conversion is applied to the non-Unicode side. For example, the following statement does not return an error:

    ```
    SELECT CONCAT(utf8_column, latin1_column) FROM t1;
    ```

    It returns a result that has a character set of utf8 and the same collation as utf8_column. Values of latin1_column are automatically converted to utf8 before concatenating.
  - For an operation with operands from the same character set but that mix a _bin collation and a _ci or _cs collation, the _bin collation is used. This is similar to how operations that mix nonbinary and binary strings evaluate the operands as binary strings, except that it is for collations rather than data types.

Although automatic conversion is not in the SQL standard, the SQL standard document does say that every character set is (in terms of supported characters) a “subset” of Unicode. Because it is a well-known principle that “what applies to a superset can apply to a subset,” we believe that a collation for Unicode can apply for comparisons with non-Unicode strings.

Examples:

Comparison	Collation Used
column1 = 'A'	Use collation of column1
column1 = 'A' COLLATE x	Use collation of 'A' COLLATE x

Comparison	Collation Used
column1 COLLATE x = 'A' COLLATE y	Error

The COERCIBILITY() function can be used to determine the coercibility of a string expression:

```
mysql> SELECT COERCIBILITY('A' COLLATE latin1_swedish_ci);
        -> 0
mysql> SELECT COERCIBILITY(VERSION());
        -> 3
mysql> SELECT COERCIBILITY('A');
        -> 4
```

See Section 12.14, “Information Functions”.

For implicit conversion of a numeric or temporal value to a string, such as occurs for the argument 1 in the expression CONCAT(1, 'abc'), the result is a character (nonbinary) string that has a character set and collation determined by the character_set_connection and collation_connection system variables. See Section 12.2, “Type Conversion in Expression Evaluation”.

10.1.7.6 The _bin and binary Collations

This section describes how _bin collations for nonbinary strings differ from the binary “collation” for binary strings.

Nonbinary strings (as stored in the CHAR, VARCHAR, and TEXT data types) have a character set and collation. A given character set can have several collations, each of which defines a particular sorting and comparison order for the characters in the set. One of these is the binary collation for the character set, indicated by a _bin suffix in the collation name. For example, latin1 and utf8 have binary collations named latin1_bin and utf8_bin.

Binary strings (as stored in the BINARY, VARBINARY, and BLOB data types) have no character set or collation in the sense that nonbinary strings do. (Applied to a binary string, the CHARSET() and COLLATION() functions both return a value of binary.) Binary strings are sequences of bytes and the numeric values of those bytes determine sort order.

The _bin collations differ from the binary collation in several respects.

The unit for sorting and comparison. Binary strings are sequences of bytes. Sorting and comparison is always based on numeric byte values. Nonbinary strings are sequences of characters, which might be multi-byte. Collations for nonbinary strings define an ordering of the character values for sorting and comparison. For the _bin collation, this ordering is based solely on binary code values of the characters (which is similar to ordering for binary strings except that a _bin collation must take into account that a character might contain multiple bytes). For other collations, character ordering might take additional factors such as lettercase into account.

Character set conversion. A nonbinary string has a character set and is converted to another character set in many cases, even when the string has a _bin collation:

- When assigning column values from another column that has a different character set:

```
UPDATE t1 SET utf8_bin_column=latin1_column;
INSERT INTO t1 (latin1_column) SELECT utf8_bin_column FROM t2;
```

- When assigning column values for INSERT or UPDATE using a string literal:

```
SET NAMES latin1;
INSERT INTO t1 (utf8_bin_column) VALUES ('string-in-latin1');
```

- When sending results from the server to a client:

```
SET NAMES latin1;
SELECT utf8_bin_column FROM t2;
```

For binary string columns, no conversion occurs. For the preceding cases, the string value is copied byte-wise.

Lettercase conversion. Collations provide information about lettercase of characters, so characters in a nonbinary string can be converted from one lettercase to another, even for `_bin` collations that ignore lettercase for ordering:

```
mysql> SET NAMES latin1 COLLATE latin1_bin;
Query OK, 0 rows affected (0.02 sec)

mysql> SELECT LOWER('aA'), UPPER('zZ');
+-------------+-------------+
| LOWER('aA') | UPPER('zZ') |
+-------------+-------------+
| aa          | ZZ          |
+-------------+-------------+
1 row in set (0.13 sec)
```

The concept of lettercase does not apply to bytes in a binary string. To perform lettercase conversion, the string must be converted to a nonbinary string:

```
mysql> SET NAMES binary;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT LOWER('aA'), LOWER(CONVERT('aA' USING latin1));
+-------------+-----------------------------------+
| LOWER('aA') | LOWER(CONVERT('aA' USING latin1)) |
+-------------+-----------------------------------+
| aA          | aa                                |
+-------------+-----------------------------------+
1 row in set (0.00 sec)
```

Trailing space handling in comparisons. Nonbinary strings have `PADSPACE` behavior for all collations, including `_bin` collations. Trailing spaces are insignificant in comparisons:

```
mysql> SET NAMES utf8 COLLATE utf8_bin;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT 'a ' = 'a';
+------------+
| 'a ' = 'a' |
+------------+
|          1 |
+------------+
1 row in set (0.00 sec)
```

For binary strings, all characters are significant in comparisons, including trailing spaces:

```
mysql> SET NAMES binary;
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> SELECT 'a ' = 'a';
+------------+
| 'a ' = 'a' |
+------------+
|          0 |
+------------+
1 row in set (0.00 sec)
```

Trailing space handling for inserts and retrievals. `CHAR(N)` columns store nonbinary strings. Values shorter than *N* characters are extended with spaces on insertion. For retrieval, trailing spaces are removed.

`BINARY(N)` columns store binary strings. Values shorter than *N* bytes are extended with `0x00` bytes on insertion. For retrieval, nothing is removed; a value of the declared length is always returned.

```
mysql> CREATE TABLE t1 (
    ->   a CHAR(10) CHARACTER SET utf8 COLLATE utf8_bin,
    ->   b BINARY(10)
    -> );
Query OK, 0 rows affected (0.09 sec)

mysql> INSERT INTO t1 VALUES ('a','a');
Query OK, 1 row affected (0.01 sec)

mysql> SELECT HEX(a), HEX(b) FROM t1;
+--------+----------------------+
| HEX(a) | HEX(b)               |
+--------+----------------------+
| 61     | 61000000000000000000 |
+--------+----------------------+
1 row in set (0.04 sec)
```

10.1.7.7 The BINARY Operator

The `BINARY` operator casts the string following it to a binary string. This is an easy way to force a comparison to be done byte by byte rather than character by character. `BINARY` also causes trailing spaces to be significant.

```
mysql> SELECT 'a' = 'A';
        -> 1
mysql> SELECT BINARY 'a' = 'A';
        -> 0
mysql> SELECT 'a' = 'a ';
        -> 1
mysql> SELECT BINARY 'a' = 'a ';
        -> 0
```

`BINARY str` is shorthand for `CAST(str AS BINARY)`.

The `BINARY` attribute in character column definitions has a different effect. A character column defined with the `BINARY` attribute is assigned the binary collation of the column character set. Every character set has a binary collation. For example, the binary collation for the `latin1` character set is `latin1_bin`, so if the table default character set is `latin1`, these two column definitions are equivalent:

```
CHAR(10) BINARY
CHAR(10) CHARACTER SET latin1 COLLATE latin1_bin
```

The effect of `BINARY` as a column attribute differs from its effect prior to MySQL 4.1. Formerly, `BINARY` resulted in a column that was treated as a binary string. A binary string is a string of bytes that has no character set or collation, which differs from a nonbinary character string that has a binary collation. For both types of strings, comparisons are based on the numeric values of the string unit, but for nonbinary strings the unit is the character and some character sets support multi-byte characters. Section 11.4.2, "The BINARY and VARBINARY Types".

The use of CHARACTER SET binary in the definition of a CHAR, VARCHAR, or TEXT column causes the column to be treated as a binary data type. For example, the following pairs of definitions are equivalent:

```
CHAR(10) CHARACTER SET binary
BINARY(10)

VARCHAR(10) CHARACTER SET binary
VARBINARY(10)

TEXT CHARACTER SET binary
BLOB
```

10.1.7.8 Examples of the Effect of Collation

Example 1: Sorting German Umlauts

Suppose that column X in table T has these latin1 column values:

```
Muffler
Müller
MX Systems
MySQL
```

Suppose also that the column values are retrieved using the following statement:

```
SELECT X FROM T ORDER BY X COLLATE collation_name;
```

The following table shows the resulting order of the values if we use ORDER BY with different collations.

latin1_swedish_ci	latin1_german1_ci	latin1_german2_ci
Muffler	Muffler	Müller
MX Systems	Müller	Muffler
Müller	MX Systems	MX Systems
MySQL	MySQL	MySQL

The character that causes the different sort orders in this example is the U with two dots over it (ü), which the Germans call "U-umlaut."

- The first column shows the result of the SELECT using the Swedish/Finnish collating rule, which says that U-umlaut sorts with Y.
- The second column shows the result of the SELECT using the German DIN-1 rule, which says that U-umlaut sorts with U.
- The third column shows the result of the SELECT using the German DIN-2 rule, which says that U-umlaut sorts with UE.

Example 2: Searching for German Umlauts

Suppose that you have three tables that differ only by the character set and collation used:

```
mysql> SET NAMES utf8;
mysql> CREATE TABLE german1 (
    ->   c CHAR(10)
    -> ) CHARACTER SET latin1 COLLATE latin1_german1_ci;
mysql> CREATE TABLE german2 (
    ->   c CHAR(10)
    -> ) CHARACTER SET latin1 COLLATE latin1_german2_ci;
mysql> CREATE TABLE germanutf8 (
    ->   c CHAR(10)
    -> ) CHARACTER SET utf8 COLLATE utf8_unicode_ci;
```

Each table contains two records:

```
mysql> INSERT INTO german1 VALUES ('Bar'), ('Bär');
mysql> INSERT INTO german2 VALUES ('Bar'), ('Bär');
mysql> INSERT INTO germanutf8 VALUES ('Bar'), ('Bär');
```

Two of the above collations have an `A = Ä` equality, and one has no such equality (`latin1_german2_ci`). For that reason, you'll get these results in comparisons:

```
mysql> SELECT * FROM german1 WHERE c = 'Bär';
+------+
| c    |
+------+
| Bar  |
| Bär  |
+------+
mysql> SELECT * FROM german2 WHERE c = 'Bär';
+------+
| c    |
+------+
| Bär  |
+------+
mysql> SELECT * FROM germanutf8 WHERE c = 'Bär';
+------+
| c    |
+------+
| Bar  |
| Bär  |
+------+
```

This is not a bug but rather a consequence of the sorting properties of `latin1_german1_ci` and `utf8_unicode_ci` (the sorting shown is done according to the German DIN 5007 standard).

10.1.7.9 Collation and `INFORMATION_SCHEMA` Searches

String columns in `INFORMATION_SCHEMA` tables have a collation of `utf8_general_ci`, which is case insensitive. However, searches in `INFORMATION_SCHEMA` string columns are also affected by file system case sensitivity. For values that correspond to objects that are represented in the file system, such as names of databases and tables, searches may be case sensitive if the file system is case sensitive. This section describes how to work around this issue if necessary; see also Bug #34921.

Suppose that a query searches the `SCHEMATA.SCHEMA_NAME` column for the `test` database. On Linux, file systems are case sensitive, so comparisons of `SCHEMATA.SCHEMA_NAME` with `'test'` match, but comparisons with `'TEST'` do not:

```
mysql> SELECT SCHEMA_NAME FROM INFORMATION_SCHEMA.SCHEMATA
    -> WHERE SCHEMA_NAME = 'test';
+-------------+
| SCHEMA_NAME |
+-------------+
| test        |
+-------------+
1 row in set (0.01 sec)

mysql> SELECT SCHEMA_NAME FROM INFORMATION_SCHEMA.SCHEMATA
    -> WHERE SCHEMA_NAME = 'TEST';
Empty set (0.00 sec)
```

On Windows or Mac OS X where file systems are not case sensitive, comparisons match both `'test'` and `'TEST'`:

```
mysql> SELECT SCHEMA_NAME FROM INFORMATION_SCHEMA.SCHEMATA
    -> WHERE SCHEMA_NAME = 'test';
+-------------+
| SCHEMA_NAME |
+-------------+
| test        |
+-------------+
1 row in set (0.00 sec)

mysql> SELECT SCHEMA_NAME FROM INFORMATION_SCHEMA.SCHEMATA
    -> WHERE SCHEMA_NAME = 'TEST';
+-------------+
| SCHEMA_NAME |
+-------------+
| TEST        |
+-------------+
1 row in set (0.00 sec)
```

The value of the `lower_case_table_names` system variable makes no difference in this context.

This behavior occurs because the `utf8_general_ci` collation is not used for `INFORMATION_SCHEMA` queries when searching the file system for database objects. It is a result of optimizations implemented for `INFORMATION_SCHEMA` searches in MySQL. For information about these optimizations, see Section 8.2.4, “Optimizing `INFORMATION_SCHEMA` Queries”.

Searches in `INFORMATION_SCHEMA` string columns for values that refer to `INFORMATION_SCHEMA` itself do use the `utf8_general_ci` collation because `INFORMATION_SCHEMA` is a “virtual” database and is not represented in the file system. For example, comparisons with `SCHEMATA.SCHEMA_NAME` match `'information_schema'` or `'INFORMATION_SCHEMA'` regardless of platform:

```
mysql> SELECT SCHEMA_NAME FROM INFORMATION_SCHEMA.SCHEMATA
    -> WHERE SCHEMA_NAME = 'information_schema';
+--------------------+
| SCHEMA_NAME        |
+--------------------+
| information_schema |
+--------------------+
1 row in set (0.00 sec)

mysql> SELECT SCHEMA_NAME FROM INFORMATION_SCHEMA.SCHEMATA
    -> WHERE SCHEMA_NAME = 'INFORMATION_SCHEMA';
+--------------------+
| SCHEMA_NAME        |
+--------------------+
| information_schema |
+--------------------+
1 row in set (0.00 sec)
```

If the result of a string operation on an `INFORMATION_SCHEMA` column differs from expectations, a workaround is to use an explicit `COLLATE` clause to force a suitable collation (Section 10.1.7.2, “Using

COLLATE in SQL Statements”). For example, to perform a case-insensitive search, use COLLATE with the INFORMATION_SCHEMA column name:

```
mysql> SELECT SCHEMA_NAME FROM INFORMATION_SCHEMA.SCHEMATA
    -> WHERE SCHEMA_NAME COLLATE utf8_general_ci = 'test';
+-------------+
| SCHEMA_NAME |
+-------------+
| test        |
+-------------+
1 row in set (0.00 sec)

mysql> SELECT SCHEMA_NAME FROM INFORMATION_SCHEMA.SCHEMATA
    -> WHERE SCHEMA_NAME COLLATE utf8_general_ci = 'TEST';
| SCHEMA_NAME |
+-------------+
| test        |
+-------------+
1 row in set (0.00 sec)
```

You can also use the UPPER() or LOWER() function:

```
WHERE UPPER(SCHEMA_NAME) = 'TEST'
WHERE LOWER(SCHEMA_NAME) = 'test'
```

Although a case-insensitive comparison can be performed even on platforms with case-sensitive file systems, as just shown, it is not necessarily always the right thing to do. On such platforms, it is possible to have multiple objects with names that differ only in lettercase. For example, tables named city, CITY, and City can all exist simultaneously. Consider whether a search should match all such names or just one and write queries accordingly:

```
WHERE TABLE_NAME COLLATE utf8_bin = 'City'
WHERE TABLE_NAME COLLATE utf8_general_ci = 'city'
WHERE UPPER(TABLE_NAME) = 'CITY'
WHERE LOWER(TABLE_NAME) = 'city'
```

The first of those comparisons (with utf8_bin) is case sensitive; the others are not.

10.1.8 String Repertoire

The *repertoire* of a character set is the collection of characters in the set.

String expressions have a repertoire attribute, which can have two values:

- ASCII: The expression can contain only characters in the Unicode range U+0000 to U+007F.
- UNICODE: The expression can contain characters in the Unicode range U+0000 to U+FFFF.

The ASCII range is a subset of UNICODE range, so a string with ASCII repertoire can be converted safely without loss of information to the character set of any string with UNICODE repertoire or to a character set that is a superset of ASCII. (All MySQL character sets are supersets of ASCII with the exception of swe7, which reuses some punctuation characters for Swedish accented characters.) The use of repertoire enables character set conversion in expressions for many cases where MySQL would otherwise return an “illegal mix of collations” error.

The following discussion provides examples of expressions and their repertoires, and describes how the use of repertoire changes string expression evaluation:

- The repertoire for string constants depends on string content:

  ```
  SET NAMES utf8; SELECT 'abc';
  SELECT _utf8'def';
  SELECT N'MySQL';
  ```

  Although the character set is `utf8` in each of the preceding cases, the strings do not actually contain any characters outside the ASCII range, so their repertoire is `ASCII` rather than `UNICODE`.

- Columns having the `ascii` character set have `ASCII` repertoire because of their character set. In the following table, `c1` has `ASCII` repertoire:

  ```
  CREATE TABLE t1 (c1 CHAR(1) CHARACTER SET ascii);
  ```

  The following example illustrates how repertoire enables a result to be determined in a case where an error occurs without repertoire:

  ```
  CREATE TABLE t1 (
    c1 CHAR(1) CHARACTER SET latin1,
    c2 CHAR(1) CHARACTER SET ascii
  );
  INSERT INTO t1 VALUES ('a','b');
  SELECT CONCAT(c1,c2) FROM t1;
  ```

  Without repertoire, this error occurs:

  ```
  ERROR 1267 (HY000): Illegal mix of collations (latin1_swedish_ci,IMPLICIT)
  and (ascii_general_ci,IMPLICIT) for operation 'concat'
  ```

  Using repertoire, subset to superset (`ascii` to `latin1`) conversion can occur and a result is returned:

  ```
  +---------------+
  | CONCAT(c1,c2) |
  +---------------+
  | ab            |
  +---------------+
  ```

- Functions with one string argument inherit the repertoire of their argument. The result of `UPPER(_utf8'abc')` has `ASCII` repertoire because its argument has `ASCII` repertoire.

- For functions that return a string but do not have string arguments and use `character_set_connection` as the result character set, the result repertoire is `ASCII` if `character_set_connection` is `ascii`, and `UNICODE` otherwise:

  ```
  FORMAT(numeric_column, 4);
  ```

  Use of repertoire changes how MySQL evaluates the following example:

  ```
  SET NAMES ascii;
  CREATE TABLE t1 (a INT, b VARCHAR(10) CHARACTER SET latin1);
  INSERT INTO t1 VALUES (1,'b');
  SELECT CONCAT(FORMAT(a, 4), b) FROM t1;
  ```

  Without repertoire, this error occurs:

```
ERROR 1267 (HY000): Illegal mix of collations (ascii_general_ci,COERCIBLE)
and (latin1_swedish_ci,IMPLICIT) for operation 'concat'
```

With repertoire, a result is returned:

```
+-------------------------+
| CONCAT(FORMAT(a, 4), b) |
+-------------------------+
| 1.0000b                 |
+-------------------------+
```

- Functions with two or more string arguments use the “widest” argument repertoire for the result repertoire (`UNICODE` is wider than `ASCII`). Consider the following `CONCAT()` calls:

  ```
  CONCAT(_ucs2 0x0041, _ucs2 0x0042)
  CONCAT(_ucs2 0x0041, _ucs2 0x00C2)
  ```

  For the first call, the repertoire is `ASCII` because both arguments are within the range of the `ascii` character set. For the second call, the repertoire is `UNICODE` because the second argument is outside the `ascii` character set range.

- The repertoire for function return values is determined based only on the repertoire of the arguments that affect the result's character set and collation.

  ```
  IF(column1 < column2, 'smaller', 'greater')
  ```

  The result repertoire is `ASCII` because the two string arguments (the second argument and the third argument) both have `ASCII` repertoire. The first argument does not matter for the result repertoire, even if the expression uses string values.

10.1.9 Operations Affected by Character Set Support

This section describes operations that take character set information into account.

10.1.9.1 Result Strings

MySQL has many operators and functions that return a string. This section answers the question: What is the character set and collation of such a string?

For simple functions that take string input and return a string result as output, the output's character set and collation are the same as those of the principal input value. For example, `UPPER(X)` returns a string whose character string and collation are the same as that of *X*. The same applies for `INSTR()`, `LCASE()`, `LOWER()`, `LTRIM()`, `MID()`, `REPEAT()`, `REPLACE()`, `REVERSE()`, `RIGHT()`, `RPAD()`, `RTRIM()`, `SOUNDEX()`, `SUBSTRING()`, `TRIM()`, `UCASE()`, and `UPPER()`.

Note: The `REPLACE()` function, unlike all other functions, always ignores the collation of the string input and performs a case-sensitive comparison.

If a string input or function result is a binary string, the string has no character set or collation. This can be checked by using the `CHARSET()` and `COLLATION()` functions, both of which return `binary` to indicate that their argument is a binary string:

```
mysql> SELECT CHARSET(BINARY 'a'), COLLATION(BINARY 'a');
+---------------------+-----------------------+
| CHARSET(BINARY 'a') | COLLATION(BINARY 'a') |
```

```
+----------------------+-----------------------+
| binary               | binary                |
+----------------------+-----------------------+
```

For operations that combine multiple string inputs and return a single string output, the “aggregation rules” of standard SQL apply for determining the collation of the result:

- If an explicit `COLLATE X` occurs, use *X*.
- If explicit `COLLATE X` and `COLLATE Y` occur, raise an error.
- Otherwise, if all collations are *X*, use *X*.
- Otherwise, the result has no collation.

For example, with `CASE ... WHEN a THEN b WHEN b THEN c COLLATE X END`, the resulting collation is *X*. The same applies for `UNION`, `||`, `CONCAT()`, `ELT()`, `GREATEST()`, `IF()`, and `LEAST()`.

For operations that convert to character data, the character set and collation of the strings that result from the operations are defined by the `character_set_connection` and `collation_connection` system variables. This applies only to `CAST()`, `CONV()`, `FORMAT()`, `HEX()`, and `SPACE()`.

If you are uncertain about the character set or collation of the result returned by a string function, you can use the `CHARSET()` or `COLLATION()` function to find out:

```
mysql> SELECT USER(), CHARSET(USER()), COLLATION(USER());
+----------------+-----------------+-------------------+
| USER()         | CHARSET(USER()) | COLLATION(USER()) |
+----------------+-----------------+-------------------+
| test@localhost | utf8            | utf8_general_ci   |
+----------------+-----------------+-------------------+
```

10.1.9.2 `CONVERT()` and `CAST()`

`CONVERT()` provides a way to convert data between different character sets. The syntax is:

```
CONVERT(expr USING transcoding_name)
```

In MySQL, transcoding names are the same as the corresponding character set names.

Examples:

```
SELECT CONVERT(_latin1'Müller' USING utf8);
INSERT INTO utf8table (utf8column)
    SELECT CONVERT(latin1field USING utf8) FROM latin1table;
```

`CONVERT(... USING ...)` is implemented according to the standard SQL specification.

You may also use `CAST()` to convert a string to a different character set. The syntax is:

```
CAST(character_string AS character_data_type CHARACTER SET charset_name)
```

Example:

```
SELECT CAST(_latin1'test' AS CHAR CHARACTER SET utf8);
```

If you use CAST() without specifying CHARACTER SET, the resulting character set and collation are defined by the character_set_connection and collation_connection system variables. If you use CAST() with CHARACTER SET X, the resulting character set and collation are X and the default collation of X.

You may not use a COLLATE clause inside a CONVERT() or CAST() call, but you may use it outside. For example, CAST(... COLLATE ...) is illegal, but CAST(...) COLLATE ... is legal:

```
SELECT CAST(_latin1'test' AS CHAR CHARACTER SET utf8) COLLATE utf8_bin;
```

10.1.9.3 SHOW Statements and INFORMATION_SCHEMA

Several SHOW statements provide additional character set information. These include SHOW CHARACTER SET, SHOW COLLATION, SHOW CREATE DATABASE, SHOW CREATE TABLE and SHOW COLUMNS. These statements are described here briefly. For more information, see Section 13.7.5, “SHOW Syntax”.

INFORMATION_SCHEMA has several tables that contain information similar to that displayed by the SHOW statements. For example, the CHARACTER_SETS and COLLATIONS tables contain the information displayed by SHOW CHARACTER SET and SHOW COLLATION. See Chapter 19, *INFORMATION_SCHEMA Tables*.

The SHOW CHARACTER SET statement shows all available character sets. It takes an optional LIKE clause that indicates which character set names to match. For example:

```
mysql> SHOW CHARACTER SET LIKE 'latin%';
+---------+-----------------------------+-------------------+--------+
| Charset | Description                 | Default collation | Maxlen |
+---------+-----------------------------+-------------------+--------+
| latin1  | cp1252 West European        | latin1_swedish_ci |      1 |
| latin2  | ISO 8859-2 Central European | latin2_general_ci |      1 |
| latin5  | ISO 8859-9 Turkish          | latin5_turkish_ci |      1 |
| latin7  | ISO 8859-13 Baltic          | latin7_general_ci |      1 |
+---------+-----------------------------+-------------------+--------+
```

The output from SHOW COLLATION includes all available character sets. It takes an optional LIKE clause that indicates which collation names to match. For example:

```
mysql> SHOW COLLATION LIKE 'latin1%';
+-------------------+---------+----+---------+----------+---------+
| Collation         | Charset | Id | Default | Compiled | Sortlen |
+-------------------+---------+----+---------+----------+---------+
| latin1_german1_ci | latin1  |  5 |         |          |       0 |
| latin1_swedish_ci | latin1  |  8 | Yes     | Yes      |       0 |
| latin1_danish_ci  | latin1  | 15 |         |          |       0 |
| latin1_german2_ci | latin1  | 31 |         | Yes      |       2 |
| latin1_bin        | latin1  | 47 |         | Yes      |       0 |
| latin1_general_ci | latin1  | 48 |         |          |       0 |
| latin1_general_cs | latin1  | 49 |         |          |       0 |
| latin1_spanish_ci | latin1  | 94 |         |          |       0 |
+-------------------+---------+----+---------+----------+---------+
```

SHOW CREATE DATABASE displays the CREATE DATABASE statement that creates a given database:

```
mysql> SHOW CREATE DATABASE test;
+----------+-----------------------------------------------------------------+
| Database | Create Database                                                 |
+----------+-----------------------------------------------------------------+
| test     | CREATE DATABASE `test` /*!40100 DEFAULT CHARACTER SET latin1 */ |
+----------+-----------------------------------------------------------------+
```

If no COLLATE clause is shown, the default collation for the character set applies.

SHOW CREATE TABLE is similar, but displays the CREATE TABLE statement to create a given table. The column definitions indicate any character set specifications, and the table options include character set information.

The SHOW COLUMNS statement displays the collations of a table's columns when invoked as SHOW FULL COLUMNS. Columns with CHAR, VARCHAR, or TEXT data types have collations. Numeric and other noncharacter types have no collation (indicated by NULL as the Collation value). For example:

```
mysql> SHOW FULL COLUMNS FROM person\G
*************************** 1. row ***************************
     Field: id
      Type: smallint(5) unsigned
 Collation: NULL
      Null: NO
       Key: PRI
   Default: NULL
     Extra: auto_increment
Privileges: select,insert,update,references
   Comment:
*************************** 2. row ***************************
     Field: name
      Type: char(60)
 Collation: latin1_swedish_ci
      Null: NO
       Key:
   Default:
     Extra:
Privileges: select,insert,update,references
   Comment:
```

The character set is not part of the display but is implied by the collation name.

10.1.10 Unicode Support

The initial implementation of Unicode support (in MySQL 4.1) included two character sets for storing Unicode data:

- ucs2, the UCS-2 encoding of the Unicode character set using 16 bits per character.
- utf8, a UTF-8 encoding of the Unicode character set using one to three bytes per character.

These two character sets support the characters from the Basic Multilingual Plane (BMP) of Unicode Version 3.0. BMP characters have these characteristics:

- Their code values are between 0 and 65535 (or U+0000 .. U+FFFF).
- They can be encoded with a fixed 16-bit word, as in ucs2.
- They can be encoded with 8, 16, or 24 bits, as in utf8.
- They are sufficient for almost all characters in major languages.

Characters not supported by the aforementioned character sets include supplementary characters that lie outside the BMP. Characters outside the BMP compare as REPLACEMENT CHARACTER and convert to '?' when converted to a Unicode character set.

In MySQL 5.7, Unicode support includes supplementary characters, which requires new character sets that have a broader range and therefore take more space. The following table shows a brief feature comparison of previous and current Unicode support.

Before MySQL 5.5	MySQL 5.5 and up
All Unicode 3.0 characters	All Unicode 5.0 and 6.0 characters
No supplementary characters	With supplementary characters
ucs2 character set, BMP only	No change
utf8 character set for up to three bytes, BMP only	No change
	New utf8mb4 character set for up to four bytes, BMP or supplemental
	New utf16 character set, BMP or supplemental
	New utf16le character set, BMP or supplemental
	New utf32 character set, BMP or supplemental

These changes are upward compatible. If you want to use the new character sets, there are potential incompatibility issues for your applications; see Section 10.1.11, “Upgrading from Previous to Current Unicode Support”. That section also describes how to convert tables from utf8 to the (4-byte) utf8mb4 character set, and what constraints may apply in doing so.

MySQL 5.7 supports these Unicode character sets:

- ucs2, the UCS-2 encoding of the Unicode character set using 16 bits per character.
- utf16, the UTF-16 encoding for the Unicode character set; like ucs2 but with an extension for supplementary characters.
- utf16le, the UTF-16LE encoding for the Unicode character set; like utf16 but little-endian rather than big-endian.
- utf32, the UTF-32 encoding for the Unicode character set using 32 bits per character.
- utf8, a UTF-8 encoding of the Unicode character set using one to three bytes per character.
- utf8mb4, a UTF-8 encoding of the Unicode character set using one to four bytes per character.

ucs2 and utf8 support BMP characters. utf8mb4, utf16, utf16le, and utf32 support BMP and supplementary characters.

A similar set of collations is available for most Unicode character sets. For example, each has a Danish collation, the names of which are ucs2_danish_ci, utf16_danish_ci, utf32_danish_ci, utf8_danish_ci, and utf8mb4_danish_ci. The exception is utf16le, which has only two collations. All Unicode collations are listed at Section 10.1.14.1, “Unicode Character Sets”, which also describes collation properties for supplementary characters.

Note that although many of the supplementary characters come from East Asian languages, what MySQL 5.7 adds is support for more Japanese and Chinese characters in Unicode character sets, not support for new Japanese and Chinese character sets.

The MySQL implementation of UCS-2, UTF-16, and UTF-32 stores characters in big-endian byte order and does not use a byte order mark (BOM) at the beginning of values. Other database systems might use little-endian byte order or a BOM. In such cases, conversion of values will need to be performed when transferring data between those systems and MySQL. The implementation of UTF-16LE is little-endian.

MySQL uses no BOM for UTF-8 values.

Client applications that need to communicate with the server using Unicode should set the client character set accordingly; for example, by issuing a SET NAMES 'utf8' statement. ucs2, utf16, utf16le, and utf32 cannot be used as a client character set, which means that they do not work for SET NAMES or SET CHARACTER SET. (See Section 10.1.4, “Connection Character Sets and Collations”.)

The following sections provide additional detail on the Unicode character sets in MySQL.

10.1.10.1 The ucs2 Character Set (UCS-2 Unicode Encoding)

In UCS-2, every character is represented by a 2-byte Unicode code with the most significant byte first. For example: LATIN CAPITAL LETTER A has the code 0x0041 and it is stored as a 2-byte sequence: 0x00 0x41. CYRILLIC SMALL LETTER YERU (Unicode 0x044B) is stored as a 2-byte sequence: 0x04 0x4B. For Unicode characters and their codes, please refer to the Unicode Home Page.

In MySQL, the ucs2 character set is a fixed-length 16-bit encoding for Unicode BMP characters.

10.1.10.2 The utf16 Character Set (UTF-16 Unicode Encoding)

The utf16 character set is the ucs2 character set with an extension that enables encoding of supplementary characters:

- For a BMP character, utf16 and ucs2 have identical storage characteristics: same code values, same encoding, same length.
- For a supplementary character, utf16 has a special sequence for representing the character using 32 bits. This is called the “surrogate” mechanism: For a number greater than 0xffff, take 10 bits and add them to 0xd800 and put them in the first 16-bit word, take 10 more bits and add them to 0xdc00 and put them in the next 16-bit word. Consequently, all supplementary characters require 32 bits, where the first 16 bits are a number between 0xd800 and 0xdbff, and the last 16 bits are a number between 0xdc00 and 0xdfff. Examples are in Section 15.5 Surrogates Area of the Unicode 4.0 document.

Because utf16 supports surrogates and ucs2 does not, there is a validity check that applies only in utf16: You cannot insert a top surrogate without a bottom surrogate, or vice versa. For example:

```
INSERT INTO t (ucs2_column) VALUES (0xd800); /* legal */
INSERT INTO t (utf16_column)VALUES (0xd800); /* illegal */
```

There is no validity check for characters that are technically valid but are not true Unicode (that is, characters that Unicode considers to be “unassigned code points” or “private use” characters or even “illegals” like 0xffff). For example, since U+F8FF is the Apple Logo, this is legal:

```
INSERT INTO t (utf16_column)VALUES (0xf8ff); /* legal */
```

Such characters cannot be expected to mean the same thing to everyone.

Because MySQL must allow for the worst case (that one character requires four bytes) the maximum length of a utf16 column or index is only half of the maximum length for a ucs2 column or index. For example, in MySQL 5.7, the maximum length of a MEMORY table index key is 3072 bytes, so these statements create tables with the longest permitted indexes for ucs2 and utf16 columns:

```
CREATE TABLE tf (s1 VARCHAR(1536) CHARACTER SET ucs2) ENGINE=MEMORY;
CREATE INDEX i ON tf (s1);
CREATE TABLE tg (s1 VARCHAR(768) CHARACTER SET utf16) ENGINE=MEMORY;
CREATE INDEX i ON tg (s1);
```

10.1.10.3 The `utf16le` Character Set (UTF-16LE Unicode Encoding)

This is the same as `utf16` but is little-endian rather than big-endian.

10.1.10.4 The `utf32` Character Set (UTF-32 Unicode Encoding)

The `utf32` character set is fixed length (like `ucs2` and unlike `utf16`). `utf32` uses 32 bits for every character, unlike `ucs2` (which uses 16 bits for every character), and unlike `utf16` (which uses 16 bits for some characters and 32 bits for others).

`utf32` takes twice as much space as `ucs2` and more space than `utf16`, but `utf32` has the same advantage as `ucs2` that it is predictable for storage: The required number of bytes for `utf32` equals the number of characters times 4. Also, unlike `utf16`, there are no tricks for encoding in `utf32`, so the stored value equals the code value.

To demonstrate how the latter advantage is useful, here is an example that shows how to determine a `utf8mb4` value given the `utf32` code value:

```
/* Assume code value = 100cc LINEAR B WHEELED CHARIOT */
CREATE TABLE tmp (utf32_col CHAR(1) CHARACTER SET utf32,
                  utf8mb4_col CHAR(1) CHARACTER SET utf8mb4);
INSERT INTO tmp VALUES (0x000100cc,NULL);
UPDATE tmp SET utf8mb4_col = utf32_col;
SELECT HEX(utf32_col),HEX(utf8mb4_col) FROM tmp;
```

MySQL is very forgiving about additions of unassigned Unicode characters or private-use-area characters. There is in fact only one validity check for `utf32`: No code value may be greater than `0x10ffff`. For example, this is illegal:

```
INSERT INTO t (utf32_column) VALUES (0x110000); /* illegal */
```

10.1.10.5 The `utf8` Character Set (3-Byte UTF-8 Unicode Encoding)

UTF-8 (Unicode Transformation Format with 8-bit units) is an alternative way to store Unicode data. It is implemented according to RFC 3629, which describes encoding sequences that take from one to four bytes. (An older standard for UTF-8 encoding, RFC 2279, describes UTF-8 sequences that take from one to six bytes. RFC 3629 renders RFC 2279 obsolete; for this reason, sequences with five and six bytes are no longer used.)

The idea of UTF-8 is that various Unicode characters are encoded using byte sequences of different lengths:

- Basic Latin letters, digits, and punctuation signs use one byte.

- Most European and Middle East script letters fit into a 2-byte sequence: extended Latin letters (with tilde, macron, acute, grave and other accents), Cyrillic, Greek, Armenian, Hebrew, Arabic, Syriac, and others.

- Korean, Chinese, and Japanese ideographs use 3-byte or 4-byte sequences.

The `utf8` character set is the same in MySQL 5.7 as before 5.7 and has exactly the same characteristics:

- No support for supplementary characters (BMP characters only).

- A maximum of three bytes per multi-byte character.

Exactly the same set of characters is available in `utf8` as in `ucs2`. That is, they have the same repertoire.

Tip: To save space with UTF-8, use `VARCHAR` instead of `CHAR`. Otherwise, MySQL must reserve three bytes for each character in a `CHAR CHARACTER SET utf8` column because that is the maximum possible length. For example, MySQL must reserve 30 bytes for a `CHAR(10) CHARACTER SET utf8` column.

For additional information about data type storage, see Section 11.6, “Data Type Storage Requirements”. For information about `InnoDB` physical row storage, including how `InnoDB` tables that use `COMPACT` row format handle UTF-8 `CHAR(`*`N`*`)` columns internally, see Physical Row Structure.

10.1.10.6 The `utf8mb3` “Character Set” (Alias for `utf8`)

In a future version of MySQL, it is possible that `utf8` will become the 4-byte `utf8`, and that users who want to indicate 3-byte `utf8` will have to say `utf8mb3`. To avoid some future problems which might occur with replication when master and slave servers have different MySQL versions, it is possible for users to specify `utf8mb3` in `CHARACTER SET` clauses, and `utf8mb3_`*`collation_substring`* in `COLLATE` clauses, where *`collation_substring`* is `bin`, `czech_ci`, `danish_ci`, `esperanto_ci`, `estonian_ci`, and so forth. For example:

```
CREATE TABLE t (s1 CHAR(1) CHARACTER SET utf8mb3;
SELECT * FROM t WHERE s1 COLLATE utf8mb3_general_ci = 'x';
DECLARE x VARCHAR(5) CHARACTER SET utf8mb3 COLLATE utf8mb3_danish_ci;
SELECT CAST('a' AS CHAR CHARACTER SET utf8) COLLATE utf8_czech_ci;
```

MySQL immediately converts instances of `utf8mb3` in an alias to `utf8`, so in statements such as `SHOW CREATE TABLE` or `SELECT CHARACTER_SET_NAME FROM INFORMATION_SCHEMA.COLUMNS` or `SELECT COLLATION_NAME FROM INFORMATION_SCHEMA.COLUMNS`, users will see the true name, `utf8` or `utf8_`*`collation_substring`*.

The `utf8mb3` alias is valid only in `CHARACTER SET` clauses, and in certain other places. For example, these are legal:

```
mysqld --character-set-server=utf8mb3
SET NAMES 'utf8mb3'; /* and other SET statements that have similar effect */
SELECT _utf8mb3 'a';
```

There is no `utf8mb3` alias to the corresponding `utf8` collation for collation names that include a version number (for example, *`utf8_unicode_520_ci`*) to indicate the Unicode Collation Algorithm version on which the collation is based.

10.1.10.7 The `utf8mb4` Character Set (4-Byte UTF-8 Unicode Encoding)

The character set named `utf8` uses a maximum of three bytes per character and contains only BMP characters. The `utf8mb4` character set uses a maximum of four bytes per character supports supplemental characters:

- For a BMP character, `utf8` and `utf8mb4` have identical storage characteristics: same code values, same encoding, same length.

- For a supplementary character, `utf8` cannot store the character at all, while `utf8mb4` requires four bytes to store it. Since `utf8` cannot store the character at all, you do not have any supplementary characters in utf8 columns and you need not worry about converting characters or losing data when upgrading utf8 data from older versions of MySQL.

utf8mb4 is a superset of utf8, so for an operation such as the following concatenation, the result has character set utf8mb4 and the collation of utf8mb4_col:

```
SELECT CONCAT(utf8_col, utf8mb4_col);
```

Similarly, the following comparison in the WHERE clause works according to the collation of utf8mb_col:

```
SELECT * FROM utf8_tbl, utf8mb4_tbl
WHERE utf8_tbl.utf8_col = utf8mb4_tbl.utf8mb4_col;
```

Tip: To save space with UTF-8, use VARCHAR instead of CHAR. Otherwise, MySQL must reserve three (or four) bytes for each character in a CHAR CHARACTER SET utf8 (or utf8mb4) column because that is the maximum possible length. For example, MySQL must reserve 40 bytes for a CHAR(10) CHARACTER SET utf8mb4 column.

10.1.11 Upgrading from Previous to Current Unicode Support

This section describes issues pertaining to Unicode support that you may face when upgrading to MySQL 5.7 from an older MySQL release. It also provides guidelines for downgrading from MySQL 5.7 to an older release.

In most respects, upgrading to MySQL 5.7 should present few problems with regard to Unicode usage, although there are some potential areas of incompatibility. These are the primary areas of concern:

- For the variable-length character data types (VARCHAR and the TEXT types), the maximum length in characters is less for utf8mb4 columns than for utf8 columns.
- For all character data types (CHAR, VARCHAR, and the TEXT types), the maximum number of characters that can be indexed is less for utf8mb4 columns than for utf8 columns.

Consequently, if you want to upgrade tables from utf8 to utf8mb4 to take advantage of supplementary-character support, it may be necessary to change some column or index definitions.

Tables can be converted from utf8 to utf8mb4 by using ALTER TABLE. Suppose that a table was originally defined as follows:

```
CREATE TABLE t1 (
  col1 CHAR(10) CHARACTER SET utf8 COLLATE utf8_unicode_ci NOT NULL,
  col2 CHAR(10) CHARACTER SET utf8 COLLATE utf8_bin NOT NULL
) CHARACTER SET utf8;
```

The following statement converts t1 to use utf8mb4:

```
ALTER TABLE t1
  DEFAULT CHARACTER SET utf8mb4,
  MODIFY col1 CHAR(10)
    CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci NOT NULL,
  MODIFY col2 CHAR(10)
    CHARACTER SET utf8mb4 COLLATE utf8mb4_bin NOT NULL;
```

In terms of table content, conversion from utf8 to utf8mb4 presents no problems:

- For a BMP character, utf8 and utf8mb4 have identical storage characteristics: same code values, same encoding, same length.

- For a supplementary character, utf8 cannot store the character at all, while utf8mb4 requires four bytes to store it. Since utf8 cannot store the character at all, you do not have any supplementary characters in utf8 columns and you need not worry about converting characters or losing data when upgrading utf8 data from older versions of MySQL.

In terms of table structure, the catch when converting from utf8 to utf8mb4 is that the maximum length of a column or index key is unchanged in terms of *bytes*. Therefore, it is smaller in terms of *characters* because the maximum length of a character is four bytes instead of three. For the CHAR, VARCHAR, and TEXT data types, watch for these things when converting your MySQL tables:

- Check all definitions of utf8 columns and make sure they will not exceed the maximum length for the storage engine.

- Check all indexes on utf8 columns and make sure they will not exceed the maximum length for the storage engine. Sometimes the maximum can change due to storage engine enhancements.

If the preceding conditions apply, you must either reduce the defined length of columns or indexes, or continue to use utf8 rather than utf8mb4.

Here are some examples where structural changes may be needed:

- A TINYTEXT column can hold up to 255 bytes, so it can hold up to 85 3-byte or 63 4-byte characters. Suppose that you have a TINYTEXT column that uses utf8 but must be able to contain more than 63 characters. You cannot convert it to utf8mb4 unless you also change the data type to a longer type such as TEXT.

  Similarly, a very long VARCHAR column may need to be changed to one of the longer TEXT types if you want to convert it from utf8 to utf8mb4.

- InnoDB has a maximum index length of 767 bytes, so for utf8 or utf8mb4 columns, you can index a maximum of 255 or 191 characters, respectively. If you currently have utf8 columns with indexes longer than 191 characters, you will need to index a smaller number of characters. In an InnoDB table, these column and index definitions are legal:

  ```
  col1 VARCHAR(500) CHARACTER SET utf8, INDEX (col1(255))
  ```

  To use utf8mb4 instead, the index must be smaller:

  ```
  col1 VARCHAR(500) CHARACTER SET utf8mb4, INDEX (col1(191))
  ```

The preceding types of changes are most likely to be required only if you have very long columns or indexes. Otherwise, you should be able to convert your tables from utf8 to utf8mb4 without problems. You can do this by using ALTER TABLE as described earlier in this section after upgrading in place to 5.7.

The following items summarize other potential areas of incompatibility:

- Performance of 4-byte UTF-8 (utf8mb4) is slower than for 3-byte UTF-8 (utf8). If you do not want to incur this penalty, continue to use utf8.

- SET NAMES 'utf8mb4' causes use of the 4-byte character set for connection character sets. As long as no 4-byte characters are sent from the server, there should be no problems. Otherwise, applications that expect to receive a maximum of three bytes per character may have problems. Conversely, applications that expect to send 4-byte characters must ensure that the server understands them.

- Applications cannot send utf16, utf16le, or utf32 character data to an older server that does not understand them.
- For replication, if the character sets that support supplementary characters are going to be used on the master, all slaves must understand them as well. If you attempt to replicate from a MySQL 5.7 master to an older slave, utf8 data will be seen as utf8 by the slave and should replicate correctly. But you cannot send utf8mb4, utf16, utf16le, or utf32 data.

  Also, keep in mind the general principle that if a table has different definitions on the master and slave, this can lead to unexpected results. For example, the differences in limitations on index key length makes it risky to use utf8 on the master and utf8mb4 on the slave.

If you have upgraded to MySQL 5.7, and then decide to downgrade back to an older release, these considerations apply:

- ucs2 and utf8 data should present no problems.
- Any definitions that refer to the utf8mb4, utf16, utf16le, or utf32 character sets will not be recognized by the older server.
- For object definitions that refer to the utf8mb4 character set, you can dump them with mysqldump in MySQL 5.7, edit the dump file to change instances of utf8mb4 to utf8, and reload the file in the older server, as long as there are no 4-byte characters in the data. The older server will see utf8 in the dump file object definitions and create new objects that use the (3-byte) utf8 character set.

10.1.12 UTF-8 for Metadata

Metadata is "the data about the data." Anything that *describes* the database—as opposed to being the *contents* of the database—is metadata. Thus column names, database names, user names, version names, and most of the string results from SHOW are metadata. This is also true of the contents of tables in INFORMATION_SCHEMA because those tables by definition contain information about database objects.

Representation of metadata must satisfy these requirements:

- All metadata must be in the same character set. Otherwise, neither the SHOW statements nor SELECT statements for tables in INFORMATION_SCHEMA would work properly because different rows in the same column of the results of these operations would be in different character sets.
- Metadata must include all characters in all languages. Otherwise, users would not be able to name columns and tables using their own languages.

To satisfy both requirements, MySQL stores metadata in a Unicode character set, namely UTF-8. This does not cause any disruption if you never use accented or non-Latin characters. But if you do, you should be aware that metadata is in UTF-8.

The metadata requirements mean that the return values of the USER(), CURRENT_USER(), SESSION_USER(), SYSTEM_USER(), DATABASE(), and VERSION() functions have the UTF-8 character set by default.

The server sets the character_set_system system variable to the name of the metadata character set:

```
mysql> SHOW VARIABLES LIKE 'character_set_system';
+----------------------+-------+
| Variable_name        | Value |
+----------------------+-------+
| character_set_system | utf8  |
+----------------------+-------+
```

Storage of metadata using Unicode does *not* mean that the server returns headers of columns and the results of DESCRIBE functions in the character_set_system character set by default. When you use SELECT column1 FROM t, the name column1 itself is returned from the server to the client in the character set determined by the value of the character_set_results system variable, which has a default value of latin1. If you want the server to pass metadata results back in a different character set, use the SET NAMES statement to force the server to perform character set conversion. SET NAMES sets the character_set_results and other related system variables. (See Section 10.1.4, “Connection Character Sets and Collations”.) Alternatively, a client program can perform the conversion after receiving the result from the server. It is more efficient for the client to perform the conversion, but this option is not always available for all clients.

If character_set_results is set to NULL, no conversion is performed and the server returns metadata using its original character set (the set indicated by character_set_system).

Error messages returned from the server to the client are converted to the client character set automatically, as with metadata.

If you are using (for example) the USER() function for comparison or assignment within a single statement, don't worry. MySQL performs some automatic conversion for you.

```
SELECT * FROM t1 WHERE USER() = latin1_column;
```

This works because the contents of latin1_column are automatically converted to UTF-8 before the comparison.

```
INSERT INTO t1 (latin1_column) SELECT USER();
```

This works because the contents of USER() are automatically converted to latin1 before the assignment.

Although automatic conversion is not in the SQL standard, the SQL standard document does say that every character set is (in terms of supported characters) a “subset” of Unicode. Because it is a well-known principle that “what applies to a superset can apply to a subset,” we believe that a collation for Unicode can apply for comparisons with non-Unicode strings. For more information about coercion of strings, see Section 10.1.7.5, “Collation of Expressions”.

10.1.13 Column Character Set Conversion

To convert a binary or nonbinary string column to use a particular character set, use ALTER TABLE. For successful conversion to occur, one of the following conditions must apply:

- If the column has a binary data type (BINARY, VARBINARY, BLOB), all the values that it contains must be encoded using a single character set (the character set you're converting the column to). If you use a binary column to store information in multiple character sets, MySQL has no way to know which values use which character set and cannot convert the data properly.

- If the column has a nonbinary data type (CHAR, VARCHAR, TEXT), its contents should be encoded in the column character set, not some other character set. If the contents are encoded in a different character set, you can convert the column to use a binary data type first, and then to a nonbinary column with the desired character set.

Suppose that a table t has a binary column named col1 defined as VARBINARY(50). Assuming that the information in the column is encoded using a single character set, you can convert it to a nonbinary column that has that character set. For example, if col1 contains binary data representing characters in the greek character set, you can convert it as follows:

```
ALTER TABLE t MODIFY col1 VARCHAR(50) CHARACTER SET greek;
```

If your original column has a type of BINARY(50), you could convert it to CHAR(50), but the resulting values will be padded with 0x00 bytes at the end, which may be undesirable. To remove these bytes, use the TRIM() function:

```
UPDATE t SET col1 = TRIM(TRAILING 0x00 FROM col1);
```

Suppose that table t has a nonbinary column named col1 defined as CHAR(50) CHARACTER SET latin1 but you want to convert it to use utf8 so that you can store values from many languages. The following statement accomplishes this:

```
ALTER TABLE t MODIFY col1 CHAR(50) CHARACTER SET utf8;
```

Conversion may be lossy if the column contains characters that are not in both character sets.

A special case occurs if you have old tables from before MySQL 4.1 where a nonbinary column contains values that actually are encoded in a character set different from the server's default character set. For example, an application might have stored sjis values in a column, even though MySQL's default character set was latin1. It is possible to convert the column to use the proper character set but an additional step is required. Suppose that the server's default character set was latin1 and col1 is defined as CHAR(50) but its contents are sjis values. The first step is to convert the column to a binary data type, which removes the existing character set information without performing any character conversion:

```
ALTER TABLE t MODIFY col1 BLOB;
```

The next step is to convert the column to a nonbinary data type with the proper character set:

```
ALTER TABLE t MODIFY col1 CHAR(50) CHARACTER SET sjis;
```

This procedure requires that the table not have been modified already with statements such as INSERT or UPDATE after an upgrade to MySQL 4.1 or later. In that case, MySQL would store new values in the column using latin1, and the column will contain a mix of sjis and latin1 values and cannot be converted properly.

If you specified attributes when creating a column initially, you should also specify them when altering the table with ALTER TABLE. For example, if you specified NOT NULL and an explicit DEFAULT value, you should also provide them in the ALTER TABLE statement. Otherwise, the resulting column definition will not include those attributes.

To convert all character columns in a table, the ALTER TABLE ... CONVERT TO CHARACTER SET *charset* statement may be useful. See Section 13.1.6, “ALTER TABLE Syntax”.

10.1.14 Character Sets and Collations That MySQL Supports

MySQL supports 70+ collations for 30+ character sets. This section indicates which character sets MySQL supports. There is one subsection for each group of related character sets. For each character set, the permissible collations are listed.

You can always list the available character sets and their default collations with the SHOW CHARACTER SET statement:

```
mysql> SHOW CHARACTER SET;
+----------+-----------------------------+---------------------+
| Charset  | Description                 | Default collation   |
+----------+-----------------------------+---------------------+
| big5     | Big5 Traditional Chinese    | big5_chinese_ci     |
| dec8     | DEC West European           | dec8_swedish_ci     |
| cp850    | DOS West European           | cp850_general_ci    |
| hp8      | HP West European            | hp8_english_ci      |
| koi8r    | KOI8-R Relcom Russian       | koi8r_general_ci    |
| latin1   | cp1252 West European        | latin1_swedish_ci   |
| latin2   | ISO 8859-2 Central European | latin2_general_ci   |
| swe7     | 7bit Swedish                | swe7_swedish_ci     |
| ascii    | US ASCII                    | ascii_general_ci    |
| ujis     | EUC-JP Japanese             | ujis_japanese_ci    |
| sjis     | Shift-JIS Japanese          | sjis_japanese_ci    |
| hebrew   | ISO 8859-8 Hebrew           | hebrew_general_ci   |
| tis620   | TIS620 Thai                 | tis620_thai_ci      |
| euckr    | EUC-KR Korean               | euckr_korean_ci     |
| koi8u    | KOI8-U Ukrainian            | koi8u_general_ci    |
| gb2312   | GB2312 Simplified Chinese   | gb2312_chinese_ci   |
| greek    | ISO 8859-7 Greek            | greek_general_ci    |
| cp1250   | Windows Central European    | cp1250_general_ci   |
| gbk      | GBK Simplified Chinese      | gbk_chinese_ci      |
| latin5   | ISO 8859-9 Turkish          | latin5_turkish_ci   |
| armscii8 | ARMSCII-8 Armenian          | armscii8_general_ci |
| utf8     | UTF-8 Unicode               | utf8_general_ci     |
| ucs2     | UCS-2 Unicode               | ucs2_general_ci     |
| cp866    | DOS Russian                 | cp866_general_ci    |
| keybcs2  | DOS Kamenicky Czech-Slovak  | keybcs2_general_ci  |
| macce    | Mac Central European        | macce_general_ci    |
| macroman | Mac West European           | macroman_general_ci |
| cp852    | DOS Central European        | cp852_general_ci    |
| latin7   | ISO 8859-13 Baltic          | latin7_general_ci   |
| utf8mb4  | UTF-8 Unicode               | utf8mb4_general_ci  |
| cp1251   | Windows Cyrillic            | cp1251_general_ci   |
| utf16    | UTF-16 Unicode              | utf16_general_ci    |
| utf16le  | UTF-16LE Unicode            | utf16le_general_ci  |
| cp1256   | Windows Arabic              | cp1256_general_ci   |
| cp1257   | Windows Baltic              | cp1257_general_ci   |
| utf32    | UTF-32 Unicode              | utf32_general_ci    |
| binary   | Binary pseudo charset       | binary              |
| geostd8  | GEOSTD8 Georgian            | geostd8_general_ci  |
| cp932    | SJIS for Windows Japanese   | cp932_japanese_ci   |
| eucjpms  | UJIS for Windows Japanese   | eucjpms_japanese_ci |
+----------+-----------------------------+---------------------+
```

In cases where a character set has multiple collations, it might not be clear which collation is most suitable for a given application. To avoid choosing the wrong collation, it can be helpful to perform some comparisons with representative data values to make sure that a given collation sorts values the way you expect.

Collation-Charts.Org is a useful site for information that shows how one collation compares to another.

10.1.14.1 Unicode Character Sets

MySQL 5.7 supports these Unicode character sets:

- ucs2, the UCS-2 encoding of the Unicode character set using 16 bits per character.
- utf16, the UTF-16 encoding for the Unicode character set; like ucs2 but with an extension for supplementary characters.

- utf16le, the UTF-16LE encoding for the Unicode character set; like utf16 but little-endian rather than big-endian.
- utf32, the UTF-32 encoding for the Unicode character set using 32 bits per character.
- utf8, a UTF-8 encoding of the Unicode character set using one to three bytes per character.
- utf8mb4, a UTF-8 encoding of the Unicode character set using one to four bytes per character.

ucs2 and utf8 support Basic Multilingual Plane (BMP) characters. utf8mb4, utf16, utf16le, and utf32 support BMP and supplementary characters.

You can store text in about 650 languages using these character sets. This section lists the collations available for each Unicode character set and describes their differentiating properties. For general information about the character sets, see Section 10.1.10, “Unicode Support”.

A similar set of collations is available for most Unicode character sets. These are shown in the following list, where *xxx* represents the character set name. For example, *xxx*_danish_ci represents the Danish collations, the specific names of which are ucs2_danish_ci, utf16_danish_ci, utf32_danish_ci, utf8_danish_ci, and utf8mb4_danish_ci.

Collation support for utf16le is more limited. The only collations available are utf16le_general_ci and utf16le_bin. These are similar to utf16_general_ci and utf16_bin.

Unicode collation names may also include a version number (for example, *xxx*_unicode_520_ci) to indicate the Unicode Collation Algorithm version on which the collation is based, as described later in this section. For such collations, there is no utf8mb3 alias to the corresponding utf8 collation. See Section 10.1.10.6, “The utf8mb3 “Character Set” (Alias for utf8)”.

- *xxx*_bin
- *xxx*_croatian_ci
- *xxx*_czech_ci
- *xxx*_danish_ci
- *xxx*_esperanto_ci
- *xxx*_estonian_ci
- *xxx*_general_ci (default)
- *xxx*_german2_ci
- *xxx*_general_mysql500_ci
- *xxx*_hungarian_ci
- *xxx*_icelandic_ci
- *xxx*_latvian_ci
- *xxx*_lithuanian_ci
- *xxx*_persian_ci
- *xxx*_polish_ci

- *xxx*_roman_ci
- *xxx*_romanian_ci
- *xxx*_sinhala_ci
- *xxx*_slovak_ci
- *xxx*_slovenian_ci
- *xxx*_spanish_ci
- *xxx*_spanish2_ci
- *xxx*_swedish_ci
- *xxx*_turkish_ci
- *xxx*_unicode_ci
- *xxx*_vietnamese_ci

MySQL implements the *xxx*_unicode_ci collations according to the Unicode Collation Algorithm (UCA) described at http://www.unicode.org/reports/tr10/. The collation uses the version-4.0.0 UCA weight keys: http://www.unicode.org/Public/UCA/4.0.0/allkeys-4.0.0.txt. Currently, the *xxx*_unicode_ci collations have only partial support for the Unicode Collation Algorithm. Some characters are not supported yet. Also, combining marks are not fully supported. This affects primarily Vietnamese, Yoruba, and some smaller languages such as Navajo. A combined character will be considered different from the same character written with a single unicode character in string comparisons, and the two characters are considered to have a different length (for example, as returned by the CHAR_LENGTH() function or in result set metadata).

MySQL implements language-specific Unicode collations only if the ordering with *xxx*_unicode_ci does not work well for a language. Language-specific collations are UCA-based. They are derived from *xxx*_unicode_ci with additional language tailoring rules.

Collations based on UCA versions later than 4.0.0 include the version in the collation name. Thus, *xxx*_unicode_520_ci collations are based on UCA 5.2.0 weight keys: http://www.unicode.org/Public/UCA/5.2.0/allkeys.txt.

LOWER() and UPPER() perform case folding according to the collation of their argument. A character that has uppercase and lowercase versions only in a Unicode version more recent than 4.0.0 will be converted by these functions only if the argument has a collation that uses a recent enough UCA version.

For any Unicode character set, operations performed using the *xxx*_general_ci collation are faster than those for the *xxx*_unicode_ci collation. For example, comparisons for the utf8_general_ci collation are faster, but slightly less correct, than comparisons for utf8_unicode_ci. The reason for this is that utf8_unicode_ci supports mappings such as expansions; that is, when one character compares as equal to combinations of other characters. For example, in German and some other languages “ß” is equal to “ss”. utf8_unicode_ci also supports contractions and ignorable characters. utf8_general_ci is a legacy collation that does not support expansions, contractions, or ignorable characters. It can make only one-to-one comparisons between characters.

To further illustrate, the following equalities hold in both utf8_general_ci and utf8_unicode_ci (for the effect this has in comparisons or when doing searches, see Section 10.1.7.8, “Examples of the Effect of Collation”):

```
Ä = A
Ö = O
Ü = U
```

A difference between the collations is that this is true for utf8_general_ci:

```
ß = s
```

Whereas this is true for utf8_unicode_ci, which supports the German DIN-1 ordering (also known as dictionary order):

```
ß = ss
```

MySQL implements language-specific collations for the utf8 character set only if the ordering with utf8_unicode_ci does not work well for a language. For example, utf8_unicode_ci works fine for German dictionary order and French, so there is no need to create special utf8 collations.

utf8_general_ci also is satisfactory for both German and French, except that "ß" is equal to "s", and not to "ss". If this is acceptable for your application, you should use utf8_general_ci because it is faster. If this is not acceptable (for example, if you require German dictionary order), use utf8_unicode_ci because it is more accurate.

If you require German DIN-2 (phone book) ordering, use the utf8_german2_ci collation, which compares the following sets of characters equal:

```
Ä = Æ = AE
Ö = Œ = OE
Ü = UE
ß = ss
```

utf8_german2_ci is similar to latin1_german2_ci, but the latter does not compare "Æ" equal to "AE" or "Œ" equal to "OE". There is no utf8_german_ci corresponding to latin1_german_ci for German dictionary order because utf8_general_ci suffices.

*xxx*_swedish_ci includes Swedish rules. For example, in Swedish, the following relationship holds, which is not something expected by a German or French speaker:

```
Ü = Y < Ö
```

The *xxx*_spanish_ci and *xxx*_spanish2_ci collations correspond to modern Spanish and traditional Spanish, respectively. In both collations, "ñ" (n-tilde) is a separate letter between "n" and "o". In addition, for traditional Spanish, "ch" is a separate letter between "c" and "d", and "ll" is a separate letter between "l" and "m"

The *xxx*_spanish2_ci collations may also be used for Asturian and Galician.

The *xxx*_danich_ci collations may also be used for Norwegian.

In the *xxx*_roman_ci collations, I and J compare as equal, and U and V compare as equal.

The *xxx*_croatian_ci collations are tailored for these Croatian letters: Č, Ć, Dž, Đ, Lj, Nj, Š, Ž.

For all Unicode collations except the "binary" (*xxx*_bin) collations, MySQL performs a table lookup to find a character's collating weight. This weight can be displayed using the WEIGHT_STRING() function.

(See Section 12.5, “String Functions”.) If a character is not in the table (for example, because it is a “new” character), collating weight determination becomes more complex:

- For BMP characters in general collations (*xxx*_general_ci), weight = code point.
- For BMP characters in UCA collations (for example, *xxx*_unicode_ci and language-specific collations), the following algorithm applies:

  ```
  if (code >= 0x3400 && code <= 0x4DB5)
    base= 0xFB80; /* CJK Ideograph Extension */
  else if (code >= 0x4E00 && code <= 0x9FA5)
    base= 0xFB40; /* CJK Ideograph */
  else
    base= 0xFBC0; /* All other characters */
  aaaa= base +  (code >> 15);
  bbbb= (code & 0x7FFF) | 0x8000;
  ```

  The result is a sequence of two collating elements, aaaa followed by bbbb. For example:

  ```
  mysql> SELECT HEX(WEIGHT_STRING(_ucs2 0x04CF COLLATE ucs2_unicode_ci));
  +----------------------------------------------------------+
  | HEX(WEIGHT_STRING(_ucs2 0x04CF COLLATE ucs2_unicode_ci)) |
  +----------------------------------------------------------+
  | FBC084CF                                                 |
  +----------------------------------------------------------+
  ```

  Thus, U+04cf CYRILLIC SMALL LETTER PALOCHKA is, with all UCA 4.0.0 collations, greater than U+04c0 CYRILLIC LETTER PALOCHKA. With UCA 5.2.0 collations, all palochkas sort together.

- For supplementary characters in general collations, the weight is the weight for 0xfffd REPLACEMENT CHARACTER. For supplementary characters in UCA 4.0.0 collations, their collating weight is 0xfffd. That is, to MySQL, all supplementary characters are equal to each other, and greater than almost all BMP characters.

  An example with Deseret characters and COUNT(DISTINCT):

  ```
  CREATE TABLE t (s1 VARCHAR(5) CHARACTER SET utf32 COLLATE utf32_unicode_ci);
  INSERT INTO t VALUES (0xfffd);   /* REPLACEMENT CHARACTER */
  INSERT INTO t VALUES (0x010412); /* DESERET CAPITAL LETTER BEE */
  INSERT INTO t VALUES (0x010413); /* DESERET CAPITAL LETTER TEE */
  SELECT COUNT(DISTINCT s1) FROM t;
  ```

  The result is 2 because in the MySQL *xxx*_unicode_ci collations, the replacement character has a weight of 0x0dc6, whereas Deseret Bee and Deseret Tee both have a weight of 0xfffd. (Were the utf32_general_ci collation used instead, the result would be 1 because all three characters have a weight of 0xfffd in that collation.)

  An example with cuneiform characters and WEIGHT_STRING():

  ```
  /*
  The four characters in the INSERT string are
  00000041  # LATIN CAPITAL LETTER A
  0001218F  # CUNEIFORM SIGN KAB
  000121A7  # CUNEIFORM SIGN KISH
  00000042  # LATIN CAPITAL LETTER B
  */
  CREATE TABLE t (s1 CHAR(4) CHARACTER SET utf32 COLLATE utf32_unicode_ci);
  INSERT INTO t VALUES (0x000000410001218f000121a700000042);
  SELECT HEX(WEIGHT_STRING(s1)) FROM t;
  ```

The result is:

```
0E33 FFFD FFFD 0E4A
```

0E33 and 0E4A are primary weights as in UCA 4.0.0. FFFD is the weight for KAB and also for KISH.

The rule that all supplementary characters are equal to each other is nonoptimal but is not expected to cause trouble. These characters are very rare, so it will be very rare that a multi-character string consists entirely of supplementary characters. In Japan, since the supplementary characters are obscure Kanji ideographs, the typical user does not care what order they are in, anyway. If you really want rows sorted by MySQL's rule and secondarily by code point value, it is easy:

```
ORDER BY s1 COLLATE utf32_unicode_ci, s1 COLLATE utf32_bin
```

- For supplementary characters based on UCA versions later than 4.0.0 (for example, *xxx*_unicode_520_ci), supplementary characters do not necessarily all have the same collation weight. Some have explicit weights from the UCA allkeys.txt file. Others have weights calculated from this algorithm:

```
aaaa= base +  (code >> 15);
bbbb= (code & 0x7FFF) | 0x8000;
```

The utf16_bin Collation

There is a difference between “ordering by the character's code value” and “ordering by the character's binary representation,” a difference that appears only with utf16_bin, because of surrogates.

Suppose that utf16_bin (the binary collation for utf16) was a binary comparison “byte by byte” rather than “character by character.” If that were so, the order of characters in utf16_bin would differ from the order in utf8_bin. For example, the following chart shows two rare characters. The first character is in the range E000-FFFF, so it is greater than a surrogate but less than a supplementary. The second character is a supplementary.

```
Code point  Character                    utf8         utf16
----------  ---------                    ----         -----
0FF9D       HALFWIDTH KATAKANA LETTER N  EF BE 9D     FF 9D
10384       UGARITIC LETTER DELTA        F0 90 8E 84  D8 00 DF 84
```

The two characters in the chart are in order by code point value because 0xff9d < 0x10384. And they are in order by utf8 value because 0xef < 0xf0. But they are not in order by utf16 value, if we use byte-by-byte comparison, because 0xff > 0xd8.

So MySQL's utf16_bin collation is not “byte by byte.” It is “by code point.” When MySQL sees a supplementary-character encoding in utf16, it converts to the character's code-point value, and then compares. Therefore, utf8_bin and utf16_bin are the same ordering. This is consistent with the SQL:2008 standard requirement for a UCS_BASIC collation: “UCS_BASIC is a collation in which the ordering is determined entirely by the Unicode scalar values of the characters in the strings being sorted. It is applicable to the UCS character repertoire. Since every character repertoire is a subset of the UCS repertoire, the UCS_BASIC collation is potentially applicable to every character set. NOTE 11: The Unicode scalar value of a character is its code point treated as an unsigned integer.”

If the character set is ucs2, comparison is byte-by-byte, but ucs2 strings should not contain surrogates, anyway.

The *xxx*_general_mysql500_ci collations preserve the pre-5.1.24 ordering of the original *xxx*_general_ci collations and permit upgrades for tables created before MySQL 5.1.24. For more information, see Section 2.10.3, “Checking Whether Tables or Indexes Must Be Rebuilt”, and Section 2.10.4, “Rebuilding or Repairing Tables or Indexes”.

For additional information about Unicode collations in MySQL, see Collation-Charts.Org (utf8).

10.1.14.2 West European Character Sets

Western European character sets cover most West European languages, such as French, Spanish, Catalan, Basque, Portuguese, Italian, Albanian, Dutch, German, Danish, Swedish, Norwegian, Finnish, Faroese, Icelandic, Irish, Scottish, and English.

- ascii (US ASCII) collations:
  - ascii_bin
  - ascii_general_ci (default)
- cp850 (DOS West European) collations:
  - cp850_bin
  - cp850_general_ci (default)
- dec8 (DEC Western European) collations:
  - dec8_bin
  - dec8_swedish_ci (default)
- hp8 (HP Western European) collations:
  - hp8_bin
  - hp8_english_ci (default)
- latin1 (cp1252 West European) collations:
  - latin1_bin
  - latin1_danish_ci
  - latin1_general_ci
  - latin1_general_cs
  - latin1_german1_ci
  - latin1_german2_ci
  - latin1_spanish_ci
  - latin1_swedish_ci (default)

  latin1 is the default character set. MySQL's latin1 is the same as the Windows cp1252 character set. This means it is the same as the official ISO 8859-1 or IANA (Internet Assigned Numbers Authority) latin1, except that IANA latin1 treats the code points between 0x80 and 0x9f as “undefined,” whereas cp1252, and therefore MySQL's latin1, assign characters for those positions. For example, 0x80 is the Euro sign. For the “undefined” entries in cp1252, MySQL translates 0x81 to Unicode 0x0081, 0x8d to 0x008d, 0x8f to 0x008f, 0x90 to 0x0090, and 0x9d to 0x009d.

  The latin1_swedish_ci collation is the default that probably is used by the majority of MySQL customers. Although it is frequently said that it is based on the Swedish/Finnish collation rules, there are Swedes and Finns who disagree with this statement.

  The latin1_german1_ci and latin1_german2_ci collations are based on the DIN-1 and DIN-2 standards, where DIN stands for *Deutsches Institut für Normung* (the German equivalent of ANSI). DIN-1 is called the “dictionary collation” and DIN-2 is called the “phone book collation.” For an example of the effect this has in comparisons or when doing searches, see Section 10.1.7.8, “Examples of the Effect of Collation”.

  - latin1_german1_ci (dictionary) rules:

    ```
    Ä = A
    Ö = O
    Ü = U
    ß = s
    ```

  - latin1_german2_ci (phone-book) rules:

    ```
    Ä = AE
    Ö = OE
    Ü = UE
    ß = ss
    ```

  In the latin1_spanish_ci collation, “ñ” (n-tilde) is a separate letter between “n” and “o”.

- macroman (Mac West European) collations:

  - macroman_bin

  - macroman_general_ci (default)

- swe7 (7bit Swedish) collations:

  - swe7_bin

  - swe7_swedish_ci (default)

For additional information about Western European collations in MySQL, see Collation-Charts.Org (ascii, cp850, dec8, hp8, latin1, macroman, swe7).

10.1.14.3 Central European Character Sets

MySQL provides some support for character sets used in the Czech Republic, Slovakia, Hungary, Romania, Slovenia, Croatia, Poland, and Serbia (Latin).

- cp1250 (Windows Central European) collations:

  - cp1250_bin

  - cp1250_croatian_ci

  - cp1250_czech_cs

  - `cp1250_general_ci` (default)
  - `cp1250_polish_ci`
- `cp852` (DOS Central European) collations:
  - `cp852_bin`
  - `cp852_general_ci` (default)
- `keybcs2` (DOS Kamenicky Czech-Slovak) collations:
  - `keybcs2_bin`
  - `keybcs2_general_ci` (default)
- `latin2` (ISO 8859-2 Central European) collations:
  - `latin2_bin`
  - `latin2_croatian_ci`
  - `latin2_czech_cs`
  - `latin2_general_ci` (default)
  - `latin2_hungarian_ci`
- `macce` (Mac Central European) collations:
  - `macce_bin`
  - `macce_general_ci` (default)

For additional information about Central European collations in MySQL, see Collation-Charts.Org (cp1250, cp852, keybcs2, latin2, macce).

10.1.14.4 South European and Middle East Character Sets

South European and Middle Eastern character sets supported by MySQL include Armenian, Arabic, Georgian, Greek, Hebrew, and Turkish.

- `armscii8` (ARMSCII-8 Armenian) collations:
  - `armscii8_bin`
  - `armscii8_general_ci` (default)
- `cp1256` (Windows Arabic) collations:
  - `cp1256_bin`
  - `cp1256_general_ci` (default)
- `geostd8` (GEOSTD8 Georgian) collations:
  - `geostd8_bin`
  - `geostd8_general_ci` (default)

- greek (ISO 8859-7 Greek) collations:
  - greek_bin
  - greek_general_ci (default)
- hebrew (ISO 8859-8 Hebrew) collations:
  - hebrew_bin
  - hebrew_general_ci (default)
- latin5 (ISO 8859-9 Turkish) collations:
  - latin5_bin
  - latin5_turkish_ci (default)

For additional information about South European and Middle Eastern collations in MySQL, see Collation-Charts.Org (armscii8, cp1256, geostd8, greek, hebrew, latin5).

10.1.14.5 Baltic Character Sets

The Baltic character sets cover Estonian, Latvian, and Lithuanian languages.

- cp1257 (Windows Baltic) collations:
  - cp1257_bin
  - cp1257_general_ci (default)
  - cp1257_lithuanian_ci
- latin7 (ISO 8859-13 Baltic) collations:
  - latin7_bin
  - latin7_estonian_cs
  - latin7_general_ci (default)
  - latin7_general_cs

For additional information about Baltic collations in MySQL, see Collation-Charts.Org (cp1257, latin7).

10.1.14.6 Cyrillic Character Sets

The Cyrillic character sets and collations are for use with Belarusian, Bulgarian, Russian, Ukrainian, and Serbian (Cyrillic) languages.

- cp1251 (Windows Cyrillic) collations:
  - cp1251_bin
  - cp1251_bulgarian_ci
  - cp1251_general_ci (default)
  - cp1251_general_cs

  - cp1251_ukrainian_ci

- cp866 (DOS Russian) collations:
  - cp866_bin
  - cp866_general_ci (default)
- koi8r (KOI8-R Relcom Russian) collations:
  - koi8r_bin
  - koi8r_general_ci (default)
- koi8u (KOI8-U Ukrainian) collations:
  - koi8u_bin
  - koi8u_general_ci (default)

For additional information about Cyrillic collations in MySQL, see Collation-Charts.Org (cp1251, cp866, koi8r, koi8u).).

10.1.14.7 Asian Character Sets

The Asian character sets that we support include Chinese, Japanese, Korean, and Thai. These can be complicated. For example, the Chinese sets must allow for thousands of different characters. See The cp932 Character Set, for additional information about the cp932 and sjis character sets.

For answers to some common questions and problems relating support for Asian character sets in MySQL, see Section B.11, "MySQL 5.7 FAQ: MySQL Chinese, Japanese, and Korean Character Sets".

- big5 (Big5 Traditional Chinese) collations:
  - big5_bin
  - big5_chinese_ci (default)
- cp932 (SJIS for Windows Japanese) collations:
  - cp932_bin
  - cp932_japanese_ci (default)
- eucjpms (UJIS for Windows Japanese) collations:
  - eucjpms_bin
  - eucjpms_japanese_ci (default)
- euckr (EUC-KR Korean) collations:
  - euckr_bin
  - euckr_korean_ci (default)
- gb2312 (GB2312 Simplified Chinese) collations:

  - `gb2312_bin`
  - `gb2312_chinese_ci` (default)

- `gbk` (GBK Simplified Chinese) collations:
  - `gbk_bin`
  - `gbk_chinese_ci` (default)
- `gb18030` (China National Standard GB18030) collations:
  - `gb18030_bin`
  - `gb18030_chinese_ci` (default)
  - `gb18030_chinese_unicode520_ci`
- `sjis` (Shift-JIS Japanese) collations:
  - `sjis_bin`
  - `sjis_japanese_ci` (default)
- `tis620` (TIS620 Thai) collations:
  - `tis620_bin`
  - `tis620_thai_ci` (default)
- `ujis` (EUC-JP Japanese) collations:
  - `ujis_bin`
  - `ujis_japanese_ci` (default)

The `big5_chinese_ci` collation sorts on number of strokes.

For additional information about Asian collations in MySQL, see Collation-Charts.Org (big5, cp932, eucjpms, euckr, gb2312, gbk, sjis, tis620, ujis).

The `cp932` Character Set

Why is `cp932` needed?

In MySQL, the `sjis` character set corresponds to the `Shift_JIS` character set defined by IANA, which supports JIS X0201 and JIS X0208 characters. (See http://www.iana.org/assignments/character-sets.)

However, the meaning of “SHIFT JIS” as a descriptive term has become very vague and it often includes the extensions to `Shift_JIS` that are defined by various vendors.

For example, “SHIFT JIS” used in Japanese Windows environments is a Microsoft extension of `Shift_JIS` and its exact name is `Microsoft Windows Codepage : 932` or `cp932`. In addition to the characters supported by `Shift_JIS`, `cp932` supports extension characters such as NEC special characters, NEC selected—IBM extended characters, and IBM selected characters.

Many Japanese users have experienced problems using these extension characters. These problems stem from the following factors:

- MySQL automatically converts character sets.
- Character sets are converted using Unicode (`ucs2`).
- The `sjis` character set does not support the conversion of these extension characters.
- There are several conversion rules from so-called “SHIFT JIS” to Unicode, and some characters are converted to Unicode differently depending on the conversion rule. MySQL supports only one of these rules (described later).

The MySQL `cp932` character set is designed to solve these problems.

Because MySQL supports character set conversion, it is important to separate IANA `Shift_JIS` and `cp932` into two different character sets because they provide different conversion rules.

How does `cp932` differ from `sjis`?

The `cp932` character set differs from `sjis` in the following ways:

- `cp932` supports NEC special characters, NEC selected—IBM extended characters, and IBM selected characters.
- Some `cp932` characters have two different code points, both of which convert to the same Unicode code point. When converting from Unicode back to `cp932`, one of the code points must be selected. For this “round trip conversion,” the rule recommended by Microsoft is used. (See http://support.microsoft.com/kb/170559/EN-US/.)

  The conversion rule works like this:

  - If the character is in both JIS X 0208 and NEC special characters, use the code point of JIS X 0208.
  - If the character is in both NEC special characters and IBM selected characters, use the code point of NEC special characters.
  - If the character is in both IBM selected characters and NEC selected—IBM extended characters, use the code point of IBM extended characters.

  The table shown at http://www.microsoft.com/globaldev/reference/dbcs/932.htm provides information about the Unicode values of `cp932` characters. For `cp932` table entries with characters under which a four-digit number appears, the number represents the corresponding Unicode (`ucs2`) encoding. For table entries with an underlined two-digit value appears, there is a range of `cp932` character values that begin with those two digits. Clicking such a table entry takes you to a page that displays the Unicode value for each of the `cp932` characters that begin with those digits.

  The following links are of special interest. They correspond to the encodings for the following sets of characters:

  - NEC special characters:

    ```
    http://www.microsoft.com/globaldev/reference/dbcs/932/932_87.htm
    ```

  - NEC selected—IBM extended characters:

    ```
    http://www.microsoft.com/globaldev/reference/dbcs/932/932_ED.htm
    http://www.microsoft.com/globaldev/reference/dbcs/932/932_EE.htm
    ```

  - IBM selected characters:

```
http://www.microsoft.com/globaldev/reference/dbcs/932/932_FA.htm
http://www.microsoft.com/globaldev/reference/dbcs/932/932_FB.htm
http://www.microsoft.com/globaldev/reference/dbcs/932/932_FC.htm
```

- cp932 supports conversion of user-defined characters in combination with eucjpms, and solves the problems with sjis/ujis conversion. For details, please refer to http://www.opengroup.or.jp/jvc/cde/sjis-euc-e.html.

For some characters, conversion to and from ucs2 is different for sjis and cp932. The following tables illustrate these differences.

Conversion to ucs2:

sjis/cp932 Value	**sjis -> ucs2 Conversion**	**cp932 -> ucs2 Conversion**
5C	005C	005C
7E	007E	007E
815C	2015	2015
815F	005C	FF3C
8160	301C	FF5E
8161	2016	2225
817C	2212	FF0D
8191	00A2	FFE0
8192	00A3	FFE1
81CA	00AC	FFE2

Conversion from ucs2:

ucs2 value	**ucs2 -> sjis Conversion**	**ucs2 -> cp932 Conversion**
005C	815F	5C
007E	7E	7E
00A2	8191	3F
00A3	8192	3F
00AC	81CA	3F
2015	815C	815C
2016	8161	3F
2212	817C	3F
2225	3F	8161
301C	8160	3F
FF0D	3F	817C
FF3C	3F	815F
FF5E	3F	8160
FFE0	3F	8191
FFE1	3F	8192
FFE2	3F	81CA

Users of any Japanese character sets should be aware that using --character-set-client-handshake (or --skip-character-set-client-handshake) has an important effect. See Section 5.1.3, “Server Command Options”.

10.2 Setting the Error Message Language

By default, mysqld produces error messages in English, but they can also be displayed in any of several other languages: Czech, Danish, Dutch, Estonian, French, German, Greek, Hungarian, Italian, Japanese, Korean, Norwegian, Norwegian-ny, Polish, Portuguese, Romanian, Russian, Slovak, Spanish, or Swedish.

You can select which language the server uses for error messages using the instructions in this section.

In MySQL 5.7, the server searches for the error message file in two locations:

- It tries to find the file in a directory constructed from two system variable values, lc_messages_dir and lc_messages, with the latter converted to a language name. Suppose that you start the server using this command:

  ```
  shell> mysqld --lc_messages_dir=/usr/share/mysql --lc_messages=fr_FR
  ```

  In this case, mysqld maps the locale fr_FR to the language french and looks for the error file in the /usr/share/mysql/french directory.

- If the message file cannot be found in the directory constructed as just described, the server ignores the lc_messages value and uses only the lc_messages_dir value as the location in which to look.

The lc_messages_dir system variable has only a global value and is read only. lc_messages has global and session values and can be modified at runtime, so the error message language can be changed while the server is running, and individual clients each can have a different error message language by changing their session lc_messages value to a different locale name. For example, if the server is using the fr_FR locale for error messages, a client can execute this statement to receive error messages in English:

```
mysql> SET lc_messages = 'en_US';
```

By default, the language files are located in the share/mysql/*LANGUAGE* directory under the MySQL base directory.

For information about changing the character set for error messages (rather than the language), see Section 10.1.6, “Character Set for Error Messages”.

You can change the content of the error messages produced by the server using the instructions in the MySQL Internals manual, available at MySQL Internals: Error Messages. If you do change the content of error messages, remember to repeat your changes after each upgrade to a newer version of MySQL.

10.3 Adding a Character Set

This section discusses the procedure for adding a character set to MySQL. The proper procedure depends on whether the character set is simple or complex:

- If the character set does not need special string collating routines for sorting and does not need multi-byte character support, it is simple.

- If the character set needs either of those features, it is complex.

For example, `greek` and `swe7` are simple character sets, whereas `big5` and `czech` are complex character sets.

To use the following instructions, you must have a MySQL source distribution. In the instructions, *`MYSET`* represents the name of the character set that you want to add.

1. Add a `<charset>` element for *`MYSET`* to the `sql/share/charsets/Index.xml` file. Use the existing contents in the file as a guide to adding new contents. A partial listing for the `latin1` `<charset>` element follows:

   ```
   <charset name="latin1">
     <family>Western</family>
     <description>cp1252 West European</description>
     ...
     <collation name="latin1_swedish_ci" id="8" order="Finnish, Swedish">
       <flag>primary</flag>
       <flag>compiled</flag>
     </collation>
     <collation name="latin1_danish_ci" id="15" order="Danish"/>
     ...
     <collation name="latin1_bin" id="47" order="Binary">
       <flag>binary</flag>
       <flag>compiled</flag>
     </collation>
     ...
   </charset>
   ```

   The `<charset>` element must list all the collations for the character set. These must include at least a binary collation and a default (primary) collation. The default collation is often named using a suffix of `general_ci` (general, case insensitive). It is possible for the binary collation to be the default collation, but usually they are different. The default collation should have a `primary` flag. The binary collation should have a `binary` flag.

   You must assign a unique ID number to each collation. The range of IDs from 1024 to 2047 is reserved for user-defined collations. To find the maximum of the currently used collation IDs, use this query:

   ```
   SELECT MAX(ID) FROM INFORMATION_SCHEMA.COLLATIONS;
   ```

2. This step depends on whether you are adding a simple or complex character set. A simple character set requires only a configuration file, whereas a complex character set requires C source file that defines collation functions, multi-byte functions, or both.

   For a simple character set, create a configuration file, *`MYSET`*`.xml`, that describes the character set properties. Create this file in the `sql/share/charsets` directory. You can use a copy of `latin1.xml` as the basis for this file. The syntax for the file is very simple:

   - Comments are written as ordinary XML comments (`<!--` *`text`* `-->`).

   - Words within `<map>` array elements are separated by arbitrary amounts of whitespace.

   - Each word within `<map>` array elements must be a number in hexadecimal format.

   - The `<map>` array element for the `<ctype>` element has 257 words. The other `<map>` array elements after that have 256 words. See Section 10.3.1, “Character Definition Arrays”.

   - For each collation listed in the `<charset>` element for the character set in `Index.xml`, *`MYSET`*`.xml` must contain a `<collation>` element that defines the character ordering.

   For a complex character set, create a C source file that describes the character set properties and defines the support routines necessary to properly perform operations on the character set:

   - Create the file `ctype-MYSET.c` in the `strings` directory. Look at one of the existing `ctype-*.c` files (such as `ctype-big5.c`) to see what needs to be defined. The arrays in your file must have names like `ctype_MYSET`, `to_lower_MYSET`, and so on. These correspond to the arrays for a simple character set. See Section 10.3.1, "Character Definition Arrays".

   - For each `<collation>` element listed in the `<charset>` element for the character set in `Index.xml`, the `ctype-MYSET.c` file must provide an implementation of the collation.

   - If the character set requires string collating functions, see Section 10.3.2, "String Collating Support for Complex Character Sets".

   - If the character set requires multi-byte character support, see Section 10.3.3, "Multi-Byte Character Support for Complex Character Sets".

3. Modify the configuration information. Use the existing configuration information as a guide to adding information for *MYSYS*. The example here assumes that the character set has default and binary collations, but more lines are needed if *MYSET* has additional collations.

   a. Edit `mysys/charset-def.c`, and "register" the collations for the new character set.

      Add these lines to the "declaration" section:

      ```
      #ifdef HAVE_CHARSET_MYSET
      extern CHARSET_INFO my_charset_MYSET_general_ci;
      extern CHARSET_INFO my_charset_MYSET_bin;
      #endif
      ```

      Add these lines to the "registration" section:

      ```
      #ifdef HAVE_CHARSET_MYSET
        add_compiled_collation(&my_charset_MYSET_general_ci);
        add_compiled_collation(&my_charset_MYSET_bin);
      #endif
      ```

   b. If the character set uses `ctype-MYSET.c`, edit `strings/CMakeLists.txt` and add `ctype-MYSET.c` to the definition of the `STRINGS_SOURCES` variable.

   c. Edit `cmake/character_sets.cmake`:

      i. Add *MYSET* to the value of with `CHARSETS_AVAILABLE` in alphabetic order.

      ii. Add *MYSET* to the value of `CHARSETS_COMPLEX` in alphabetic order. This is needed even for simple character sets, or `CMake` will not recognize `-DDEFAULT_CHARSET=MYSET`.

4. Reconfigure, recompile, and test.

10.3.1 Character Definition Arrays

Each simple character set has a configuration file located in the `sql/share/charsets` directory. For a character set named *MYSYS*, the file is named `MYSET.xml`. It uses `<map>` array elements to list character set properties. `<map>` elements appear within these elements:

- `<ctype>` defines attributes for each character.

- `<lower>` and `<upper>` list the lowercase and uppercase characters.
- `<unicode>` maps 8-bit character values to Unicode values.
- `<collation>` elements indicate character ordering for comparisons and sorts, one element per collation. Binary collations need no `<map>` element because the character codes themselves provide the ordering.

For a complex character set as implemented in a `ctype-MYSET.c` file in the `strings` directory, there are corresponding arrays: `ctype_MYSET[]`, `to_lower_MYSET[]`, and so forth. Not every complex character set has all of the arrays. See also the existing `ctype-*.c` files for examples. See the `CHARSET_INFO.txt` file in the `strings` directory for additional information.

Most of the arrays are indexed by character value and have 256 elements. The `<ctype>` array is indexed by character value + 1 and has 257 elements. This is a legacy convention for handling `EOF`.

`<ctype>` array elements are bit values. Each element describes the attributes of a single character in the character set. Each attribute is associated with a bitmask, as defined in `include/m_ctype.h`:

```
#define _MY_U   01      /* Upper case */
#define _MY_L   02      /* Lower case */
#define _MY_NMR 04      /* Numeral (digit) */
#define _MY_SPC 010     /* Spacing character */
#define _MY_PNT 020     /* Punctuation */
#define _MY_CTR 040     /* Control character */
#define _MY_B   0100    /* Blank */
#define _MY_X   0200    /* heXadecimal digit */
```

The `<ctype>` value for a given character should be the union of the applicable bitmask values that describe the character. For example, `'A'` is an uppercase character (`_MY_U`) as well as a hexadecimal digit (`_MY_X`), so its `ctype` value should be defined like this:

```
ctype['A'+1] = _MY_U | _MY_X = 01 | 0200 = 0201
```

The bitmask values in `m_ctype.h` are octal values, but the elements of the `<ctype>` array in `MYSET.xml` should be written as hexadecimal values.

The `<lower>` and `<upper>` arrays hold the lowercase and uppercase characters corresponding to each member of the character set. For example:

```
lower['A'] should contain 'a'
upper['a'] should contain 'A'
```

Each `<collation>` array indicates how characters should be ordered for comparison and sorting purposes. MySQL sorts characters based on the values of this information. In some cases, this is the same as the `<upper>` array, which means that sorting is case-insensitive. For more complicated sorting rules (for complex character sets), see the discussion of string collating in Section 10.3.2, “String Collating Support for Complex Character Sets”.

10.3.2 String Collating Support for Complex Character Sets

For a simple character set named `MYSET`, sorting rules are specified in the `MYSET.xml` configuration file using `<map>` array elements within `<collation>` elements. If the sorting rules for your language are too complex to be handled with simple arrays, you must define string collating functions in the `ctype-MYSET.c` source file in the `strings` directory.

The existing character sets provide the best documentation and examples to show how these functions are implemented. Look at the `ctype-*.c` files in the `strings` directory, such as the files for the `big5`, `czech`, `gbk`, `sjis`, and `tis160` character sets. Take a look at the `MY_COLLATION_HANDLER` structures to see how they are used. See also the `CHARSET_INFO.txt` file in the `strings` directory for additional information.

10.3.3 Multi-Byte Character Support for Complex Character Sets

If you want to add support for a new character set named *`MYSET`* that includes multi-byte characters, you must use multi-byte character functions in the `ctype-MYSET.c` source file in the `strings` directory.

The existing character sets provide the best documentation and examples to show how these functions are implemented. Look at the `ctype-*.c` files in the `strings` directory, such as the files for the `euc_kr`, `gb2312`, `gbk`, `sjis`, and `ujis` character sets. Take a look at the `MY_CHARSET_HANDLER` structures to see how they are used. See also the `CHARSET_INFO.txt` file in the `strings` directory for additional information.

10.4 Adding a Collation to a Character Set

A collation is a set of rules that defines how to compare and sort character strings. Each collation in MySQL belongs to a single character set. Every character set has at least one collation, and most have two or more collations.

A collation orders characters based on weights. Each character in a character set maps to a weight. Characters with equal weights compare as equal, and characters with unequal weights compare according to the relative magnitude of their weights.

The `WEIGHT_STRING()` function can be used to see the weights for the characters in a string. The value that it returns to indicate weights is a binary string, so it is convenient to use `HEX(WEIGHT_STRING(str))` to display the weights in printable form. The following example shows that weights do not differ for lettercase for the letters in `'AaBb'` it if is a nonbinary case-insensitive string, but do differ if it is a binary string:

```
mysql> SELECT HEX(WEIGHT_STRING('AaBb' COLLATE latin1_swedish_ci));
+------------------------------------------------------+
| HEX(WEIGHT_STRING('AaBb' COLLATE latin1_swedish_ci)) |
+------------------------------------------------------+
| 41414242                                             |
+------------------------------------------------------+
mysql> SELECT HEX(WEIGHT_STRING(BINARY 'AaBb'));
+-----------------------------------+
| HEX(WEIGHT_STRING(BINARY 'AaBb')) |
+-----------------------------------+
| 41614262                          |
+-----------------------------------+
```

MySQL supports several collation implementations, as discussed in Section 10.4.1, “Collation Implementation Types”. Some of these can be added to MySQL without recompiling:

- Simple collations for 8-bit character sets.
- UCA-based collations for Unicode character sets.
- Binary (*`xxx`*`_bin`) collations.

The following sections describe how to add collations of the first two types to existing character sets. All existing character sets already have a binary collation, so there is no need here to describe how to add one.

Summary of the procedure for adding a new collation:

1. Choose a collation ID.

2. Add configuration information that names the collation and describes the character-ordering rules.

3. Restart the server.

4. Verify that the collation is present.

The instructions here cover only collations that can be added without recompiling MySQL. To add a collation that does require recompiling (as implemented by means of functions in a C source file), use the instructions in Section 10.3, “Adding a Character Set”. However, instead of adding all the information required for a complete character set, just modify the appropriate files for an existing character set. That is, based on what is already present for the character set's current collations, add data structures, functions, and configuration information for the new collation.

> **Note**
>
> If you modify an existing collation, that may affect the ordering of rows for indexes on columns that use the collation. In this case, rebuild any such indexes to avoid problems such as incorrect query results. For further information, see Section 2.10.3, “Checking Whether Tables or Indexes Must Be Rebuilt”.

Additional Resources

- The Unicode Collation Algorithm (UCA) specification: http://www.unicode.org/reports/tr10/

- The Locale Data Markup Language (LDML) specification: http://www.unicode.org/reports/tr35/

- MySQL Blog article “Instructions for adding a new Unicode collation”: http://blogs.mysql.com/peterg/2008/05/19/instructions-for-adding-a-new-unicode-collation/

10.4.1 Collation Implementation Types

MySQL implements several types of collations:

Simple collations for 8-bit character sets

This kind of collation is implemented using an array of 256 weights that defines a one-to-one mapping from character codes to weights. `latin1_swedish_ci` is an example. It is a case-insensitive collation, so the uppercase and lowercase versions of a character have the same weights and they compare as equal.

```
mysql> SET NAMES 'latin1' COLLATE 'latin1_swedish_ci';
Query OK, 0 rows affected (0.01 sec)

mysql> SELECT HEX(WEIGHT_STRING('a')), HEX(WEIGHT_STRING('A'));
+-------------------------+-------------------------+
| HEX(WEIGHT_STRING('a')) | HEX(WEIGHT_STRING('A')) |
+-------------------------+-------------------------+
| 41                      | 41                      |
+-------------------------+-------------------------+
1 row in set (0.01 sec)

mysql> SELECT 'a' = 'A';
+-----------+
| 'a' = 'A' |
+-----------+
|         1 |
+-----------+
1 row in set (0.12 sec)
```

For implementation instructions, see Section 10.4.3, “Adding a Simple Collation to an 8-Bit Character Set”.

Complex collations for 8-bit character sets

This kind of collation is implemented using functions in a C source file that define how to order characters, as described in Section 10.3, “Adding a Character Set”.

Collations for non-Unicode multi-byte character sets

For this type of collation, 8-bit (single-byte) and multi-byte characters are handled differently. For 8-bit characters, character codes map to weights in case-insensitive fashion. (For example, the single-byte characters 'a' and 'A' both have a weight of 0x41.) For multi-byte characters, there are two types of relationship between character codes and weights:

- Weights equal character codes. sjis_japanese_ci is an example of this kind of collation. The multi-byte character 'ぢ' has a character code of 0x82C0, and the weight is also 0x82C0.

```
mysql> CREATE TABLE t1
    -> (c1 VARCHAR(2) CHARACTER SET sjis COLLATE sjis_japanese_ci);
Query OK, 0 rows affected (0.01 sec)

mysql> INSERT INTO t1 VALUES ('a'),('A'),(0x82C0);
Query OK, 3 rows affected (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> SELECT c1, HEX(c1), HEX(WEIGHT_STRING(c1)) FROM t1;
+------+---------+------------------------+
| c1   | HEX(c1) | HEX(WEIGHT_STRING(c1)) |
+------+---------+------------------------+
| a    | 61      | 41                     |
| A    | 41      | 41                     |
| ぢ    | 82C0    | 82C0                   |
+------+---------+------------------------+
3 rows in set (0.00 sec)
```

- Character codes map one-to-one to weights, but a code is not necessarily equal to the weight. gbk_chinese_ci is an example of this kind of collation. The multi-byte character '膰' has a character code of 0x81B0 but a weight of 0xC286.

```
mysql> CREATE TABLE t1
    -> (c1 VARCHAR(2) CHARACTER SET gbk COLLATE gbk_chinese_ci);
Query OK, 0 rows affected (0.33 sec)

mysql> INSERT INTO t1 VALUES ('a'),('A'),(0x81B0);
Query OK, 3 rows affected (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> SELECT c1, HEX(c1), HEX(WEIGHT_STRING(c1)) FROM t1;
+------+---------+------------------------+
| c1   | HEX(c1) | HEX(WEIGHT_STRING(c1)) |
+------+---------+------------------------+
| a    | 61      | 41                     |
| A    | 41      | 41                     |
| 膰    | 81B0    | C286                   |
+------+---------+------------------------+
3 rows in set (0.00 sec)
```

For implementation instructions, see Section 10.3, “Adding a Character Set”.

Collations for Unicode multi-byte character sets

Some of these collations are based on the Unicode Collation Algorithm (UCA), others are not.

Non-UCA collations have a one-to-one mapping from character code to weight. In MySQL, such collations are case insensitive and accent insensitive. `utf8_general_ci` is an example: `'a'`, `'A'`, `'À'`, and `'á'` each have different character codes but all have a weight of `0x0041` and compare as equal.

```
mysql> SET NAMES 'utf8' COLLATE 'utf8_general_ci';
Query OK, 0 rows affected (0.00 sec)

mysql> CREATE TABLE t1
    -> (c1 CHAR(1) CHARACTER SET UTF8 COLLATE utf8_general_ci);
Query OK, 0 rows affected (0.01 sec)

mysql> INSERT INTO t1 VALUES ('a'),('A'),('À'),('á');
Query OK, 4 rows affected (0.00 sec)
Records: 4  Duplicates: 0  Warnings: 0

mysql> SELECT c1, HEX(c1), HEX(WEIGHT_STRING(c1)) FROM t1;
+------+---------+------------------------+
| c1   | HEX(c1) | HEX(WEIGHT_STRING(c1)) |
+------+---------+------------------------+
| a    | 61      | 0041                   |
| A    | 41      | 0041                   |
| À    | C380    | 0041                   |
| á    | C3A1    | 0041                   |
+------+---------+------------------------+
4 rows in set (0.00 sec)
```

UCA-based collations in MySQL have these properties:

- If a character has weights, each weight uses 2 bytes (16 bits).
- A character may have zero weights (or an empty weight). In this case, the character is ignorable. Example: "U+0000 NULL" does not have a weight and is ignorable.
- A character may have one weight. Example: `'a'` has a weight of `0x0E33`.

  ```
  mysql> SET NAMES 'utf8' COLLATE 'utf8_unicode_ci';
  Query OK, 0 rows affected (0.05 sec)

  mysql> SELECT HEX('a'), HEX(WEIGHT_STRING('a'));
  +----------+------------------------+
  | HEX('a') | HEX(WEIGHT_STRING('a')) |
  +----------+------------------------+
  | 61       | 0E33                   |
  +----------+------------------------+
  1 row in set (0.02 sec)
  ```

- A character may have many weights. This is an expansion. Example: The German letter `'ß'` (SZ ligature, or SHARP S) has a weight of `0x0FEA0FEA`.

  ```
  mysql> SET NAMES 'utf8' COLLATE 'utf8_unicode_ci';
  Query OK, 0 rows affected (0.11 sec)

  mysql> SELECT HEX('ß'), HEX(WEIGHT_STRING('ß'));
  +-----------+-------------------------+
  | HEX('ß')  | HEX(WEIGHT_STRING('ß'))  |
  ```

```
+-----------+-------------------------+
| C39F      | 0FEA0FEA                |
+-----------+-------------------------+
1 row in set (0.00 sec)
```

- Many characters may have one weight. This is a contraction. Example: 'ch' is a single letter in Czech and has a weight of 0x0EE2.

```
mysql> SET NAMES 'utf8' COLLATE 'utf8_czech_ci';
Query OK, 0 rows affected (0.09 sec)

mysql> SELECT HEX('ch'), HEX(WEIGHT_STRING('ch'));
+-----------+-------------------------+
| HEX('ch') | HEX(WEIGHT_STRING('ch')) |
+-----------+-------------------------+
| 6368      | 0EE2                    |
+-----------+-------------------------+
1 row in set (0.00 sec)
```

A many-characters-to-many-weights mapping is also possible (this is contraction with expansion), but is not supported by MySQL.

For implementation instructions, for a non-UCA collation, see Section 10.3, “Adding a Character Set”. For a UCA collation, see Section 10.4.4, “Adding a UCA Collation to a Unicode Character Set”.

Miscellaneous collations

There are also a few collations that do not fall into any of the previous categories.

10.4.2 Choosing a Collation ID

Each collation must have a unique ID. To add a collation, you must choose an ID value that is not currently used. MySQL supports two-byte collation IDs. The range of IDs from 1024 to 2047 is reserved for user-defined collations. The collation ID that you choose will appear in these contexts:

- The ID column of the INFORMATION_SCHEMA.COLLATIONS table.
- The Id column of SHOW COLLATION output.
- The charsetnr member of the MYSQL_FIELD C API data structure.
- The number member of the MY_CHARSET_INFO data structure returned by the mysql_get_character_set_info() C API function.

To determine the largest currently used ID, issue the following statement:

```
mysql> SELECT MAX(ID) FROM INFORMATION_SCHEMA.COLLATIONS;
+---------+
| MAX(ID) |
+---------+
|     210 |
+---------+
```

To display a list of all currently used IDs, issue this statement:

```
mysql> SELECT ID FROM INFORMATION_SCHEMA.COLLATIONS ORDER BY ID;
+-----+
| ID  |
+-----+
|   1 |
|   2 |
| ... |
|  52 |
|  53 |
|  57 |
|  58 |
| ... |
|  98 |
|  99 |
| 128 |
| 129 |
| ... |
| 210 |
+-----+
```

Warning

Before MySQL 5.5, which provides for a range of user-defined collation IDs, you must choose an ID in the range from 1 to 254. In this case, if you upgrade MySQL, you may find that the collation ID you choose has been assigned to a collation included in the new MySQL distribution. In this case, you will need to choose a new value for your own collation.

In addition, before upgrading, you should save the configuration files that you change. If you upgrade in place, the process will replace the your modified files.

10.4.3 Adding a Simple Collation to an 8-Bit Character Set

This section describes how to add a simple collation for an 8-bit character set by writing the `<collation>` elements associated with a `<charset>` character set description in the MySQL `Index.xml` file. The procedure described here does not require recompiling MySQL. The example adds a collation named `latin1_test_ci` to the `latin1` character set.

1. Choose a collation ID, as shown in Section 10.4.2, “Choosing a Collation ID”. The following steps use an ID of 1024.

2. Modify the `Index.xml` and `latin1.xml` configuration files. These files will be located in the directory named by the `character_sets_dir` system variable. You can check the variable value as follows, although the path name might be different on your system:

```
mysql> SHOW VARIABLES LIKE 'character_sets_dir';
+--------------------+-----------------------------------------+
| Variable_name      | Value                                   |
+--------------------+-----------------------------------------+
| character_sets_dir | /user/local/mysql/share/mysql/charsets/ |
+--------------------+-----------------------------------------+
```

3. Choose a name for the collation and list it in the `Index.xml` file. Find the `<charset>` element for the character set to which the collation is being added, and add a `<collation>` element that indicates the collation name and ID, to associate the name with the ID. For example:

```
<charset name="latin1">
  ...
  <collation name="latin1_test_ci" id="1024"/>
  ...
</charset>
```

4. In the `latin1.xml` configuration file, add a `<collation>` element that names the collation and that contains a `<map>` element that defines a character code-to-weight mapping table for character codes 0 to 255. Each value within the `<map>` element must be a number in hexadecimal format.

```
<collation name="latin1_test_ci">
<map>
 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F
 20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F
 30 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F
 40 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F
 50 51 52 53 54 55 56 57 58 59 5A 5B 5C 5D 5E 5F
 60 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F
 50 51 52 53 54 55 56 57 58 59 5A 7B 7C 7D 7E 7F
 80 81 82 83 84 85 86 87 88 89 8A 8B 8C 8D 8E 8F
 90 91 92 93 94 95 96 97 98 99 9A 9B 9C 9D 9E 9F
 A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 AA AB AC AD AE AF
 B0 B1 B2 B3 B4 B5 B6 B7 B8 B9 BA BB BC BD BE BF
 41 41 41 41 5B 5D 5B 43 45 45 45 45 49 49 49 49
 44 4E 4F 4F 4F 4F 5C D7 5C 55 55 55 59 59 DE DF
 41 41 41 41 5B 5D 5B 43 45 45 45 45 49 49 49 49
 44 4E 4F 4F 4F 4F 5C F7 5C 55 55 55 59 59 DE FF
</map>
</collation>
```

5. Restart the server and use this statement to verify that the collation is present:

```
mysql> SHOW COLLATION LIKE 'latin1_test_ci';
+----------------+---------+------+---------+----------+---------+
| Collation      | Charset | Id   | Default | Compiled | Sortlen |
+----------------+---------+------+---------+----------+---------+
| latin1_test_ci | latin1  | 1024 |         |          |       1 |
+----------------+---------+------+---------+----------+---------+
```

10.4.4 Adding a UCA Collation to a Unicode Character Set

This section describes how to add a UCA collation for a Unicode character set by writing the `<collation>` element within a `<charset>` character set description in the MySQL `Index.xml` file. The procedure described here does not require recompiling MySQL. It uses a subset of the Locale Data Markup Language (LDML) specification, which is available at http://www.unicode.org/reports/tr35/. With this method, you need not define the entire collation. Instead, you begin with an existing “base” collation and describe the new collation in terms of how it differs from the base collation. The following table lists the base collations of the Unicode character sets for which UCA collations can be defined. It is not possible to create user-defined UCA collations for `utf16le`; there is no `utf16le_unicode_ci` collation that would serve as the basis for such collations.

Table 10.1 MySQL Character Sets Available for User-Defined UCA Collations

Character Set	Base Collation
`utf8`	`utf8_unicode_ci`
`ucs2`	`ucs2_unicode_ci`
`utf16`	`utf16_unicode_ci`
`utf32`	`utf32_unicode_ci`

The following sections show how to add a collation that is defined using LDML syntax, and provide a summary of LDML rules supported in MySQL.

10.4.4.1 Defining a UCA Collation Using LDML Syntax

To add a UCA collation for a Unicode character set without recompiling MySQL, use the following procedure. If you are unfamiliar with the LDML rules used to describe the collation's sort characteristics, see Section 10.4.4.2, “LDML Syntax Supported in MySQL”.

The example adds a collation named `utf8_phone_ci` to the `utf8` character set. The collation is designed for a scenario involving a Web application for which users post their names and phone numbers. Phone numbers can be given in very different formats:

```
+7-12345-67
+7-12-345-67
+7 12 345 67
+7 (12) 345 67
+71234567
```

The problem raised by dealing with these kinds of values is that the varying permissible formats make searching for a specific phone number very difficult. The solution is to define a new collation that reorders punctuation characters, making them ignorable.

1. Choose a collation ID, as shown in Section 10.4.2, “Choosing a Collation ID”. The following steps use an ID of 1029.

2. To modify the `Index.xml` configuration file. This file will be located in the directory named by the `character_sets_dir` system variable. You can check the variable value as follows, although the path name might be different on your system:

```
mysql> SHOW VARIABLES LIKE 'character_sets_dir';
+--------------------+-----------------------------------------+
| Variable_name      | Value                                   |
+--------------------+-----------------------------------------+
| character_sets_dir | /user/local/mysql/share/mysql/charsets/ |
+--------------------+-----------------------------------------+
```

3. Choose a name for the collation and list it in the `Index.xml` file. In addition, you'll need to provide the collation ordering rules. Find the `<charset>` element for the character set to which the collation is being added, and add a `<collation>` element that indicates the collation name and ID, to associate the name with the ID. Within the `<collation>` element, provide a `<rules>` element containing the ordering rules:

```
<charset name="utf8">
  ...
  <collation name="utf8_phone_ci" id="1029">
    <rules>
      <reset>\u0000</reset>
      <i>\u0020</i> <!-- space -->
      <i>\u0028</i> <!-- left parenthesis -->
      <i>\u0029</i> <!-- right parenthesis -->
      <i>\u002B</i> <!-- plus -->
      <i>\u002D</i> <!-- hyphen -->
    </rules>
  </collation>
  ...
</charset>
```

4. If you want a similar collation for other Unicode character sets, add other `<collation>` elements. For example, to define `ucs2_phone_ci`, add a `<collation>` element to the `<charset name="ucs2">` element. Remember that each collation must have its own unique ID.

5. Restart the server and use this statement to verify that the collation is present:

```
mysql> SHOW COLLATION LIKE 'utf8_phone_ci';
+---------------+---------+------+---------+----------+---------+
| Collation     | Charset | Id   | Default | Compiled | Sortlen |
+---------------+---------+------+---------+----------+---------+
| utf8_phone_ci | utf8    | 1029 |         |          |       8 |
+---------------+---------+------+---------+----------+---------+
```

Now test the collation to make sure that it has the desired properties.

Create a table containing some sample phone numbers using the new collation:

```
mysql> CREATE TABLE phonebook (
    ->   name VARCHAR(64),
    ->   phone VARCHAR(64) CHARACTER SET utf8 COLLATE utf8_phone_ci
    -> );
Query OK, 0 rows affected (0.09 sec)

mysql> INSERT INTO phonebook VALUES ('Svoj','+7 912 800 80 02');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO phonebook VALUES ('Hf','+7 (912) 800 80 04');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO phonebook VALUES ('Bar','+7-912-800-80-01');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO phonebook VALUES ('Ramil','(7912) 800 80 03');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO phonebook VALUES ('Sanja','+380 (912) 8008005');
Query OK, 1 row affected (0.00 sec)
```

Run some queries to see whether the ignored punctuation characters are in fact ignored for sorting and comparisons:

```
mysql> SELECT * FROM phonebook ORDER BY phone;
+-------+--------------------+
| name  | phone              |
+-------+--------------------+
| Sanja | +380 (912) 8008005 |
| Bar   | +7-912-800-80-01   |
| Svoj  | +7 912 800 80 02   |
| Ramil | (7912) 800 80 03   |
| Hf    | +7 (912) 800 80 04 |
+-------+--------------------+
5 rows in set (0.00 sec)

mysql> SELECT * FROM phonebook WHERE phone='+7(912)800-80-01';
+------+------------------+
| name | phone            |
+------+------------------+
| Bar  | +7-912-800-80-01 |
+------+------------------+
1 row in set (0.00 sec)

mysql> SELECT * FROM phonebook WHERE phone='79128008001';
+------+------------------+
| name | phone            |
+------+------------------+
| Bar  | +7-912-800-80-01 |
+------+------------------+
1 row in set (0.00 sec)

mysql> SELECT * FROM phonebook WHERE phone='7 9 1 2 8 0 0 8 0 0 1';
+------+------------------+
| name | phone            |
+------+------------------+
| Bar  | +7-912-800-80-01 |
+------+------------------+
1 row in set (0.00 sec)
```

10.4.4.2 LDML Syntax Supported in MySQL

This section describes the LDML syntax that MySQL recognizes. This is a subset of the syntax described in the LDML specification available at http://www.unicode.org/reports/tr35/, which should be consulted for further information. MySQL recognizes a large enough subset of the syntax that, in many cases, it is possible to download a collation definition from the Unicode Common Locale Data Repository and paste the relevant part (that is, the part between the `<rules>` and `</rules>` tags) into the MySQL `Index.xml` file. The rules described here are all supported except that character sorting occurs only at the primary level. Rules that specify differences at secondary or higher sort levels are recognized (and thus can be included in collation definitions) but are treated as equality at the primary level.

The MySQL server generates diagnostics when it finds problems while parsing the `Index.xml` file. See Section 10.4.4.3, “Diagnostics During `Index.xml` Parsing”.

Character Representation

Characters named in LDML rules can be written literally or in `\u`*nnnn* format, where *nnnn* is the hexadecimal Unicode code point value. For example, `A` and `á` can be written literally or as `\u0041` and `\u00E1`. Within hexadecimal values, the digits `A` through `F` are not case sensitive; `\u00E1` and `\u00e1` are equivalent. For UCA 4.0.0 collations, hexadecimal notation can be used only for characters in the Basic Multilingual Plane, not for characters outside the BMP range of `0000` to `FFFF`. For UCA 5.2.0 collations, hexadecimal notation can be used for any character.

The `Index.xml` file itself should be written using UTF-8 encoding.

Syntax Rules

LDML has reset rules and shift rules to specify character ordering. Orderings are given as a set of rules that begin with a reset rule that establishes an anchor point, followed by shift rules that indicate how characters sort relative to the anchor point.

- A `<reset>` rule does not specify any ordering in and of itself. Instead, it “resets” the ordering for subsequent shift rules to cause them to be taken in relation to a given character. Either of the following rules resets subsequent shift rules to be taken in relation to the letter `'A'`:

  ```
  <reset>A</reset>

  <reset>\u0041</reset>
  ```

- The `<p>`, `<s>`, and `<t>` shift rules define primary, secondary, and tertiary differences of a character from another character:

  - Use primary differences to distinguish separate letters.

  - Use secondary differences to distinguish accent variations.

  - Use tertiary differences to distinguish lettercase variations.

Either of these rules specifies a primary shift rule for the 'G' character:

```
<p>G</p>

<p>\u0047</p>
```

- The <i> shift rule indicates that one character sorts identically to another. The following rules cause 'b' to sort the same as 'a':

```
<reset>a</reset>
<i>b</i>
```

- Abbreviated shift syntax specifies multiple shift rules using a single pair of tags. The following table shows the correspondence between abbreviated syntax rules and the equivalent nonabbreviated rules.

Table 10.2 Abbreviated Shift Syntax

Abbreviated Syntax	Nonabbreviated Syntax
<pc>xyz</pc>	<p>x</p><p>y</p><p>z</p>
<sc>xyz</sc>	<s>x</s><s>y</s><s>z</s>
<tc>xyz</tc>	<t>x</t><t>y</t><t>z</t>
<ic>xyz</ic>	<i>x</i><i>y</i><i>z</i>

- An expansion is a reset rule that establishes an anchor point for a multiple-character sequence. MySQL supports expansions 2 to 6 characters long. The following rules put 'z' greater at the primary level than the sequence of three characters 'abc':

```
<reset>abc</reset>
<p>z</p>
```

- A contraction is a shift rule that sorts a multiple-character sequence. MySQL supports contractions 2 to 6 characters long. The following rules put the sequence of three characters 'xyz' greater at the primary level than 'a':

```
<reset>a</reset>
<p>xyz</p>
```

- Long expansions and long contractions can be used together. These rules put the sequence of three characters 'xyz' greater at the primary level than the sequence of three characters 'abc':

```
<reset>abc</reset>
<p>xyz</p>
```

- Normal expansion syntax uses <x> plus <extend> elements to specify an expansion. The following rules put the character 'k' greater at the secondary level than the sequence 'ch'. That is, 'k' behaves as if it expands to a character after 'c' followed by 'h':

```
<reset>c</reset>
<x><s>k</s><extend>h</extend></x>
```

This syntax permits long sequences. These rules sort the sequence 'ccs' greater at the tertiary level than the sequence 'cscs':

```
<reset>cs</reset>
<x><t>ccs</t><extend>cs</extend></x>
```

The LDML specification describes normal expansion syntax as “tricky.” See that specification for details.

- Previous context syntax uses `<x>` plus `<context>` elements to specify that the context before a character affects how it sorts. The following rules put `'-'` greater at the secondary level than `'a'`, but only when `'-'` occurs after `'b'`:

```
<reset>a</reset>
<x><context>b</context><s>-</s></x>
```

- Previous context syntax can include the `<extend>` element. These rules put `'def'` greater at the primary level than `'aghi'`, but only when `'def'` comes after `'abc'`:

```
<reset>a</reset>
<x><context>abc</context><p>def</p><extend>ghi</extend></x>
```

- Reset rules permit a `before` attribute. Normally, shift rules after a reset rule indicate characters that sort after the reset character. Shift rules after a reset rule that has the `before` attribute indicate characters that sort before the reset character. The following rules put the character `'b'` immediately before `'a'` at the primary level:

```
<reset before="primary">a</reset>
<p>b</p>
```

Permissible `before` attribute values specify the sort level by name or the equivalent numeric value:

```
<reset before="primary">
<reset before="1">

<reset before="secondary">
<reset before="2">

<reset before="tertiary">
<reset before="3">
```

- A reset rule can name a logical reset position rather than a literal character:

```
<first_tertiary_ignorable/>
<last_tertiary_ignorable/>
<first_secondary_ignorable/>
<last_secondary_ignorable/>
<first_primary_ignorable/>
<last_primary_ignorable/>
<first_variable/>
<last_variable/>
<first_non_ignorable/>
<last_non_ignorable/>
<first_trailing/>
<last_trailing/>
```

These rules put `'z'` greater at the primary level than nonignorable characters that have a Default Unicode Collation Element Table (DUCET) entry and that are not CJK:

```
<reset><last_non_ignorable/></reset>
```

```
<p>z</p>
```

Logical positions have the code points shown in the following table.

Table 10.3 Logical Reset Position Code Points

Logical Position	Unicode 4.0.0 Code Point	Unicode 5.2.0 Code Point
`<first_non_ignorable/>`	U+02D0	U+02D0
`<last_non_ignorable/>`	U+A48C	U+1342E
`<first_primary_ignorable/>`	U+0332	U+0332
`<last_primary_ignorable/>`	U+20EA	U+101FD
`<first_secondary_ignorable/>`	U+0000	U+0000
`<last_secondary_ignorable/>`	U+FE73	U+FE73
`<first_tertiary_ignorable/>`	U+0000	U+0000
`<last_tertiary_ignorable/>`	U+FE73	U+FE73
`<first_trailing/>`	U+0000	U+0000
`<last_trailing/>`	U+0000	U+0000
`<first_variable/>`	U+0009	U+0009
`<last_variable/>`	U+2183	U+1D371

- The `<collation>` element permits a `shift-after-method` attribute that affects character weight calculation for shift rules. The attribute has these permitted values:

  - `simple`: Calculate character weights as for reset rules that do not have a `before` attribute. This is the default if the attribute is not given.

  - `expand`: Use expansions for shifts after reset rules.

  Suppose that `'0'` and `'1'` have weights of `0E29` and `0E2A` and we want to put all basic Latin letters between `'0'` and `'1'`:

  ```
  <reset>0</reset>
  <pc>abcdefghijklmnopqrstuvwxyz</pc>
  ```

  For simple shift mode, weights are calculated as follows:

  ```
  'a' has weight 0E29+1
  'b' has weight 0E29+2
  'c' has weight 0E29+3
  ...
  ```

  However, there are not enough vacant positions to put 26 characters between `'0'` and `'1'`. The result is that digits and letters are intermixed.

  To solve this, use `shift-after-method="expand"`. Then weights are calculated like this:

  ```
  'a' has weight [0E29][233D+1]
  'b' has weight [0E29][233D+2]
  'c' has weight [0E29][233D+3]
  ...
  ```

233D is the UCA 4.0.0 weight for character 0xA48C, which is the last nonignorable character (a sort of the greatest character in the collation, excluding CJK). UCA 5.2.0 is similar but uses 3ACA, for character 0x1342E.

MySQL-Specific LDML Extensions

In MySQL 5.7, an extension to LDML rules permits the <collation> element to include an optional version attribute in <collation> tags to indicate the UCA version on which the collation is based. If the version attribute is omitted, its default value is 4.0.0. For example, this specification indicates a collation that is based on UCA 5.2.0:

```
<collation id="nnn" name="utf8_xxx_ci" version="5.2.0">
...
</collation>
```

10.4.4.3 Diagnostics During Index.xml Parsing

The MySQL server generates diagnostics when it finds problems while parsing the Index.xml file:

- Unknown tags are written to the error log. For example, the following message results if a collation definition contains a <aaa> tag:

  ```
  [Warning] Buffered warning: Unknown LDML tag:
  'charsets/charset/collation/rules/aaa'
  ```

- If collation initialization is not possible, the server reports an “Unknown collation” error, and also generates warnings explaining the problems, such as in the previous example. In other cases, when a collation description is generally correct but contains some unknown tags, the collation is initialized and is available for use. The unknown parts are ignored, but a warning is generated in the error log.

- Problems with collations generate warnings that clients can display with SHOW WARNINGS. Suppose that a reset rule contains an expansion longer than the maximum supported length of 6 characters:

  ```
  <reset>abcdefghi</reset>
  <i>x</i>
  ```

  An attempt to use the collation produces warnings:

  ```
  mysql> SELECT _utf8'test' COLLATE utf8_test_ci;
  ERROR 1273 (HY000): Unknown collation: 'utf8_test_ci'
  mysql> SHOW WARNINGS;
  +---------+------+-----------------------------------------+
  | Level   | Code | Message                                 |
  +---------+------+-----------------------------------------+
  | Error   | 1273 | Unknown collation: 'utf8_test_ci'       |
  | Warning | 1273 | Expansion is too long at 'abcdefghi=x' |
  +---------+------+-----------------------------------------+
  ```

10.5 Character Set Configuration

You can change the default server character set and collation with the --character-set-server and --collation-server options when you start the server. The collation must be a legal collation for the default character set. (Use the SHOW COLLATION statement to determine which collations are available for each character set.) See Section 5.1.3, “Server Command Options”.

If you try to use a character set that is not compiled into your binary, you might run into the following problems:

- Your program uses an incorrect path to determine where the character sets are stored (which is typically the `share/mysql/charsets` or `share/charsets` directory under the MySQL installation directory). This can be fixed by using the `--character-sets-dir` option when you run the program in question. For example, to specify a directory to be used by MySQL client programs, list it in the `[client]` group of your option file. The examples given here show what the setting might look like for Unix or Windows, respectively:

  ```
  [client]
  character-sets-dir=/usr/local/mysql/share/mysql/charsets

  [client]
  character-sets-dir="C:/Program Files/MySQL/MySQL Server 5.7/share/charsets"
  ```

- The character set is a complex character set that cannot be loaded dynamically. In this case, you must recompile the program with support for the character set.

  For Unicode character sets, you can define collations without recompiling by using LDML notation. See Section 10.4.4, “Adding a UCA Collation to a Unicode Character Set”.

- The character set is a dynamic character set, but you do not have a configuration file for it. In this case, you should install the configuration file for the character set from a new MySQL distribution.

- If your character set index file does not contain the name for the character set, your program displays an error message. The file is named `Index.xml` and the message is:

  ```
  Character set 'charset_name' is not a compiled character set and is not
  specified in the '/usr/share/mysql/charsets/Index.xml' file
  ```

  To solve this problem, you should either get a new index file or manually add the name of any missing character sets to the current file.

You can force client programs to use specific character set as follows:

```
[client]
default-character-set=charset_name
```

This is normally unnecessary. However, when `character_set_system` differs from `character_set_server` or `character_set_client`, and you input characters manually (as database object identifiers, column values, or both), these may be displayed incorrectly in output from the client or the output itself may be formatted incorrectly. In such cases, starting the mysql client with `--default-character-set=`*`system_character_set`*—that is, setting the client character set to match the system character set—should fix the problem.

For `MyISAM` tables, you can check the character set name and number for a table with `myisamchk -dvv` *`tbl_name`*.

10.6 MySQL Server Time Zone Support

The MySQL server maintains several time zone settings:

- The system time zone. When the server starts, it attempts to determine the time zone of the host machine and uses it to set the `system_time_zone` system variable. The value does not change thereafter.

You can set the system time zone for MySQL Server at startup with the `--timezone=timezone_name` option to `mysqld_safe`. You can also set it by setting the `TZ` environment variable before you start `mysqld`. The permissible values for `--timezone` or `TZ` are system dependent. Consult your operating system documentation to see what values are acceptable.

- The server's current time zone. The global `time_zone` system variable indicates the time zone the server currently is operating in. The initial value for `time_zone` is `'SYSTEM'`, which indicates that the server time zone is the same as the system time zone.

  The initial global server time zone value can be specified explicitly at startup with the `--default-time-zone=timezone` option on the command line, or you can use the following line in an option file:

  ```
  default-time-zone='timezone'
  ```

  If you have the `SUPER` privilege, you can set the global server time zone value at runtime with this statement:

  ```
  mysql> SET GLOBAL time_zone = timezone;
  ```

- Per-connection time zones. Each client that connects has its own time zone setting, given by the session `time_zone` variable. Initially, the session variable takes its value from the global `time_zone` variable, but the client can change its own time zone with this statement:

  ```
  mysql> SET time_zone = timezone;
  ```

The current session time zone setting affects display and storage of time values that are zone-sensitive. This includes the values displayed by functions such as `NOW()` or `CURTIME()`, and values stored in and retrieved from `TIMESTAMP` columns. Values for `TIMESTAMP` columns are converted from the current time zone to UTC for storage, and from UTC to the current time zone for retrieval.

The current time zone setting does not affect values displayed by functions such as `UTC_TIMESTAMP()` or values in `DATE`, `TIME`, or `DATETIME` columns. Nor are values in those data types stored in UTC; the time zone applies for them only when converting from `TIMESTAMP` values. If you want locale-specific arithmetic for `DATE`, `TIME`, or `DATETIME` values, convert them to UTC, perform the arithmetic, and then convert back.

The current values of the global and client-specific time zones can be retrieved like this:

```
mysql> SELECT @@global.time_zone, @@session.time_zone;
```

`timezone` values can be given in several formats, none of which are case sensitive:

- The value `'SYSTEM'` indicates that the time zone should be the same as the system time zone.

- The value can be given as a string indicating an offset from UTC, such as `'+10:00'` or `'-6:00'`.

- The value can be given as a named time zone, such as `'Europe/Helsinki'`, `'US/Eastern'`, or `'MET'`. Named time zones can be used only if the time zone information tables in the `mysql` database have been created and populated.

The MySQL installation procedure creates the time zone tables in the `mysql` database, but does not load them. You must do so manually using the following instructions. (If you are upgrading to MySQL 4.1.3 or later from an earlier version, you can create the tables by upgrading your `mysql` database. Use the instructions in Section 4.4.7, “`mysql_upgrade` — Check and Upgrade MySQL Tables”. After creating the tables, you can load them.)

Note

Loading the time zone information is not necessarily a one-time operation because the information changes occasionally. For example, the rules for Daylight Saving Time in the United States, Mexico, and parts of Canada changed in 2007. When such changes occur, applications that use the old rules become out of date and you may find it necessary to reload the time zone tables to keep the information used by your MySQL server current. See the notes at the end of this section.

If your system has its own *zoneinfo* database (the set of files describing time zones), you should use the `mysql_tzinfo_to_sql` program for filling the time zone tables. Examples of such systems are Linux, FreeBSD, Solaris, and Mac OS X. One likely location for these files is the `/usr/share/zoneinfo` directory. If your system does not have a zoneinfo database, you can use the downloadable package described later in this section.

The `mysql_tzinfo_to_sql` program is used to load the time zone tables. On the command line, pass the zoneinfo directory path name to `mysql_tzinfo_to_sql` and send the output into the `mysql` program. For example:

```
shell> mysql_tzinfo_to_sql /usr/share/zoneinfo | mysql -u root mysql
```

`mysql_tzinfo_to_sql` reads your system's time zone files and generates SQL statements from them. `mysql` processes those statements to load the time zone tables.

`mysql_tzinfo_to_sql` also can be used to load a single time zone file or to generate leap second information:

- To load a single time zone file *`tz_file`* that corresponds to a time zone name *`tz_name`*, invoke `mysql_tzinfo_to_sql` like this:

  ```
  shell> mysql_tzinfo_to_sql tz_file tz_name | mysql -u root mysql
  ```

  With this approach, you must execute a separate command to load the time zone file for each named zone that the server needs to know about.

- If your time zone needs to account for leap seconds, initialize the leap second information like this, where *`tz_file`* is the name of your time zone file:

  ```
  shell> mysql_tzinfo_to_sql --leap tz_file | mysql -u root mysql
  ```

- After running `mysql_tzinfo_to_sql`, it is best to restart the server so that it does not continue to use any previously cached time zone data.

If your system is one that has no zoneinfo database (for example, Windows), you can use the package of pre-built time zone tables that is available for download at the MySQL Developer Zone:

```
http://dev.mysql.com/downloads/timezones.html
```

This time zone package contains `.frm`, `.MYD`, and `.MYI` files for the `MyISAM` time zone tables. These tables should be part of the `mysql` database, so you should place the files in the `mysql` subdirectory of your MySQL server's data directory. The server should be stopped while you do this and restarted afterward.

Warning

Do not use the downloadable package if your system has a zoneinfo database. Use the `mysql_tzinfo_to_sql` utility instead. Otherwise, you may cause a difference in datetime handling between MySQL and other applications on your system.

For information about time zone settings in replication setup, please see Section 16.4.1, “Replication Features and Issues”.

10.6.1 Staying Current with Time Zone Changes

As mentioned earlier, when the time zone rules change, applications that use the old rules become out of date. To stay current, it is necessary to make sure that your system uses current time zone information is used. For MySQL, there are two factors to consider in staying current:

- The operating system time affects the value that the MySQL server uses for times if its time zone is set to `SYSTEM`. Make sure that your operating system is using the latest time zone information. For most operating systems, the latest update or service pack prepares your system for the time changes. Check the Web site for your operating system vendor for an update that addresses the time changes.

- If you replace the system's `/etc/localtime` timezone file with a version that uses rules differing from those in effect at `mysqld` startup, you should restart `mysqld` so that it uses the updated rules. Otherwise, `mysqld` might not notice when the system changes its time.

- If you use named time zones with MySQL, make sure that the time zone tables in the `mysql` database are up to date. If your system has its own zoneinfo database, you should reload the MySQL time zone tables whenever the zoneinfo database is updated, using the instructions given earlier in this section. For systems that do not have their own zoneinfo database, check the MySQL Developer Zone for updates. When a new update is available, download it and use it to replace your current time zone tables. `mysqld` caches time zone information that it looks up, so after replacing the time zone tables, you should restart `mysqld` to make sure that it does not continue to serve outdated time zone data.

If you are uncertain whether named time zones are available, for use either as the server's time zone setting or by clients that set their own time zone, check whether your time zone tables are empty. The following query determines whether the table that contains time zone names has any rows:

```
mysql> SELECT COUNT(*) FROM mysql.time_zone_name;
+----------+
| COUNT(*) |
+----------+
|        0 |
+----------+
```

A count of zero indicates that the table is empty. In this case, no one can be using named time zones, and you don't need to update the tables. A count greater than zero indicates that the table is not empty and that its contents are available to be used for named time zone support. In this case, you should be sure to reload your time zone tables so that anyone who uses named time zones will get correct query results.

To check whether your MySQL installation is updated properly for a change in Daylight Saving Time rules, use a test like the one following. The example uses values that are appropriate for the 2007 DST 1-hour change that occurs in the United States on March 11 at 2 a.m.

The test uses these two queries:

```
SELECT CONVERT_TZ('2007-03-11 2:00:00','US/Eastern','US/Central');
SELECT CONVERT_TZ('2007-03-11 3:00:00','US/Eastern','US/Central');
```

The two time values indicate the times at which the DST change occurs, and the use of named time zones requires that the time zone tables be used. The desired result is that both queries return the same result (the input time, converted to the equivalent value in the 'US/Central' time zone).

Before updating the time zone tables, you would see an incorrect result like this:

```
mysql> SELECT CONVERT_TZ('2007-03-11 2:00:00','US/Eastern','US/Central');
+------------------------------------------------------------+
| CONVERT_TZ('2007-03-11 2:00:00','US/Eastern','US/Central') |
+------------------------------------------------------------+
| 2007-03-11 01:00:00                                        |
+------------------------------------------------------------+

mysql> SELECT CONVERT_TZ('2007-03-11 3:00:00','US/Eastern','US/Central');
+------------------------------------------------------------+
| CONVERT_TZ('2007-03-11 3:00:00','US/Eastern','US/Central') |
+------------------------------------------------------------+
| 2007-03-11 02:00:00                                        |
+------------------------------------------------------------+
```

After updating the tables, you should see the correct result:

```
mysql> SELECT CONVERT_TZ('2007-03-11 2:00:00','US/Eastern','US/Central');
+------------------------------------------------------------+
| CONVERT_TZ('2007-03-11 2:00:00','US/Eastern','US/Central') |
+------------------------------------------------------------+
| 2007-03-11 01:00:00                                        |
+------------------------------------------------------------+

mysql> SELECT CONVERT_TZ('2007-03-11 3:00:00','US/Eastern','US/Central');
+------------------------------------------------------------+
| CONVERT_TZ('2007-03-11 3:00:00','US/Eastern','US/Central') |
+------------------------------------------------------------+
| 2007-03-11 01:00:00                                        |
+------------------------------------------------------------+
```

10.6.2 Time Zone Leap Second Support

Leap second values are returned with a time part that ends with `:59:59`. This means that a function such as `NOW()` can return the same value for two or three consecutive seconds during the leap second. It remains true that literal temporal values having a time part that ends with `:59:60` or `:59:61` are considered invalid.

If it is necessary to search for `TIMESTAMP` values one second before the leap second, anomalous results may be obtained if you use a comparison with `'YYYY-MM-DD hh:mm:ss'` values. The following example demonstrates this. It changes the local time zone to UTC so there is no difference between internal values (which are in UTC) and displayed values (which have time zone correction applied).

```
mysql> CREATE TABLE t1 (
    ->   a INT,
    ->   ts TIMESTAMP DEFAULT NOW(),
    ->   PRIMARY KEY (ts)
    -> );
Query OK, 0 rows affected (0.01 sec)

mysql> -- change to UTC
mysql> SET time_zone = '+00:00';
Query OK, 0 rows affected (0.00 sec)

mysql> -- Simulate NOW() = '2008-12-31 23:59:59'
mysql> SET timestamp = 1230767999;
```

```
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO t1 (a) VALUES (1);
Query OK, 1 row affected (0.00 sec)

mysql> -- Simulate NOW() = '2008-12-31 23:59:60'
mysql> SET timestamp = 1230768000;
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO t1 (a) VALUES (2);
Query OK, 1 row affected (0.00 sec)

mysql> -- values differ internally but display the same
mysql> SELECT a, ts, UNIX_TIMESTAMP(ts) FROM t1;
+------+---------------------+--------------------+
| a    | ts                  | UNIX_TIMESTAMP(ts) |
+------+---------------------+--------------------+
|    1 | 2008-12-31 23:59:59 |         1230767999 |
|    2 | 2008-12-31 23:59:59 |         1230768000 |
+------+---------------------+--------------------+
2 rows in set (0.00 sec)

mysql> -- only the non-leap value matches
mysql> SELECT * FROM t1 WHERE ts = '2008-12-31 23:59:59';
+------+---------------------+
| a    | ts                  |
+------+---------------------+
|    1 | 2008-12-31 23:59:59 |
+------+---------------------+
1 row in set (0.00 sec)

mysql> -- the leap value with seconds=60 is invalid
mysql> SELECT * FROM t1 WHERE ts = '2008-12-31 23:59:60';
Empty set, 2 warnings (0.00 sec)
```

To work around this, you can use a comparison based on the UTC value actually stored in column, which has the leap second correction applied:

```

mysql> -- selecting using UNIX_TIMESTAMP value return leap value
mysql> SELECT * FROM t1 WHERE UNIX_TIMESTAMP(ts) = 1230768000;
+------+---------------------+
| a    | ts                  |
+------+---------------------+
|    2 | 2008-12-31 23:59:59 |
+------+---------------------+
1 row in set (0.00 sec)
```

10.7 MySQL Server Locale Support

The locale indicated by the `lc_time_names` system variable controls the language used to display day and month names and abbreviations. This variable affects the output from the `DATE_FORMAT()`, `DAYNAME()`, and `MONTHNAME()` functions.

`lc_time_names` does not affect the `STR_TO_DATE()` or `GET_FORMAT()` function.

The `lc_time_names` value does not affect the result from `FORMAT()`, but this function takes an optional third parameter that enables a locale to be specified to be used for the result number's decimal point, thousands separator, and grouping between separators. Permissible locale values are the same as the legal values for the `lc_time_names` system variable.

Locale names have language and region subtags listed by IANA (http://www.iana.org/assignments/language-subtag-registry) such as `'ja_JP'` or `'pt_BR'`. The default value is `'en_US'` regardless of your system's locale setting, but you can set the value at server startup or set the `GLOBAL` value if you have the

SUPER privilege. Any client can examine the value of lc_time_names or set its SESSION value to affect the locale for its own connection.

```
mysql> SET NAMES 'utf8';
Query OK, 0 rows affected (0.09 sec)

mysql> SELECT @@lc_time_names;
+-----------------+
| @@lc_time_names |
+-----------------+
| en_US           |
+-----------------+
1 row in set (0.00 sec)

mysql> SELECT DAYNAME('2010-01-01'), MONTHNAME('2010-01-01');
+-----------------------+-------------------------+
| DAYNAME('2010-01-01') | MONTHNAME('2010-01-01') |
+-----------------------+-------------------------+
| Friday                | January                 |
+-----------------------+-------------------------+
1 row in set (0.00 sec)

mysql> SELECT DATE_FORMAT('2010-01-01','%W %a %M %b');
+-----------------------------------------+
| DATE_FORMAT('2010-01-01','%W %a %M %b') |
+-----------------------------------------+
| Friday Fri January Jan                  |
+-----------------------------------------+
1 row in set (0.00 sec)

mysql> SET lc_time_names = 'es_MX';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @@lc_time_names;
+-----------------+
| @@lc_time_names |
+-----------------+
| es_MX           |
+-----------------+
1 row in set (0.00 sec)

mysql> SELECT DAYNAME('2010-01-01'), MONTHNAME('2010-01-01');
+-----------------------+-------------------------+
| DAYNAME('2010-01-01') | MONTHNAME('2010-01-01') |
+-----------------------+-------------------------+
| viernes               | enero                   |
+-----------------------+-------------------------+
1 row in set (0.00 sec)

mysql> SELECT DATE_FORMAT('2010-01-01','%W %a %M %b');
+-----------------------------------------+
| DATE_FORMAT('2010-01-01','%W %a %M %b') |
+-----------------------------------------+
| viernes vie enero ene                   |
+-----------------------------------------+
1 row in set (0.00 sec)
```

The day or month name for each of the affected functions is converted from utf8 to the character set indicated by the character_set_connection system variable.

lc_time_names may be set to any of the following locale values. The set of locales supported by MySQL may differ from those supported by your operating system.

ar_AE: Arabic - United Arab Emirates	ar_BH: Arabic - Bahrain

ar_DZ: Arabic - Algeria	ar_EG: Arabic - Egypt
ar_IN: Arabic - India	ar_IQ: Arabic - Iraq
ar_JO: Arabic - Jordan	ar_KW: Arabic - Kuwait
ar_LB: Arabic - Lebanon	ar_LY: Arabic - Libya
ar_MA: Arabic - Morocco	ar_OM: Arabic - Oman
ar_QA: Arabic - Qatar	ar_SA: Arabic - Saudi Arabia
ar_SD: Arabic - Sudan	ar_SY: Arabic - Syria
ar_TN: Arabic - Tunisia	ar_YE: Arabic - Yemen
be_BY: Belarusian - Belarus	bg_BG: Bulgarian - Bulgaria
ca_ES: Catalan - Spain	cs_CZ: Czech - Czech Republic
da_DK: Danish - Denmark	de_AT: German - Austria
de_BE: German - Belgium	de_CH: German - Switzerland
de_DE: German - Germany	de_LU: German - Luxembourg
el_GR: Greek - Greece	en_AU: English - Australia
en_CA: English - Canada	en_GB: English - United Kingdom
en_IN: English - India	en_NZ: English - New Zealand
en_PH: English - Philippines	en_US: English - United States
en_ZA: English - South Africa	en_ZW: English - Zimbabwe
es_AR: Spanish - Argentina	es_BO: Spanish - Bolivia
es_CL: Spanish - Chile	es_CO: Spanish - Columbia
es_CR: Spanish - Costa Rica	es_DO: Spanish - Dominican Republic
es_EC: Spanish - Ecuador	es_ES: Spanish - Spain
es_GT: Spanish - Guatemala	es_HN: Spanish - Honduras
es_MX: Spanish - Mexico	es_NI: Spanish - Nicaragua
es_PA: Spanish - Panama	es_PE: Spanish - Peru
es_PR: Spanish - Puerto Rico	es_PY: Spanish - Paraguay
es_SV: Spanish - El Salvador	es_US: Spanish - United States
es_UY: Spanish - Uruguay	es_VE: Spanish - Venezuela
et_EE: Estonian - Estonia	eu_ES: Basque - Basque
fi_FI: Finnish - Finland	fo_FO: Faroese - Faroe Islands
fr_BE: French - Belgium	fr_CA: French - Canada
fr_CH: French - Switzerland	fr_FR: French - France
fr_LU: French - Luxembourg	gl_ES: Galician - Spain
gu_IN: Gujarati - India	he_IL: Hebrew - Israel
hi_IN: Hindi - India	hr_HR: Croatian - Croatia
hu_HU: Hungarian - Hungary	id_ID: Indonesian - Indonesia
is_IS: Icelandic - Iceland	it_CH: Italian - Switzerland
it_IT: Italian - Italy	ja_JP: Japanese - Japan
ko_KR: Korean - Republic of Korea	lt_LT: Lithuanian - Lithuania

lv_LV: Latvian - Latvia	mk_MK: Macedonian - FYROM
mn_MN: Mongolia - Mongolian	ms_MY: Malay - Malaysia
nb_NO: Norwegian(Bokmål) - Norway	nl_BE: Dutch - Belgium
nl_NL: Dutch - The Netherlands	no_NO: Norwegian - Norway
pl_PL: Polish - Poland	pt_BR: Portugese - Brazil
pt_PT: Portugese - Portugal	rm_CH: Romansh - Switzerland
ro_RO: Romanian - Romania	ru_RU: Russian - Russia
ru_UA: Russian - Ukraine	sk_SK: Slovak - Slovakia
sl_SI: Slovenian - Slovenia	sq_AL: Albanian - Albania
sr_RS: Serbian - Yugoslavia	sv_FI: Swedish - Finland
sv_SE: Swedish - Sweden	ta_IN: Tamil - India
te_IN: Telugu - India	th_TH: Thai - Thailand
tr_TR: Turkish - Turkey	uk_UA: Ukrainian - Ukraine
ur_PK: Urdu - Pakistan	vi_VN: Vietnamese - Viet Nam
zh_CN: Chinese - China	zh_HK: Chinese - Hong Kong
zh_TW: Chinese - Taiwan Province of China	

Chapter 11 Data Types

Table of Contents

MySQL supports a number of SQL data types in several categories: numeric types, date and time types, and string (character and byte) types. This chapter provides an overview of these data types, a more detailed description of the properties of the types in each category, and a summary of the data type storage requirements. The initial overview is intentionally brief. The more detailed descriptions later in the chapter should be consulted for additional information about particular data types, such as the permissible formats in which you can specify values.

MySQL also supports extensions for handling spatial data. For information about these data types, see Section 12.18, “Spatial Extensions”.

Data type descriptions use these conventions:

- *M* indicates the maximum display width for integer types. For floating-point and fixed-point types, *M* is the total number of digits that can be stored (the precision). For string types, *M* is the maximum length. The maximum permissible value of *M* depends on the data type.

- *D* applies to floating-point and fixed-point types and indicates the number of digits following the decimal point (the scale). The maximum possible value is 30, but should be no greater than *M*–2.
- *fsp* applies to the TIME, DATETIME, and TIMESTAMP types and represents fractional seconds precision; that is, the number of digits following the decimal point for fractional parts of seconds. The *fsp* value, if given, must be in the range 0 to 6. A value of 0 signifies that there is no fractional part. If omitted, the default precision is 0. (This differs from the standard SQL default of 6, for compatibility with previous MySQL versions.)
- Square brackets (“[” and “]”) indicate optional parts of type definitions.

11.1 Data Type Overview

11.1.1 Numeric Type Overview

A summary of the numeric data types follows. For additional information about properties and storage requirements of the numeric types, see Section 11.2, “Numeric Types”, and Section 11.6, “Data Type Storage Requirements”.

M indicates the maximum display width for integer types. The maximum display width is 255. Display width is unrelated to the range of values a type can contain, as described in Section 11.2, “Numeric Types”. For floating-point and fixed-point types, *M* is the total number of digits that can be stored.

If you specify ZEROFILL for a numeric column, MySQL automatically adds the UNSIGNED attribute to the column.

Numeric data types that permit the UNSIGNED attribute also permit SIGNED. However, these data types are signed by default, so the SIGNED attribute has no effect.

SERIAL is an alias for BIGINT UNSIGNED NOT NULL AUTO_INCREMENT UNIQUE.

SERIAL DEFAULT VALUE in the definition of an integer column is an alias for NOT NULL AUTO_INCREMENT UNIQUE.

> **Warning**
>
> When you use subtraction between integer values where one is of type UNSIGNED, the result is unsigned unless the NO_UNSIGNED_SUBTRACTION SQL mode is enabled. See Section 12.10, “Cast Functions and Operators”.

- BIT[(*M*)]

  A bit-field type. *M* indicates the number of bits per value, from 1 to 64. The default is 1 if *M* is omitted.

- TINYINT[(*M*)] [UNSIGNED] [ZEROFILL]

  A very small integer. The signed range is -128 to 127. The unsigned range is 0 to 255.

- BOOL, BOOLEAN

  These types are synonyms for TINYINT(1). A value of zero is considered false. Nonzero values are considered true:

```
mysql> SELECT IF(0, 'true', 'false');
+------------------------+
| IF(0, 'true', 'false') |
+------------------------+
| false                  |
+------------------------+

mysql> SELECT IF(1, 'true', 'false');
+------------------------+
| IF(1, 'true', 'false') |
+------------------------+
| true                   |
+------------------------+

mysql> SELECT IF(2, 'true', 'false');
+------------------------+
| IF(2, 'true', 'false') |
+------------------------+
| true                   |
+------------------------+
```

However, the values TRUE and FALSE are merely aliases for 1 and 0, respectively, as shown here:

```
mysql> SELECT IF(0 = FALSE, 'true', 'false');
+--------------------------------+
| IF(0 = FALSE, 'true', 'false') |
+--------------------------------+
| true                           |
+--------------------------------+

mysql> SELECT IF(1 = TRUE, 'true', 'false');
+-------------------------------+
| IF(1 = TRUE, 'true', 'false') |
+-------------------------------+
| true                          |
+-------------------------------+

mysql> SELECT IF(2 = TRUE, 'true', 'false');
+-------------------------------+
| IF(2 = TRUE, 'true', 'false') |
+-------------------------------+
| false                         |
+-------------------------------+

mysql> SELECT IF(2 = FALSE, 'true', 'false');
+--------------------------------+
| IF(2 = FALSE, 'true', 'false') |
+--------------------------------+
| false                          |
+--------------------------------+
```

The last two statements display the results shown because 2 is equal to neither 1 nor 0.

- SMALLINT[(*M*)] [UNSIGNED] [ZEROFILL]

  A small integer. The signed range is -32768 to 32767. The unsigned range is 0 to 65535.

- MEDIUMINT[(*M*)] [UNSIGNED] [ZEROFILL]

  A medium-sized integer. The signed range is -8388608 to 8388607. The unsigned range is 0 to 16777215.

- INT[(*M*)] [UNSIGNED] [ZEROFILL]

  A normal-size integer. The signed range is -2147483648 to 2147483647. The unsigned range is 0 to 4294967295.

- INTEGER[(*M*)] [UNSIGNED] [ZEROFILL]

  This type is a synonym for INT.

- BIGINT[(*M*)] [UNSIGNED] [ZEROFILL]

  A large integer. The signed range is -9223372036854775808 to 9223372036854775807. The unsigned range is 0 to 18446744073709551615.

  SERIAL is an alias for BIGINT UNSIGNED NOT NULL AUTO_INCREMENT UNIQUE.

  Some things you should be aware of with respect to BIGINT columns:

  - All arithmetic is done using signed BIGINT or DOUBLE values, so you should not use unsigned big integers larger than 9223372036854775807 (63 bits) except with bit functions! If you do that, some of the last digits in the result may be wrong because of rounding errors when converting a BIGINT value to a DOUBLE.

    MySQL can handle BIGINT in the following cases:

    - When using integers to store large unsigned values in a BIGINT column.
    - In MIN(*col_name*) or MAX(*col_name*), where *col_name* refers to a BIGINT column.
    - When using operators (+, -, *, and so on) where both operands are integers.

  - You can always store an exact integer value in a BIGINT column by storing it using a string. In this case, MySQL performs a string-to-number conversion that involves no intermediate double-precision representation.

  - The -, +, and * operators use BIGINT arithmetic when both operands are integer values. This means that if you multiply two big integers (or results from functions that return integers), you may get unexpected results when the result is larger than 9223372036854775807.

- DECIMAL[(*M*[,*D*])] [UNSIGNED] [ZEROFILL]

  A packed “exact” fixed-point number. *M* is the total number of digits (the precision) and *D* is the number of digits after the decimal point (the scale). The decimal point and (for negative numbers) the “-” sign are not counted in *M*. If *D* is 0, values have no decimal point or fractional part. The maximum number of digits (*M*) for DECIMAL is 65. The maximum number of supported decimals (*D*) is 30. If *D* is omitted, the default is 0. If *M* is omitted, the default is 10.

  UNSIGNED, if specified, disallows negative values.

  All basic calculations (+, -, *, /) with DECIMAL columns are done with a precision of 65 digits.

- DEC[(*M*[,*D*])] [UNSIGNED] [ZEROFILL], NUMERIC[(*M*[,*D*])] [UNSIGNED] [ZEROFILL], FIXED[(*M*[,*D*])] [UNSIGNED] [ZEROFILL]

  These types are synonyms for DECIMAL. The FIXED synonym is available for compatibility with other database systems.

- FLOAT[(*M*,*D*)] [UNSIGNED] [ZEROFILL]

  A small (single-precision) floating-point number. Permissible values are -3.402823466E+38 to -1.175494351E-38, 0, and 1.175494351E-38 to 3.402823466E+38. These are the theoretical limits, based on the IEEE standard. The actual range might be slightly smaller depending on your hardware or operating system.

M is the total number of digits and *D* is the number of digits following the decimal point. If *M* and *D* are omitted, values are stored to the limits permitted by the hardware. A single-precision floating-point number is accurate to approximately 7 decimal places.

`UNSIGNED`, if specified, disallows negative values.

Using `FLOAT` might give you some unexpected problems because all calculations in MySQL are done with double precision. See Section C.5.5.7, “Solving Problems with No Matching Rows”.

- `DOUBLE[(M,D)] [UNSIGNED] [ZEROFILL]`

  A normal-size (double-precision) floating-point number. Permissible values are `-1.7976931348623157E+308` to `-2.2250738585072014E-308`, `0`, and `2.2250738585072014E-308` to `1.7976931348623157E+308`. These are the theoretical limits, based on the IEEE standard. The actual range might be slightly smaller depending on your hardware or operating system.

  M is the total number of digits and *D* is the number of digits following the decimal point. If *M* and *D* are omitted, values are stored to the limits permitted by the hardware. A double-precision floating-point number is accurate to approximately 15 decimal places.

  `UNSIGNED`, if specified, disallows negative values.

- `DOUBLE PRECISION[(M,D)] [UNSIGNED] [ZEROFILL]`, `REAL[(M,D)] [UNSIGNED] [ZEROFILL]`

  These types are synonyms for `DOUBLE`. Exception: If the `REAL_AS_FLOAT` SQL mode is enabled, `REAL` is a synonym for `FLOAT` rather than `DOUBLE`.

- `FLOAT(p) [UNSIGNED] [ZEROFILL]`

  A floating-point number. *p* represents the precision in bits, but MySQL uses this value only to determine whether to use `FLOAT` or `DOUBLE` for the resulting data type. If *p* is from 0 to 24, the data type becomes `FLOAT` with no *M* or *D* values. If *p* is from 25 to 53, the data type becomes `DOUBLE` with no *M* or *D* values. The range of the resulting column is the same as for the single-precision `FLOAT` or double-precision `DOUBLE` data types described earlier in this section.

  `FLOAT(p)` syntax is provided for ODBC compatibility.

11.1.2 Date and Time Type Overview

A summary of the temporal data types follows. For additional information about properties and storage requirements of the temporal types, see Section 11.3, “Date and Time Types”, and Section 11.6, “Data Type Storage Requirements”. For descriptions of functions that operate on temporal values, see Section 12.7, “Date and Time Functions”.

For the `DATE` and `DATETIME` range descriptions, “supported” means that although earlier values might work, there is no guarantee.

MySQL permits fractional seconds for `TIME`, `DATETIME`, and `TIMESTAMP` values, with up to microseconds (6 digits) precision. To define a column that includes a fractional seconds part, use the syntax `type_name(fsp)`, where *type_name* is `TIME`, `DATETIME`, or `TIMESTAMP`, and *fsp* is the fractional seconds precision. For example:

```
CREATE TABLE t1 (t TIME(3), dt DATETIME(6));
```

The *`fsp`* value, if given, must be in the range 0 to 6. A value of 0 signifies that there is no fractional part. If omitted, the default precision is 0. (This differs from the standard SQL default of 6, for compatibility with previous MySQL versions.)

MySQL 5.6.5 introduces expanded automatic initialization and updating of temporal types. Any `TIMESTAMP` column in a table can have these properties, rather than at most one column per table. In addition, these properties are now available for `DATETIME` columns.

The `YEAR(2)` data type has certain issues that you should consider before choosing to use it. As of MySQL 5.6.6, `YEAR(2)` is deprecated. `YEAR(2)` columns in existing tables are treated as before, but `YEAR(2)` in new or altered tables are converted to `YEAR(4)`. For more information, see Section 11.3.4, “`YEAR(2)` Limitations and Migrating to `YEAR(4)`”.

- `DATE`

  A date. The supported range is `'1000-01-01'` to `'9999-12-31'`. MySQL displays `DATE` values in `'YYYY-MM-DD'` format, but permits assignment of values to `DATE` columns using either strings or numbers.

- `DATETIME[(`*`fsp`*`)]`

  A date and time combination. The supported range is `'1000-01-01 00:00:00.000000'` to `'9999-12-31 23:59:59.999999'`. MySQL displays `DATETIME` values in `'YYYY-MM-DD HH:MM:SS[.fraction]'` format, but permits assignment of values to `DATETIME` columns using either strings or numbers.

  An optional *`fsp`* value in the range from 0 to 6 may be given to specify fractional seconds precision. A value of 0 signifies that there is no fractional part. If omitted, the default precision is 0.

  Automatic initialization and updating to the current date and time for `DATETIME` columns can be specified using `DEFAULT` and `ON UPDATE` column definition clauses, as described in Section 11.3.5, “Automatic Initialization and Updating for `TIMESTAMP` and `DATETIME`”.

- `TIMESTAMP[(`*`fsp`*`)]`

  A timestamp. The range is `'1970-01-01 00:00:01.000000'` UTC to `'2038-01-19 03:14:07.999999'` UTC. `TIMESTAMP` values are stored as the number of seconds since the epoch (`'1970-01-01 00:00:00'` UTC). A `TIMESTAMP` cannot represent the value `'1970-01-01 00:00:00'` because that is equivalent to 0 seconds from the epoch and the value 0 is reserved for representing `'0000-00-00 00:00:00'`, the “zero” `TIMESTAMP` value.

  An optional *`fsp`* value in the range from 0 to 6 may be given to specify fractional seconds precision. A value of 0 signifies that there is no fractional part. If omitted, the default precision is 0.

  The way the server handles `TIMESTAMP` definitions depends on the value of the `explicit_defaults_for_timestamp` system variable (see Section 5.1.4, “Server System Variables”). By default, `explicit_defaults_for_timestamp` is disabled and the server handles `TIMESTAMP` as follows:

  Unless specified otherwise, the first `TIMESTAMP` column in a table is defined to be automatically set to the date and time of the most recent modification if not explicitly assigned a value. This makes `TIMESTAMP` useful for recording the timestamp of an `INSERT` or `UPDATE` operation. You can also set any `TIMESTAMP` column to the current date and time by assigning it a `NULL` value, unless it has been defined with the `NULL` attribute to permit `NULL` values.

Automatic initialization and updating to the current date and time can be specified using DEFAULT CURRENT_TIMESTAMP and ON UPDATE CURRENT_TIMESTAMP column definition clauses. By default, the first TIMESTAMP column has these properties, as previously noted. As of MySQL 5.6.5, any TIMESTAMP column in a table can be defined to have these properties. Before 5.6.5, at most one TIMESTAMP column per table can have them, but it is possible to suppress them for the first column and instead assign them to a different TIMESTAMP column. See Section 11.3.5, “Automatic Initialization and Updating for TIMESTAMP and DATETIME”.

If explicit_defaults_for_timestamp is enabled, there is no automatic assignment of the DEFAULT CURRENT_TIMESTAMP or ON UPDATE CURRENT_TIMESTAMP attributes to any TIMESTAMP column. They must be included explicitly in the column definition. Also, any TIMESTAMP not explicitly declared as NOT NULL permits NULL values.

- TIME[(*fsp*)]

  A time. The range is '-838:59:59.000000' to '838:59:59.000000'. MySQL displays TIME values in 'HH:MM:SS[.fraction]' format, but permits assignment of values to TIME columns using either strings or numbers.

  An optional *fsp* value in the range from 0 to 6 may be given to specify fractional seconds precision. A value of 0 signifies that there is no fractional part. If omitted, the default precision is 0.

- YEAR[(2|4)]

  A year in two-digit or four-digit format. The default is four-digit format. YEAR(2) or YEAR(4) differ in display format, but have the same range of values. In four-digit format, values display as 1901 to 2155, and 0000. In two-digit format, values display as 70 to 69, representing years from 1970 to 2069. MySQL displays YEAR values in YYYY or YYformat, but permits assignment of values to YEAR columns using either strings or numbers.

  > **Note**
  >
  > The YEAR(2) data type has certain issues that you should consider before choosing to use it. As of MySQL 5.6.6, YEAR(2) is deprecated. YEAR(2) columns in existing tables are treated as before, but YEAR(2) in new or altered tables are converted to YEAR(4). For more information, see Section 11.3.4, “YEAR(2) Limitations and Migrating to YEAR(4)”.

  For additional information about YEAR display format and interpretation of input values, see Section 11.3.3, “The YEAR Type”.

The SUM() and AVG() aggregate functions do not work with temporal values. (They convert the values to numbers, losing everything after the first nonnumeric character.) To work around this problem, convert to numeric units, perform the aggregate operation, and convert back to a temporal value. Examples:

```
SELECT SEC_TO_TIME(SUM(TIME_TO_SEC(time_col))) FROM tbl_name;
SELECT FROM_DAYS(SUM(TO_DAYS(date_col))) FROM tbl_name;
```

> **Note**
>
> The MySQL server can be run with the MAXDB SQL mode enabled. In this case, TIMESTAMP is identical with DATETIME. If this mode is enabled at the time that a table is created, TIMESTAMP columns are created as DATETIME columns. As a result, such columns use DATETIME display format, have the same range of values,

and there is no automatic initialization or updating to the current date and time. See Section 5.1.7, “Server SQL Modes”.

11.1.3 String Type Overview

A summary of the string data types follows. For additional information about properties and storage requirements of the string types, see Section 11.4, “String Types”, and Section 11.6, “Data Type Storage Requirements”.

In some cases, MySQL may change a string column to a type different from that given in a `CREATE TABLE` or `ALTER TABLE` statement. See Section 13.1.14.3, “Silent Column Specification Changes”.

MySQL interprets length specifications in character column definitions in character units. This applies to `CHAR`, `VARCHAR`, and the `TEXT` types.

Column definitions for many string data types can include attributes that specify the character set or collation of the column. These attributes apply to the `CHAR`, `VARCHAR`, the `TEXT` types, `ENUM`, and `SET` data types:

- The `CHARACTER SET` attribute specifies the character set, and the `COLLATE` attribute specifies a collation for the character set. For example:

  ```
  CREATE TABLE t
  (
      c1 VARCHAR(20) CHARACTER SET utf8,
      c2 TEXT CHARACTER SET latin1 COLLATE latin1_general_cs
  );
  ```

  This table definition creates a column named `c1` that has a character set of `utf8` with the default collation for that character set, and a column named `c2` that has a character set of `latin1` and a case-sensitive collation.

  The rules for assigning the character set and collation when either or both of the `CHARACTER SET` and `COLLATE` attributes are missing are described in Section 10.1.3.4, “Column Character Set and Collation”.

  `CHARSET` is a synonym for `CHARACTER SET`.

- Specifying the `CHARACTER SET binary` attribute for a character data type causes the column to be created as the corresponding binary data type: `CHAR` becomes `BINARY`, `VARCHAR` becomes `VARBINARY`, and `TEXT` becomes `BLOB`. For the `ENUM` and `SET` data types, this does not occur; they are created as declared. Suppose that you specify a table using this definition:

  ```
  CREATE TABLE t
  (
    c1 VARCHAR(10) CHARACTER SET binary,
    c2 TEXT CHARACTER SET binary,
    c3 ENUM('a','b','c') CHARACTER SET binary
  );
  ```

  The resulting table has this definition:

  ```
  CREATE TABLE t
  (
    c1 VARBINARY(10),
    c2 BLOB,
    c3 ENUM('a','b','c') CHARACTER SET binary
  ```

```
);
```

- The `ASCII` attribute is shorthand for `CHARACTER SET latin1`.
- The `UNICODE` attribute is shorthand for `CHARACTER SET ucs2`.
- The `BINARY` attribute is shorthand for specifying the binary collation of the column character set. In this case, sorting and comparison are based on numeric character values.

Character column sorting and comparison are based on the character set assigned to the column. For the `CHAR`, `VARCHAR`, `TEXT`, `ENUM`, and `SET` data types, you can declare a column with a binary collation or the `BINARY` attribute to cause sorting and comparison to use the underlying character code values rather than a lexical ordering.

Section 10.1, “Character Set Support”, provides additional information about use of character sets in MySQL.

- `[NATIONAL] CHAR[(M)] [CHARACTER SET charset_name] [COLLATE collation_name]`

  A fixed-length string that is always right-padded with spaces to the specified length when stored. *`M`* represents the column length in characters. The range of *`M`* is 0 to 255. If *`M`* is omitted, the length is 1.

  > **Note**
  >
  > Trailing spaces are removed when `CHAR` values are retrieved unless the `PAD_CHAR_TO_FULL_LENGTH` SQL mode is enabled.

  `CHAR` is shorthand for `CHARACTER`. `NATIONAL CHAR` (or its equivalent short form, `NCHAR`) is the standard SQL way to define that a `CHAR` column should use some predefined character set. MySQL 4.1 and up uses `utf8` as this predefined character set. Section 10.1.3.6, “National Character Set”.

  The `CHAR BYTE` data type is an alias for the `BINARY` data type. This is a compatibility feature.

  MySQL permits you to create a column of type `CHAR(0)`. This is useful primarily when you have to be compliant with old applications that depend on the existence of a column but that do not actually use its value. `CHAR(0)` is also quite nice when you need a column that can take only two values: A column that is defined as `CHAR(0) NULL` occupies only one bit and can take only the values `NULL` and `''` (the empty string).

- `[NATIONAL] VARCHAR(M) [CHARACTER SET charset_name] [COLLATE collation_name]`

  A variable-length string. *`M`* represents the maximum column length in characters. The range of *`M`* is 0 to 65,535. The effective maximum length of a `VARCHAR` is subject to the maximum row size (65,535 bytes, which is shared among all columns) and the character set used. For example, `utf8` characters can require up to three bytes per character, so a `VARCHAR` column that uses the `utf8` character set can be declared to be a maximum of 21,844 characters. See Section E.10.4, “Limits on Table Column Count and Row Size”.

  MySQL stores `VARCHAR` values as a 1-byte or 2-byte length prefix plus data. The length prefix indicates the number of bytes in the value. A `VARCHAR` column uses one length byte if values require no more than 255 bytes, two length bytes if values may require more than 255 bytes.

  > **Note**
  >
  > MySQL 5.7 follows the standard SQL specification, and does *not* remove trailing spaces from `VARCHAR` values.

VARCHAR is shorthand for CHARACTER VARYING. NATIONAL VARCHAR is the standard SQL way to define that a VARCHAR column should use some predefined character set. MySQL 4.1 and up uses utf8 as this predefined character set. Section 10.1.3.6, “National Character Set”. NVARCHAR is shorthand for NATIONAL VARCHAR.

- BINARY(*M*)

  The BINARY type is similar to the CHAR type, but stores binary byte strings rather than nonbinary character strings. *M* represents the column length in bytes.

- VARBINARY(*M*)

  The VARBINARY type is similar to the VARCHAR type, but stores binary byte strings rather than nonbinary character strings. *M* represents the maximum column length in bytes.

- TINYBLOB

  A BLOB column with a maximum length of 255 (2^8 – 1) bytes. Each TINYBLOB value is stored using a 1-byte length prefix that indicates the number of bytes in the value.

- TINYTEXT [CHARACTER SET *charset_name*] [COLLATE *collation_name*]

  A TEXT column with a maximum length of 255 (2^8 – 1) characters. The effective maximum length is less if the value contains multi-byte characters. Each TINYTEXT value is stored using a 1-byte length prefix that indicates the number of bytes in the value.

- BLOB[(*M*)]

  A BLOB column with a maximum length of 65,535 (2^{16} – 1) bytes. Each BLOB value is stored using a 2-byte length prefix that indicates the number of bytes in the value.

  An optional length *M* can be given for this type. If this is done, MySQL creates the column as the smallest BLOB type large enough to hold values *M* bytes long.

- TEXT[(*M*)] [CHARACTER SET *charset_name*] [COLLATE *collation_name*]

  A TEXT column with a maximum length of 65,535 (2^{16} – 1) characters. The effective maximum length is less if the value contains multi-byte characters. Each TEXT value is stored using a 2-byte length prefix that indicates the number of bytes in the value.

  An optional length *M* can be given for this type. If this is done, MySQL creates the column as the smallest TEXT type large enough to hold values *M* characters long.

- MEDIUMBLOB

  A BLOB column with a maximum length of 16,777,215 (2^{24} – 1) bytes. Each MEDIUMBLOB value is stored using a 3-byte length prefix that indicates the number of bytes in the value.

- MEDIUMTEXT [CHARACTER SET *charset_name*] [COLLATE *collation_name*]

  A TEXT column with a maximum length of 16,777,215 (2^{24} – 1) characters. The effective maximum length is less if the value contains multi-byte characters. Each MEDIUMTEXT value is stored using a 3-byte length prefix that indicates the number of bytes in the value.

- LONGBLOB

  A BLOB column with a maximum length of 4,294,967,295 or 4GB (2^{32} – 1) bytes. The effective maximum length of LONGBLOB columns depends on the configured maximum packet size in the client/server protocol and available memory. Each LONGBLOB value is stored using a 4-byte length prefix that indicates the number of bytes in the value.

- LONGTEXT [CHARACTER SET *charset_name*] [COLLATE *collation_name*]

  A TEXT column with a maximum length of 4,294,967,295 or 4GB (2^{32} – 1) characters. The effective maximum length is less if the value contains multi-byte characters. The effective maximum length of LONGTEXT columns also depends on the configured maximum packet size in the client/server protocol and available memory. Each LONGTEXT value is stored using a 4-byte length prefix that indicates the number of bytes in the value.

- ENUM('*value1*','*value2*',...) [CHARACTER SET *charset_name*] [COLLATE *collation_name*]

  An enumeration. A string object that can have only one value, chosen from the list of values '*value1*', '*value2*', ..., NULL or the special '' error value. ENUM values are represented internally as integers.

  An ENUM column can have a maximum of 65,535 distinct elements. (The practical limit is less than 3000.) A table can have no more than 255 unique element list definitions among its ENUM and SET columns considered as a group. For more information on these limits, see Section E.10.5, “Limits Imposed by .frm File Structure”.

- SET('*value1*','*value2*',...) [CHARACTER SET *charset_name*] [COLLATE *collation_name*]

  A set. A string object that can have zero or more values, each of which must be chosen from the list of values '*value1*', '*value2*', ... SET values are represented internally as integers.

  A SET column can have a maximum of 64 distinct members. A table can have no more than 255 unique element list definitions among its ENUM and SET columns considered as a group. For more information on this limit, see Section E.10.5, “Limits Imposed by .frm File Structure”.

11.2 Numeric Types

MySQL supports all standard SQL numeric data types. These types include the exact numeric data types (INTEGER, SMALLINT, DECIMAL, and NUMERIC), as well as the approximate numeric data types (FLOAT, REAL, and DOUBLE PRECISION). The keyword INT is a synonym for INTEGER, and the keywords DEC and FIXED are synonyms for DECIMAL. MySQL treats DOUBLE as a synonym for DOUBLE PRECISION (a nonstandard extension). MySQL also treats REAL as a synonym for DOUBLE PRECISION (a nonstandard variation), unless the REAL_AS_FLOAT SQL mode is enabled.

The BIT data type stores bit-field values and is supported for MyISAM, MEMORY, and InnoDB .

For information about how MySQL handles assignment of out-of-range values to columns and overflow during expression evaluation, see Section 11.2.6, “Out-of-Range and Overflow Handling”.

For information about numeric type storage requirements, see Section 11.6, “Data Type Storage Requirements”.

The data type used for the result of a calculation on numeric operands depends on the types of the operands and the operations performed on them. For more information, see Section 12.6.1, “Arithmetic Operators”.

11.2.1 Integer Types (Exact Value) - INTEGER, INT, SMALLINT, TINYINT, MEDIUMINT, BIGINT

MySQL supports the SQL standard integer types INTEGER (or INT) and SMALLINT. As an extension to the standard, MySQL also supports the integer types TINYINT, MEDIUMINT, and BIGINT. The following table shows the required storage and range for each integer type.

Type	Storage	Minimum Value	Maximum Value
	(Bytes)	(Signed/Unsigned)	(Signed/Unsigned)
TINYINT	1	-128	127
		0	255
SMALLINT	2	-32768	32767
		0	65535
MEDIUMINT	3	-8388608	8388607
		0	16777215
INT	4	-2147483648	2147483647
		0	4294967295
BIGINT	8	-9223372036854775808	9223372036854775807
		0	18446744073709551615

11.2.2 Fixed-Point Types (Exact Value) - DECIMAL, NUMERIC

The DECIMAL and NUMERIC types store exact numeric data values. These types are used when it is important to preserve exact precision, for example with monetary data. In MySQL, NUMERIC is implemented as DECIMAL, so the following remarks about DECIMAL apply equally to NUMERIC.

MySQL 5.7 stores DECIMAL values in binary format. See Section 12.19, “Precision Math”.

In a DECIMAL column declaration, the precision and scale can be (and usually is) specified; for example:

```
salary DECIMAL(5,2)
```

In this example, 5 is the precision and 2 is the scale. The precision represents the number of significant digits that are stored for values, and the scale represents the number of digits that can be stored following the decimal point.

Standard SQL requires that DECIMAL(5,2) be able to store any value with five digits and two decimals, so values that can be stored in the salary column range from -999.99 to 999.99.

In standard SQL, the syntax DECIMAL(*M*) is equivalent to DECIMAL(*M*,0). Similarly, the syntax DECIMAL is equivalent to DECIMAL(*M*,0), where the implementation is permitted to decide the value of *M*. MySQL supports both of these variant forms of DECIMAL syntax. The default value of *M* is 10.

If the scale is 0, DECIMAL values contain no decimal point or fractional part.

The maximum number of digits for DECIMAL is 65, but the actual range for a given DECIMAL column can be constrained by the precision or scale for a given column. When such a column is assigned a value with more digits following the decimal point than are permitted by the specified scale, the value is converted to that scale. (The precise behavior is operating system-specific, but generally the effect is truncation to the permissible number of digits.)

11.2.3 Floating-Point Types (Approximate Value) - FLOAT, DOUBLE

The FLOAT and DOUBLE types represent approximate numeric data values. MySQL uses four bytes for single-precision values and eight bytes for double-precision values.

For FLOAT, the SQL standard permits an optional specification of the precision (but not the range of the exponent) in bits following the keyword FLOAT in parentheses. MySQL also supports this optional precision specification, but the precision value is used only to determine storage size. A precision from 0 to 23 results in a 4-byte single-precision FLOAT column. A precision from 24 to 53 results in an 8-byte double-precision DOUBLE column.

MySQL permits a nonstandard syntax: FLOAT(*M*,*D*) or REAL(*M*,*D*) or DOUBLE PRECISION(*M*,*D*). Here, “(*M*,*D*)” means than values can be stored with up to *M* digits in total, of which *D* digits may be after the decimal point. For example, a column defined as FLOAT(7,4) will look like -999.9999 when displayed. MySQL performs rounding when storing values, so if you insert 999.00009 into a FLOAT(7,4) column, the approximate result is 999.0001.

Because floating-point values are approximate and not stored as exact values, attempts to treat them as exact in comparisons may lead to problems. They are also subject to platform or implementation dependencies. For more information, see Section C.5.5.8, “Problems with Floating-Point Values”

For maximum portability, code requiring storage of approximate numeric data values should use FLOAT or DOUBLE PRECISION with no specification of precision or number of digits.

11.2.4 Bit-Value Type - BIT

The BIT data type is used to store bit-field values. A type of BIT(*M*) enables storage of *M*-bit values. *M* can range from 1 to 64.

To specify bit values, b'*value*' notation can be used. *value* is a binary value written using zeros and ones. For example, b'111' and b'10000000' represent 7 and 128, respectively. See Section 9.1.6, “Bit-Field Literals”.

If you assign a value to a BIT(*M*) column that is less than *M* bits long, the value is padded on the left with zeros. For example, assigning a value of b'101' to a BIT(6) column is, in effect, the same as assigning b'000101'.

11.2.5 Numeric Type Attributes

MySQL supports an extension for optionally specifying the display width of integer data types in parentheses following the base keyword for the type. For example, INT(4) specifies an INT with a display width of four digits. This optional display width may be used by applications to display integer values having a width less than the width specified for the column by left-padding them with spaces. (That is, this width is present in the metadata returned with result sets. Whether it is used or not is up to the application.)

The display width does *not* constrain the range of values that can be stored in the column. Nor does it prevent values wider than the column display width from being displayed correctly. For example, a column specified as SMALLINT(3) has the usual SMALLINT range of -32768 to 32767, and values outside the range permitted by three digits are displayed in full using more than three digits.

When used in conjunction with the optional (nonstandard) attribute ZEROFILL, the default padding of spaces is replaced with zeros. For example, for a column declared as INT(4) ZEROFILL, a value of 5 is retrieved as 0005.

Note

The ZEROFILL attribute is ignored when a column is involved in expressions or UNION queries.

If you store values larger than the display width in an integer column that has the ZEROFILL attribute, you may experience problems when MySQL generates temporary tables for some complicated joins. In these cases, MySQL assumes that the data values fit within the column display width.

All integer types can have an optional (nonstandard) attribute UNSIGNED. Unsigned type can be used to permit only nonnegative numbers in a column or when you need a larger upper numeric range for the column. For example, if an INT column is UNSIGNED, the size of the column's range is the same but its endpoints shift from -2147483648 and 2147483647 up to 0 and 4294967295.

Floating-point and fixed-point types also can be UNSIGNED. As with integer types, this attribute prevents negative values from being stored in the column. Unlike the integer types, the upper range of column values remains the same.

If you specify ZEROFILL for a numeric column, MySQL automatically adds the UNSIGNED attribute to the column.

Integer or floating-point data types can have the additional attribute AUTO_INCREMENT. When you insert a value of NULL (recommended) or 0 into an indexed AUTO_INCREMENT column, the column is set to the next sequence value. Typically this is *value*+1, where *value* is the largest value for the column currently in the table. AUTO_INCREMENT sequences begin with 1. (Inserting NULL to generate AUTO_INCREMENT values requires that the column be declared NOT NULL. If the column is declared NULL, inserting NULL stores a NULL.)

In MySQL 5.7, negative values for AUTO_INCREMENT columns are not supported.

11.2.6 Out-of-Range and Overflow Handling

When MySQL stores a value in a numeric column that is outside the permissible range of the column data type, the result depends on the SQL mode in effect at the time:

- If strict SQL mode is enabled, MySQL rejects the out-of-range value with an error, and the insert fails, in accordance with the SQL standard.
- If no restrictive modes are enabled, MySQL clips the value to the appropriate endpoint of the range and stores the resulting value instead.

  When an out-of-range value is assigned to an integer column, MySQL stores the value representing the corresponding endpoint of the column data type range. If you store 256 into a TINYINT or TINYINT UNSIGNED column, MySQL stores 127 or 255, respectively.

  When a floating-point or fixed-point column is assigned a value that exceeds the range implied by the specified (or default) precision and scale, MySQL stores the value representing the corresponding endpoint of that range.

Column-assignment conversions that occur due to clipping when MySQL is not operating in strict mode are reported as warnings for ALTER TABLE, LOAD DATA INFILE, UPDATE, and multiple-row INSERT statements. In strict mode, these statements fail, and some or all the values will not be inserted or changed, depending on whether the table is a transactional table and other factors. For details, see Section 5.1.7, “Server SQL Modes”.

In MySQL 5.7, overflow during numeric expression evaluation results in an error. For example, the largest signed BIGINT value is 9223372036854775807, so the following expression produces an error:

```
mysql> SELECT 9223372036854775807 + 1;
ERROR 1690 (22003): BIGINT value is out of range in '(9223372036854775807 + 1)'
```

To enable the operation to succeed in this case, convert the value to unsigned;

```
mysql> SELECT CAST(9223372036854775807 AS UNSIGNED) + 1;
+-------------------------------------------+
| CAST(9223372036854775807 AS UNSIGNED) + 1 |
+-------------------------------------------+
|                       9223372036854775808 |
+-------------------------------------------+
```

Whether overflow occurs depends on the range of the operands, so another way to handle the preceding expression is to use exact-value arithmetic because DECIMAL values have a larger range than integers:

```
mysql> SELECT 9223372036854775807.0 + 1;
+---------------------------+
| 9223372036854775807.0 + 1 |
+---------------------------+
|     9223372036854775808.0 |
+---------------------------+
```

Subtraction between integer values, where one is of type UNSIGNED, produces an unsigned result by default. Prior to MySQL 5.5.5, if the result would otherwise have been negative, it becomes the maximum integer value:

```
mysql> SET sql_mode = '';
mysql> SELECT CAST(0 AS UNSIGNED) - 1;
+-------------------------+
| CAST(0 AS UNSIGNED) - 1 |
+-------------------------+
|    18446744073709551615 |
+-------------------------+
```

As of MySQL 5.5.5, if the result would otherwise have been negative, an error results:

```
mysql> SET sql_mode = '';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT CAST(0 AS UNSIGNED) - 1;
ERROR 1690 (22003): BIGINT UNSIGNED value is out of range in '(cast(0 as unsigned) - 1)'
```

If the NO_UNSIGNED_SUBTRACTION SQL mode is enabled, the result is negative:

```
mysql> SET sql_mode = 'NO_UNSIGNED_SUBTRACTION';
mysql> SELECT CAST(0 AS UNSIGNED) - 1;
+-------------------------+
| CAST(0 AS UNSIGNED) - 1 |
+-------------------------+
|                      -1 |
+-------------------------+
```

If the result of such an operation is used to update an UNSIGNED integer column, the result is clipped to the maximum value for the column type, or clipped to 0 if NO_UNSIGNED_SUBTRACTION is enabled. If strict SQL mode is enabled, an error occurs and the column remains unchanged.

11.3 Date and Time Types

The date and time types for representing temporal values are DATE, TIME, DATETIME, TIMESTAMP, and YEAR. Each temporal type has a range of valid values, as well as a “zero” value that may be used when you specify an invalid value that MySQL cannot represent. The TIMESTAMP type has special automatic updating behavior, described later. For temporal type storage requirements, see Section 11.6, “Data Type Storage Requirements”.

Keep in mind these general considerations when working with date and time types:

- MySQL retrieves values for a given date or time type in a standard output format, but it attempts to interpret a variety of formats for input values that you supply (for example, when you specify a value to be assigned to or compared to a date or time type). For a description of the permitted formats for date and time types, see Section 9.1.3, “Date and Time Literals”. It is expected that you supply valid values. Unpredictable results may occur if you use values in other formats.

- Although MySQL tries to interpret values in several formats, date parts must always be given in year-month-day order (for example, '98-09-04'), rather than in the month-day-year or day-month-year orders commonly used elsewhere (for example, '09-04-98', '04-09-98').

- Dates containing two-digit year values are ambiguous because the century is unknown. MySQL interprets two-digit year values using these rules:

  - Year values in the range 70-99 are converted to 1970-1999.

  - Year values in the range 00-69 are converted to 2000-2069.

  See also Section 11.3.8, “Two-Digit Years in Dates”.

- Conversion of values from one temporal type to another occurs according to the rules in Section 11.3.7, “Conversion Between Date and Time Types”.

- MySQL automatically converts a date or time value to a number if the value is used in a numeric context and vice versa.

- By default, when MySQL encounters a value for a date or time type that is out of range or otherwise invalid for the type, it converts the value to the “zero” value for that type. The exception is that out-of-range TIME values are clipped to the appropriate endpoint of the TIME range.

- By setting the SQL mode to the appropriate value, you can specify more exactly what kind of dates you want MySQL to support. (See Section 5.1.7, “Server SQL Modes”.) You can get MySQL to accept certain dates, such as '2009-11-31', by enabling the ALLOW_INVALID_DATES SQL mode. This is useful when you want to store a “possibly wrong” value which the user has specified (for example, in a web form) in the database for future processing. Under this mode, MySQL verifies only that the month is in the range from 1 to 12 and that the day is in the range from 1 to 31.

- MySQL permits you to store dates where the day or month and day are zero in a DATE or DATETIME column. This is useful for applications that need to store birthdates for which you may not know the exact date. In this case, you simply store the date as '2009-00-00' or '2009-01-00'. If you store dates such as these, you should not expect to get correct results for functions such as DATE_SUB() or DATE_ADD() that require complete dates. To disallow zero month or day parts in dates, enable strict SQL mode (as of MySQL 5.7.4) or the NO_ZERO_IN_DATE mode (before MySQL 5.7.4).

- MySQL permits you to store a “zero” value of '0000-00-00' as a “dummy date.” This is in some cases more convenient than using NULL values, and uses less data and index space. To disallow '0000-00-00', enable strict SQL mode (as of MySQL 5.7.4) or the NO_ZERO_DATE mode (before MySQL 5.7.4).

- “Zero” date or time values used through Connector/ODBC are converted automatically to NULL because ODBC cannot handle such values.

The following table shows the format of the “zero” value for each type. The “zero” values are special, but you can store or refer to them explicitly using the values shown in the table. You can also do this using the values '0' or 0, which are easier to write. For temporal types that include a date part (DATE, DATETIME, and TIMESTAMP), use of these values produces warnings if strict SQL mode is enabled and IGNORE is used (as of MySQL 5.7.4) or the NO_ZERO_DATE mode is enabled (before MySQL 5.7.4).

Data Type	“Zero” Value
DATE	'0000-00-00'
TIME	'00:00:00'
DATETIME	'0000-00-00 00:00:00'
TIMESTAMP	'0000-00-00 00:00:00'
YEAR	0000

11.3.1 The DATE, DATETIME, and TIMESTAMP Types

The DATE, DATETIME, and TIMESTAMP types are related. This section describes their characteristics, how they are similar, and how they differ. MySQL recognizes DATE, DATETIME, and TIMESTAMP values in several formats, described in Section 9.1.3, “Date and Time Literals”. For the DATE and DATETIME range descriptions, “supported” means that although earlier values might work, there is no guarantee.

The DATE type is used for values with a date part but no time part. MySQL retrieves and displays DATE values in 'YYYY-MM-DD' format. The supported range is '1000-01-01' to '9999-12-31'.

The DATETIME type is used for values that contain both date and time parts. MySQL retrieves and displays DATETIME values in 'YYYY-MM-DD HH:MM:SS' format. The supported range is '1000-01-01 00:00:00' to '9999-12-31 23:59:59'.

The TIMESTAMP data type is used for values that contain both date and time parts. TIMESTAMP has a range of '1970-01-01 00:00:01' UTC to '2038-01-19 03:14:07' UTC.

A DATETIME or TIMESTAMP value can include a trailing fractional seconds part in up to microseconds (6 digits) precision. In particular, any fractional part in a value inserted into a DATETIME or TIMESTAMP column is stored rather than discarded. With the fractional part included, the format for these values is 'YYYY-MM-DD HH:MM:SS[.fraction]', the range for DATETIME values is '1000-01-01 00:00:00.000000' to '9999-12-31 23:59:59.999999', and the range for TIMESTAMP values is '1970-01-01 00:00:01.000000' to '2038-01-19 03:14:07.999999'. The fractional part should always be separated from the rest of the time by a decimal point; no other fractional seconds delimiter is recognized. For information about fractional seconds support in MySQL, see Section 11.3.6, “Fractional Seconds in Time Values”.

The TIMESTAMP and DATETIME data types offer automatic initialization and updating to the current date and time. For more information, see Section 11.3.5, “Automatic Initialization and Updating for TIMESTAMP and DATETIME”.

MySQL converts TIMESTAMP values from the current time zone to UTC for storage, and back from UTC to the current time zone for retrieval. (This does not occur for other types such as DATETIME.) By default, the current time zone for each connection is the server's time. The time zone can be set on a per-connection basis. As long as the time zone setting remains constant, you get back the same value you store. If you store a TIMESTAMP value, and then change the time zone and retrieve the value, the retrieved value is different from the value you stored. This occurs because the same time zone was not used for conversion in both directions. The current time zone is available as the value of the time_zone system variable. For more information, see Section 10.6, “MySQL Server Time Zone Support”.

Invalid DATE, DATETIME, or TIMESTAMP values are converted to the “zero” value of the appropriate type ('0000-00-00' or '0000-00-00 00:00:00').

Be aware of certain properties of date value interpretation in MySQL:

- MySQL permits a “relaxed” format for values specified as strings, in which any punctuation character may be used as the delimiter between date parts or time parts. In some cases, this syntax can be deceiving. For example, a value such as '10:11:12' might look like a time value because of the “:” delimiter, but is interpreted as the year '2010-11-12' if used in a date context. The value '10:45:15' is converted to '0000-00-00' because '45' is not a valid month.

  The only delimiter recognized between a date and time part and a fractional seconds part is the decimal point.

- The server requires that month and day values be valid, and not merely in the range 1 to 12 and 1 to 31, respectively. With strict mode disabled, invalid dates such as '2004-04-31' are converted to '0000-00-00' and a warning is generated. With strict mode enabled, invalid dates generate an error. To permit such dates, enable ALLOW_INVALID_DATES. See Section 5.1.7, “Server SQL Modes”, for more information.

- MySQL does not accept TIMESTAMP values that include a zero in the day or month column or values that are not a valid date. The sole exception to this rule is the special “zero” value '0000-00-00 00:00:00'.

- Dates containing two-digit year values are ambiguous because the century is unknown. MySQL interprets two-digit year values using these rules:

  - Year values in the range 00-69 are converted to 2000-2069.

  - Year values in the range 70-99 are converted to 1970-1999.

  See also Section 11.3.8, “Two-Digit Years in Dates”.

Note

The MySQL server can be run with the MAXDB SQL mode enabled. In this case, TIMESTAMP is identical with DATETIME. If this mode is enabled at the time that a table is created, TIMESTAMP columns are created as DATETIME columns. As a result, such columns use DATETIME display format, have the same range of values, and there is no automatic initialization or updating to the current date and time. See Section 5.1.7, “Server SQL Modes”.

11.3.2 The TIME Type

MySQL retrieves and displays TIME values in 'HH:MM:SS' format (or 'HHH:MM:SS' format for large hours values). TIME values may range from '-838:59:59' to '838:59:59'. The hours part may be so large because the TIME type can be used not only to represent a time of day (which must be less than 24 hours), but also elapsed time or a time interval between two events (which may be much greater than 24 hours, or even negative).

MySQL recognizes TIME values in several formats, some of which can include a trailing fractional seconds part in up to microseconds (6 digits) precision. See Section 9.1.3, “Date and Time Literals”. For information about fractional seconds support in MySQL, see Section 11.3.6, “Fractional Seconds in Time Values”. In particular, any fractional part in a value inserted into a TIME column is stored rather than discarded. With the fractional part included, the range for TIME values is '-838:59:59.000000' to '838:59:59.000000'.

Be careful about assigning abbreviated values to a TIME column. MySQL interprets abbreviated TIME values with colons as time of the day. That is, '11:12' means '11:12:00', not '00:11:12'. MySQL interprets abbreviated values without colons using the assumption that the two rightmost digits represent seconds (that is, as elapsed time rather than as time of day). For example, you might think of '1112' and 1112 as meaning '11:12:00' (12 minutes after 11 o'clock), but MySQL interprets them as '00:11:12' (11 minutes, 12 seconds). Similarly, '12' and 12 are interpreted as '00:00:12'.

The only delimiter recognized between a time part and a fractional seconds part is the decimal point.

By default, values that lie outside the TIME range but are otherwise valid are clipped to the closest endpoint of the range. For example, '-850:00:00' and '850:00:00' are converted to '-838:59:59' and '838:59:59'. Invalid TIME values are converted to '00:00:00'. Note that because '00:00:00' is itself a valid TIME value, there is no way to tell, from a value of '00:00:00' stored in a table, whether the original value was specified as '00:00:00' or whether it was invalid.

For more restrictive treatment of invalid TIME values, enable strict SQL mode to cause errors to occur. See Section 5.1.7, “Server SQL Modes”.

11.3.3 The YEAR Type

The YEAR type is a 1-byte type used to represent year values. It can be declared as YEAR(4) or YEAR(2) to specify a display width of four or two characters. The default is four characters if no width is given.

> **Note**
>
> The YEAR(2) data type has certain issues that you should consider before choosing to use it. Also, as of MySQL 5.6.6, YEAR(2) is deprecated. YEAR(2) columns in existing tables are treated as before, but YEAR(2) in new or altered tables are converted to YEAR(4). For more information, see Section 11.3.4, “YEAR(2) Limitations and Migrating to YEAR(4)”.

YEAR(4) and YEAR(2) differ in display format, but have the same range of values. For 4-digit format, MySQL displays YEAR values in YYYY format, with a range of 1901 to 2155, or 0000. For 2-digit format, MySQL displays only the last two (least significant) digits; for example, 70 (1970 or 2070) or 69 (2069).

You can specify input YEAR values in a variety of formats:

- As a 4-digit string in the range '1901' to '2155'.

- As a 4-digit number in the range 1901 to 2155.

- As a 1- or 2-digit string in the range '0' to '99'. Values in the ranges '0' to '69' and '70' to '99' are converted to YEAR values in the ranges 2000 to 2069 and 1970 to 1999.

- As a 1- or 2-digit number in the range 1 to 99. Values in the ranges 1 to 69 and 70 to 99 are converted to YEAR values in the ranges 2001 to 2069 and 1970 to 1999.

  Inserting a numeric 0 has a different effect for YEAR(2) and YEAR(4). For YEAR(2), the result has a display value of 00 and an internal value of 2000. For YEAR(4), the result has a display value of 0000 and an internal value of 0000. To specify zero for YEAR(4) and have it be interpreted as 2000, specify it as a string '0' or '00'.

- As the result of a function that returns a value that is acceptable in a YEAR context, such as NOW().

Invalid YEAR values are converted to 0000.

See also Section 11.3.8, “Two-Digit Years in Dates”.

11.3.4 YEAR(2) Limitations and Migrating to YEAR(4)

Although the internal range of values for YEAR(4) and YEAR(2) is the same (1901 to 2155, and 0000), the display width for YEAR(2) makes that type inherently ambiguous because displayed values indicate only the last two digits of the internal values. The result can be a loss of information under certain circumstances. For this reason, consider avoiding YEAR(2) throughout your applications and using YEAR(4) wherever you need a YEAR data type. This section describes problems that can occur when using YEAR(2) and provides information about migrating existing YEAR(2) columns to YEAR(4). Note that migration will become necessary at some point because support for YEAR data types with display values other than 4, most notably YEAR(2), is reduced as of MySQL 5.6.6 and will be removed entirely in a future release.

YEAR(2) Limitations

Issues with the YEAR(2) data type include ambiguity of displayed values, and possible loss of information when values are dumped and reloaded or converted to strings.

- Displayed YEAR(2) values can be ambiguous. It is possible for up to three YEAR(2) values that have different internal values to have the same displayed value, as the following example demonstrates:

```
mysql> CREATE TABLE t (y2 YEAR(2), y4 YEAR(4));
Query OK, 0 rows affected (0.01 sec)

mysql> INSERT INTO t (y2) VALUES(1912),(2012),(2112);
Query OK, 3 rows affected (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> UPDATE t SET y4 = y2;
Query OK, 3 rows affected (0.00 sec)
Rows matched: 3  Changed: 3  Warnings: 0

mysql> SELECT * FROM t;
+------+------+
| y2   | y4   |
+------+------+
|   12 | 1912 |
|   12 | 2012 |
|   12 | 2112 |
+------+------+
3 rows in set (0.00 sec)
```

- If you use mysqldump to dump the table created in the preceding item, the dump file represents all y2 values using the same 2-digit representation (12). If you reload the table from the dump file, all resulting rows have internal value 2012 and display value 12, thus losing the distinctions among them.

- Conversion of a YEAR(2) or YEAR(4) data value to string form uses the display width of the YEAR type. Suppose that YEAR(2) and YEAR(4) columns both contain the value 1970. Assigning each column to a string results in a value of '70' or '1970', respectively. That is, loss of information occurs for conversion from YEAR(2) to string.

- Values outside the range from 1970 to 2069 are stored incorrectly when inserted into a YEAR(2) column in a CSV table. For example, inserting 2111 results in a display value of 11 but an internal value of 2011.

To avoid these problems, use YEAR(4) rather than YEAR(2). Suggestions regarding migration strategies appear later in this section.

Reduced YEAR(2) Support in MySQL 5.6

As of MySQL 5.6.6, support for YEAR(2) is diminished:

- YEAR(2) in column definitions for new tables is converted (with a warning) to YEAR(4):

```
mysql> CREATE TABLE t1 (y YEAR(2));
Query OK, 0 rows affected, 1 warning (0.03 sec)

mysql> SHOW WARNINGS\G
*************************** 1. row ***************************
  Level: Warning
   Code: 1818
Message: YEAR(2) column type is deprecated. Creating YEAR(4) column instead.
1 row in set (0.00 sec)

mysql> SHOW CREATE TABLE t1\G
*************************** 1. row ***************************
       Table: t1
Create Table: CREATE TABLE `t1` (
  `y` year(4) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1
1 row in set (0.00 sec)
```

- YEAR(2) in existing tables remains as YEAR(2) and is processed in queries as in older versions of MySQL. However, several programs or statements convert YEAR(2) to YEAR(4) automatically:

  - ALTER TABLE statements that result in a table rebuild.
  - REPAIR TABLE (which CHECK TABLE recommends you use if it finds that a table contains YEAR(2) columns).
  - mysql_upgrade (which uses REPAIR TABLE).
  - Dumping with mysqldump and reloading the dump file. Unlike the conversions performed by the preceding three items, a dump and reload has the potential to change values.

  A MySQL upgrade usually involves at least one of the last two items. However, with respect to YEAR(2), you should avoid dumping and reloading; as noted, that can change values.

Migrating from YEAR(2) to YEAR(4)

Should you decide to convert YEAR(2) columns to YEAR(4), you can do so manually at any time without upgrading. Alternatively, you can upgrade to a version of MySQL with reduced support for YEAR(2) (MySQL 5.6.6 or later), then have MySQL convert YEAR(2) columns automatically. In the latter case, avoid upgrading by dumping and reloading your data because that can change data values. In addition, if you use replication, there are upgrade considerations you must take into account.

To convert YEAR(2) columns to YEAR(4) manually, use ALTER TABLE. Suppose that a table t1 has this definition:

```
CREATE TABLE t1 (ycol YEAR(2) NOT NULL DEFAULT '70');
```

Modify the column using ALTER TABLE as follows. Remember to include any column attributes such as NOT NULL or DEFAULT:

```
ALTER TABLE t1 MODIFY ycol YEAR(4) NOT NULL DEFAULT '1970';
```

The ALTER TABLE statement converts the table without changing YEAR(2) values. If the server is a replication master, the ALTER TABLE statement replicates to slaves and makes the corresponding table change on each one.

Another migration method is to perform a binary upgrade: Install MySQL without dumping and reloading your data. Then run mysql_upgrade, which uses REPAIR TABLE to convert YEAR(2) columns to YEAR(4) without changing data values. If the server is a replication master, the REPAIR TABLE statements replicate to slaves and make the corresponding table changes on each one, unless you invoke mysql_upgrade with the --skip-write-binlog option.

Upgrades to replication servers usually involve upgrading slaves to a newer version of MySQL, then upgrading the master. For example, if a master and slave both run MySQL 5.5, a typical upgrade sequence involves upgrading the slave to 5.6, then upgrading the master to 5.6. With regard to the different treatment of YEAR(2) as of MySQL 5.6.6, that upgrade sequence results in a problem: Suppose that the slave has been upgraded but not yet the master. Then creating a table containing a YEAR(2) column on the master results in a table containing a YEAR(4) column on the slave. Consequently, these operations will have a different result on the master and slave, if you use statement-based replication:

- Inserting numeric 0. The resulting value has an internal value of 2000 on the master but 0000 on the slave.
- Converting YEAR(2) to string. This operation uses the display value of YEAR(2) on the master but YEAR(4) on the slave.

To avoid such problems, use one of these strategies:

- Use row-based replication instead of statement-based replication.
- Modify all YEAR(2) columns on the master to YEAR(4) before upgrading. (Use ALTER TABLE, as described previously.) Then you can upgrade normally (slave first, then master) without introducing any YEAR(2) to YEAR(4) differences between the master and slave).

One migration method should be avoided: Do not dump your data with mysqldump and reload the dump file after upgrading. This has the potential to change YEAR(2) values, as described previously.

A migration from YEAR(2) to YEAR(4) should also involve examining application code for the possibility of changed behavior under conditions such as these:

- Code that expects selecting a YEAR column to produce exactly two digits.
- Code that does not account for different handling for inserts of numeric 0: Inserting 0 into YEAR(2) or YEAR(4) results in an internal value of 2000 or 0000, respectively.

11.3.5 Automatic Initialization and Updating for TIMESTAMP and DATETIME

TIMESTAMP and DATETIME columns can be automatically initializated and updated to the current date and time (that is, the current timestamp).

For any TIMESTAMP or DATETIME column in a table, you can assign the current timestamp as the default value, the auto-update value, or both:

- An auto-initialized column is set to the current timestamp for inserted rows that specify no value for the column.

- An auto-updated column is automatically updated to the current timestamp when the value of any other column in the row is changed from its current value. An auto-updated column remains unchanged if all other columns are set to their current values. To prevent an auto-updated column from updating when other columns change, explicitly set it to its current value. To update an auto-updated column even when other columns do not change, explicitly set it to the value it should have (for example, set it to CURRENT_TIMESTAMP).

In addition, you can initialize or update any TIMESTAMP column to the current date and time by assigning it a NULL value, unless it has been defined with the NULL attribute to permit NULL values.

To specify automatic properties, use the DEFAULT CURRENT_TIMESTAMP and ON UPDATE CURRENT_TIMESTAMP clauses in column definitions. The order of the clauses does not matter. If both are present in a column definition, either can occur first. Any of the synonyms for CURRENT_TIMESTAMP have the same meaning as CURRENT_TIMESTAMP. These are CURRENT_TIMESTAMP(), NOW(), LOCALTIME, LOCALTIME(), LOCALTIMESTAMP, and LOCALTIMESTAMP().

Use of DEFAULT CURRENT_TIMESTAMP and ON UPDATE CURRENT_TIMESTAMP is specific to TIMESTAMP and DATETIME. The DEFAULT clause also can be used to specify a constant (nonautomatic) default value; for example, DEFAULT 0 or DEFAULT '2000-01-01 00:00:00'.

Note

The following examples that use DEFAULT 0 do not work if the SQL mode is set to cause “zero” date values (specified, for example, as 0 '0000-00-00 00:00:00') to be rejected. Such dates are rejected in strict SQL mode (as of MySQL 5.7.4) or the NO_ZERO_DATE mode is enabled (before MySQL 5.7.4). Be aware that the TRADITIONAL SQL mode includes “zero” date rejection.

TIMESTAMP or DATETIME column definitions can specify the current timestamp for both the default and auto-update values, for one but not the other, or for neither. Different columns can have different combinations of automatic properties. The following rules describe the possibilities:

- With both DEFAULT CURRENT_TIMESTAMP and ON UPDATE CURRENT_TIMESTAMP, the column has the current timestamp for its default value and is automatically updated to the current timestamp.

  ```
  CREATE TABLE t1 (
    ts TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
    dt DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
  );
  ```

- With a DEFAULT clause but no ON UPDATE CURRENT_TIMESTAMP clause, the column has the given default value and is not automatically updated to the current timestamp.

  The default depends on whether the DEFAULT clause specifies CURRENT_TIMESTAMP or a constant value. With CURRENT_TIMESTAMP, the default is the current timestamp.

  ```
  CREATE TABLE t1 (
    ts TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    dt DATETIME DEFAULT CURRENT_TIMESTAMP
  );
  ```

  With a constant, the default is the given value. In this case, the column has no automatic properties at all.

  ```
  CREATE TABLE t1 (
    ts TIMESTAMP DEFAULT 0,
  ```

```
  dt DATETIME DEFAULT 0
);
```

- With an ON UPDATE CURRENT_TIMESTAMP clause and a constant DEFAULT clause, the column is automatically updated to the current timestamp and has the given constant default value.

  ```
  CREATE TABLE t1 (
    ts TIMESTAMP DEFAULT 0 ON UPDATE CURRENT_TIMESTAMP,
    dt DATETIME DEFAULT 0 ON UPDATE CURRENT_TIMESTAMP
  );
  ```

- With an ON UPDATE CURRENT_TIMESTAMP clause but no DEFAULT clause, the column is automatically updated to the current timestamp but does not have the current timestamp for its default value.

  The default in this case is type dependent. TIMESTAMP has a default of 0 unless defined with the NULL attribute, in which case the default is NULL.

  ```
  CREATE TABLE t1 (
    ts1 TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,     -- default 0
    ts2 TIMESTAMP NULL ON UPDATE CURRENT_TIMESTAMP -- default NULL
  );
  ```

  DATETIME has a default of NULL unless defined with the NOT NULL attribute, in which case the default is 0.

  ```
  CREATE TABLE t1 (
    dt1 DATETIME ON UPDATE CURRENT_TIMESTAMP,         -- default NULL
    dt2 DATETIME NOT NULL ON UPDATE CURRENT_TIMESTAMP -- default 0
  );
  ```

TIMESTAMP and DATETIME columns have no automatic properties unless they are specified explicitly, with this exception: By default, the *first* TIMESTAMP column has both DEFAULT CURRENT_TIMESTAMP and ON UPDATE CURRENT_TIMESTAMP if neither is specified explicitly. To suppress automatic properties for the first TIMESTAMP column, use one of these strategies:

- Enable the explicit_defaults_for_timestamp system variable. If this variable is enabled, the DEFAULT CURRENT_TIMESTAMP and ON UPDATE CURRENT_TIMESTAMP clauses that specify automatic initialization and updating are available, but are not assigned to any TIMESTAMP column unless explicitly included in the column definition.

- Alternatively, if explicit_defaults_for_timestamp is disabled (the default), do either of the following:

  - Define the column with a DEFAULT clause that specifies a constant default value.

  - Specify the NULL attribute. This also causes the column to permit NULL values, which means that you cannot assign the current timestamp by setting the column to NULL. Assigning NULL sets the column to NULL.

Consider these table definitions:

```
CREATE TABLE t1 (
  ts1 TIMESTAMP DEFAULT 0,
  ts2 TIMESTAMP DEFAULT CURRENT_TIMESTAMP
                ON UPDATE CURRENT_TIMESTAMP);
CREATE TABLE t2 (
  ts1 TIMESTAMP NULL,
```

```
  ts2 TIMESTAMP DEFAULT CURRENT_TIMESTAMP
                ON UPDATE CURRENT_TIMESTAMP);
CREATE TABLE t3 (
  ts1 TIMESTAMP NULL DEFAULT 0,
  ts2 TIMESTAMP DEFAULT CURRENT_TIMESTAMP
                ON UPDATE CURRENT_TIMESTAMP);
```

The tables have these properties:

- In each table definition, the first TIMESTAMP column has no automatic initialization or updating.
- The tables differ in how the ts1 column handles NULL values. For t1, ts1 is NOT NULL and assigning it a value of NULL sets it to the current timestamp. For t2 and t3, ts1 permits NULL and assigning it a value of NULL sets it to NULL.
- t2 and t3 differ in the default value for ts1. For t2, ts1 is defined to permit NULL, so the default is also NULL in the absence of an explicit DEFAULT clause. For t3, ts1 permits NULL but has an explicit default of 0.

If a TIMESTAMP or DATETIME column definition includes an explicit fractional seconds precision value anywhere, the same value must be used throughout the column definition. This is permitted:

```
CREATE TABLE t1 (
  ts TIMESTAMP(6) DEFAULT CURRENT_TIMESTAMP(6) ON UPDATE CURRENT_TIMESTAMP(6)
);
```

This is not permitted:

```
CREATE TABLE t1 (
  ts TIMESTAMP(6) DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP(3)
);
```

Automatic Timestamp Properties Before MySQL 5.6.5

Before MySQL 5.6.5, support for automatic initialization and updating is more limited:

- DEFAULT CURRENT_TIMESTAMP and ON UPDATE CURRENT_TIMESTAMP cannot be used with DATETIME columns.
- DEFAULT CURRENT_TIMESTAMP and ON UPDATE CURRENT_TIMESTAMP can be used with at most one TIMESTAMP column per table. It is not possible to have the current timestamp be the default value for one column and the auto-update value for another column.

You can choose whether to use these properties and which TIMESTAMP column should have them. It need not be the first one in a table that is automatically initialized or updated to the current timestamp. To specify automatic initialization or updating for a different TIMESTAMP column, you must suppress the automatic properties for the first one, as previously described. Then, for the other TIMESTAMP column, the rules for the DEFAULT and ON UPDATE clauses are the same as for the first TIMESTAMP column, except that if you omit both clauses, no automatic initialization or updating occurs.

TIMESTAMP Initialization and the NULL Attribute

By default, TIMESTAMP columns are NOT NULL, cannot contain NULL values, and assigning NULL assigns the current timestamp. To permit a TIMESTAMP column to contain NULL, explicitly declare it with the NULL attribute. In this case, the default value also becomes NULL unless overridden with a DEFAULT clause that specifies a different default value. DEFAULT NULL can be used to explicitly specify NULL as the default value. (For a `TIMESTAMP` column not declared with the `NULL` attribute, `DEFAULT NULL` is invalid.) If a `TIMESTAMP` column permits `NULL` values, assigning `NULL` sets it to `NULL`, not to the current timestamp.

The following table contains several `TIMESTAMP` columns that permit `NULL` values:

```
CREATE TABLE t
(
  ts1 TIMESTAMP NULL DEFAULT NULL,
  ts2 TIMESTAMP NULL DEFAULT 0,
  ts3 TIMESTAMP NULL DEFAULT CURRENT_TIMESTAMP
);
```

A `TIMESTAMP` column that permits `NULL` values does *not* take on the current timestamp at insert time except under one of the following conditions:

- Its default value is defined as `CURRENT_TIMESTAMP` and no value is specified for the column
- `CURRENT_TIMESTAMP` or any of its synonyms such as `NOW()` is explicitly inserted into the column

In other words, a `TIMESTAMP` column defined to permit `NULL` values auto-initializes only if its definition includes `DEFAULT CURRENT_TIMESTAMP`:

```
CREATE TABLE t (ts TIMESTAMP NULL DEFAULT CURRENT_TIMESTAMP);
```

If the `TIMESTAMP` column permits `NULL` values but its definition does not include `DEFAULT CURRENT_TIMESTAMP`, you must explicitly insert a value corresponding to the current date and time. Suppose that tables `t1` and `t2` have these definitions:

```
CREATE TABLE t1 (ts TIMESTAMP NULL DEFAULT '0000-00-00 00:00:00');
CREATE TABLE t2 (ts TIMESTAMP NULL DEFAULT NULL);
```

To set the `TIMESTAMP` column in either table to the current timestamp at insert time, explicitly assign it that value. For example:

```
INSERT INTO t1 VALUES (NOW());
INSERT INTO t2 VALUES (CURRENT_TIMESTAMP);
```

11.3.6 Fractional Seconds in Time Values

MySQL 5.7 has fractional seconds support for `TIME`, `DATETIME`, and `TIMESTAMP` values, with up to microseconds (6 digits) precision:

- To define a column that includes a fractional seconds part, use the syntax *`type_name(fsp)`*, where *`type_name`* is `TIME`, `DATETIME`, or `TIMESTAMP`, and *`fsp`* is the fractional seconds precision. For example:

  ```
  CREATE TABLE t1 (t TIME(3), dt DATETIME(6));
  ```

  The *`fsp`* value, if given, must be in the range 0 to 6. A value of 0 signifies that there is no fractional part. If omitted, the default precision is 0. (This differs from the standard SQL default of 6, for compatibility with previous MySQL versions.)

- Functions that take temporal arguments accept values with fractional seconds. Return values from temporal functions include fractional seconds as appropriate. For example, `NOW()` with no argument returns the current date and time with no fractional part, but takes an optional argument from 0 to 6 to specify that the return value includes a fractional seconds part of that many digits.

- Syntax for temporal literals produces temporal values: DATE '*str*', TIME '*str*', and TIMESTAMP '*str*', and the ODBC-syntax equivalents. The resulting value includes a trailing fractional seconds part if specified. Previously, the temporal type keyword was ignored and these constructs produced the string value. See Standard SQL and ODBC Date and Time Literals

11.3.7 Conversion Between Date and Time Types

To some extent, you can convert a value from one temporal type to another. However, there may be some alteration of the value or loss of information. In all cases, conversion between temporal types is subject to the range of valid values for the resulting type. For example, although DATE, DATETIME, and TIMESTAMP values all can be specified using the same set of formats, the types do not all have the same range of values. TIMESTAMP values cannot be earlier than 1970 UTC or later than '2038-01-19 03:14:07' UTC. This means that a date such as '1968-01-01', while valid as a DATE or DATETIME value, is not valid as a TIMESTAMP value and is converted to 0.

Conversion of DATE values:

- Conversion to a DATETIME or TIMESTAMP value adds a time part of '00:00:00' because the DATE value contains no time information.

- Conversion to a TIME value is not useful; the result is '00:00:00'.

Conversion of DATETIME and TIMESTAMP values:

- Conversion to a DATE value discards the time part because the DATE type contains no time information.

- Conversion to a TIME value discards the date part because the TIME type contains no date information.

For conversion of TIME values to other temporal types, the value of CURRENT_DATE() is used for the date part. The TIME is interpreted as elapsed time (not time of day) and added to the date. This means that the date part of the result differs from the current date if the time value is outside the range from '00:00:00' to '23:59:59'.

Suppose that the current date is '2012-01-01'. TIME values of '12:00:00', '24:00:00', and '-12:00:00', when converted to DATETIME or TIMESTAMP values, result in '2012-01-01 12:00:00', '2012-01-02 00:00:00', and '2011-12-31 12:00:00', respectively.

Conversion of TIME to DATE is similar but discards the time part from the result: '2012-01-01', '2012-01-02', and '2011-12-31', respectively.

Explicit conversion can be used to override implicit conversion. For example, in comparison of DATE and DATETIME values, the DATE value is coerced to the DATETIME type by adding a time part of '00:00:00'. To perform the comparison by ignoring the time part of the DATETIME value instead, use the CAST() function in the following way:

```
date_col = CAST(datetime_col AS DATE)
```

Conversion of TIME and DATETIME values to numeric form (for example, by adding +0) depends on whether the value contains a fractional seconds part. TIME(*N*) or DATETIME(*N*) is converted to integer when *N* is 0 (or omitted) and to a DECIMAL value with *N* decimal digits when *N* is greater than 0:

```
mysql> SELECT CURTIME(), CURTIME()+0, CURTIME(3)+0;
+-----------+-------------+--------------+
| CURTIME() | CURTIME()+0 | CURTIME(3)+0 |
+-----------+-------------+--------------+
| 09:28:00  |       92800 |    92800.887 |
+-----------+-------------+--------------+
mysql> SELECT NOW(), NOW()+0, NOW(3)+0;
+---------------------+----------------+--------------------+
| NOW()               | NOW()+0        | NOW(3)+0           |
+---------------------+----------------+--------------------+
| 2012-08-15 09:28:00 | 20120815092800 | 20120815092800.889 |
+---------------------+----------------+--------------------+
```

11.3.8 Two-Digit Years in Dates

Date values with two-digit years are ambiguous because the century is unknown. Such values must be interpreted into four-digit form because MySQL stores years internally using four digits.

For DATETIME, DATE, and TIMESTAMP types, MySQL interprets dates specified with ambiguous year values using these rules:

- Year values in the range 00-69 are converted to 2000-2069.
- Year values in the range 70-99 are converted to 1970-1999.

For YEAR, the rules are the same, with this exception: A numeric 00 inserted into YEAR(4) results in 0000 rather than 2000. To specify zero for YEAR(4) and have it be interpreted as 2000, specify it as a string '0' or '00'.

Remember that these rules are only heuristics that provide reasonable guesses as to what your data values mean. If the rules used by MySQL do not produce the values you require, you must provide unambiguous input containing four-digit year values.

ORDER BY properly sorts YEAR values that have two-digit years.

Some functions like MIN() and MAX() convert a YEAR to a number. This means that a value with a two-digit year does not work properly with these functions. The fix in this case is to convert the YEAR to four-digit year format.

11.4 String Types

The string types are CHAR, VARCHAR, BINARY, VARBINARY, BLOB, TEXT, ENUM, and SET. This section describes how these types work and how to use them in your queries. For string type storage requirements, see Section 11.6, “Data Type Storage Requirements”.

11.4.1 The CHAR and VARCHAR Types

The CHAR and VARCHAR types are similar, but differ in the way they are stored and retrieved. They also differ in maximum length and in whether trailing spaces are retained.

The CHAR and VARCHAR types are declared with a length that indicates the maximum number of characters you want to store. For example, CHAR(30) can hold up to 30 characters.

The length of a CHAR column is fixed to the length that you declare when you create the table. The length can be any value from 0 to 255. When CHAR values are stored, they are right-padded with spaces to the specified length. When CHAR values are retrieved, trailing spaces are removed unless the PAD_CHAR_TO_FULL_LENGTH SQL mode is enabled.

Values in VARCHAR columns are variable-length strings. The length can be specified as a value from 0 to 65,535. The effective maximum length of a VARCHAR is subject to the maximum row size (65,535 bytes, which is shared among all columns) and the character set used. See Section E.10.4, “Limits on Table Column Count and Row Size”.

In contrast to CHAR, VARCHAR values are stored as a 1-byte or 2-byte length prefix plus data. The length prefix indicates the number of bytes in the value. A column uses one length byte if values require no more than 255 bytes, two length bytes if values may require more than 255 bytes.

If strict SQL mode is not enabled and you assign a value to a CHAR or VARCHAR column that exceeds the column's maximum length, the value is truncated to fit and a warning is generated. For truncation of nonspace characters, you can cause an error to occur (rather than a warning) and suppress insertion of the value by using strict SQL mode. See Section 5.1.7, “Server SQL Modes”.

For VARCHAR columns, trailing spaces in excess of the column length are truncated prior to insertion and a warning is generated, regardless of the SQL mode in use. For CHAR columns, truncation of excess trailing spaces from inserted values is performed silently regardless of the SQL mode.

VARCHAR values are not padded when they are stored. Trailing spaces are retained when values are stored and retrieved, in conformance with standard SQL.

The following table illustrates the differences between CHAR and VARCHAR by showing the result of storing various string values into CHAR(4) and VARCHAR(4) columns (assuming that the column uses a single-byte character set such as latin1).

Value	CHAR(4)	Storage Required	VARCHAR(4)	Storage Required
`''`	`'    '`	4 bytes	`''`	1 byte
`'ab'`	`'ab  '`	4 bytes	`'ab'`	3 bytes
`'abcd'`	`'abcd'`	4 bytes	`'abcd'`	5 bytes
`'abcdefgh'`	`'abcd'`	4 bytes	`'abcd'`	5 bytes

The values shown as stored in the last row of the table apply *only when not using strict mode*; if MySQL is running in strict mode, values that exceed the column length are *not stored*, and an error results.

If a given value is stored into the CHAR(4) and VARCHAR(4) columns, the values retrieved from the columns are not always the same because trailing spaces are removed from CHAR columns upon retrieval. The following example illustrates this difference:

```
mysql> CREATE TABLE vc (v VARCHAR(4), c CHAR(4));
Query OK, 0 rows affected (0.01 sec)

mysql> INSERT INTO vc VALUES ('ab  ', 'ab  ');
Query OK, 1 row affected (0.00 sec)

mysql> SELECT CONCAT('(', v, ')'), CONCAT('(', c, ')') FROM vc;
+---------------------+---------------------+
| CONCAT('(', v, ')') | CONCAT('(', c, ')') |
+---------------------+---------------------+
| (ab  )              | (ab)                |
+---------------------+---------------------+
1 row in set (0.06 sec)
```

Values in CHAR and VARCHAR columns are sorted and compared according to the character set collation assigned to the column.

All MySQL collations are of type PADSPACE. This means that all CHAR, VARCHAR, and TEXT values in MySQL are compared without regard to any trailing spaces. “Comparison” in this context does not include the LIKE pattern-matching operator, for which trailing spaces are significant. For example:

```
mysql> CREATE TABLE names (myname CHAR(10));
Query OK, 0 rows affected (0.03 sec)
```

```
mysql> INSERT INTO names VALUES ('Monty');
Query OK, 1 row affected (0.00 sec)

mysql> SELECT myname = 'Monty', myname = 'Monty  ' FROM names;
+------------------+--------------------+
| myname = 'Monty' | myname = 'Monty  ' |
+------------------+--------------------+
|                1 |                  1 |
+------------------+--------------------+
1 row in set (0.00 sec)

mysql> SELECT myname LIKE 'Monty', myname LIKE 'Monty  ' FROM names;
+---------------------+-----------------------+
| myname LIKE 'Monty' | myname LIKE 'Monty  ' |
+---------------------+-----------------------+
|                   1 |                     0 |
+---------------------+-----------------------+
1 row in set (0.00 sec)
```

This is true for all MySQL versions, and is not affected by the server SQL mode.

Note

For more information about MySQL character sets and collations, see Section 10.1, “Character Set Support”. For additional information about storage requirements, see Section 11.6, “Data Type Storage Requirements”.

For those cases where trailing pad characters are stripped or comparisons ignore them, if a column has an index that requires unique values, inserting into the column values that differ only in number of trailing pad characters will result in a duplicate-key error. For example, if a table contains 'a', an attempt to store 'a ' causes a duplicate-key error.

11.4.2 The BINARY and VARBINARY Types

The BINARY and VARBINARY types are similar to CHAR and VARCHAR, except that they contain binary strings rather than nonbinary strings. That is, they contain byte strings rather than character strings. This means that they have no character set, and sorting and comparison are based on the numeric values of the bytes in the values.

The permissible maximum length is the same for BINARY and VARBINARY as it is for CHAR and VARCHAR, except that the length for BINARY and VARBINARY is a length in bytes rather than in characters.

The BINARY and VARBINARY data types are distinct from the CHAR BINARY and VARCHAR BINARY data types. For the latter types, the BINARY attribute does not cause the column to be treated as a binary string column. Instead, it causes the binary collation for the column character set to be used, and the column itself contains nonbinary character strings rather than binary byte strings. For example, CHAR(5) BINARY is treated as CHAR(5) CHARACTER SET latin1 COLLATE latin1_bin, assuming that the default character set is latin1. This differs from BINARY(5), which stores 5-bytes binary strings that have no character set or collation. For information about differences between nonbinary string binary collations and binary strings, see Section 10.1.7.6, “The _bin and binary Collations”.

If strict SQL mode is not enabled and you assign a value to a BINARY or VARBINARY column that exceeds the column's maximum length, the value is truncated to fit and a warning is generated. For cases of truncation, you can cause an error to occur (rather than a warning) and suppress insertion of the value by using strict SQL mode. See Section 5.1.7, “Server SQL Modes”.

When BINARY values are stored, they are right-padded with the pad value to the specified length. The pad value is 0x00 (the zero byte). Values are right-padded with 0x00 on insert, and no trailing bytes are removed on select. All bytes are significant in comparisons, including ORDER BY and DISTINCT operations. 0x00 bytes and spaces are different in comparisons, with 0x00 < space.

Example: For a BINARY(3) column, 'a ' becomes 'a \0' when inserted. 'a\0' becomes 'a\0\0' when inserted. Both inserted values remain unchanged when selected.

For VARBINARY, there is no padding on insert and no bytes are stripped on select. All bytes are significant in comparisons, including ORDER BY and DISTINCT operations. 0x00 bytes and spaces are different in comparisons, with 0x00 < space.

For those cases where trailing pad bytes are stripped or comparisons ignore them, if a column has an index that requires unique values, inserting into the column values that differ only in number of trailing pad bytes will result in a duplicate-key error. For example, if a table contains 'a', an attempt to store 'a\0' causes a duplicate-key error.

You should consider the preceding padding and stripping characteristics carefully if you plan to use the BINARY data type for storing binary data and you require that the value retrieved be exactly the same as the value stored. The following example illustrates how 0x00-padding of BINARY values affects column value comparisons:

```
mysql> CREATE TABLE t (c BINARY(3));
Query OK, 0 rows affected (0.01 sec)

mysql> INSERT INTO t SET c = 'a';
Query OK, 1 row affected (0.01 sec)

mysql> SELECT HEX(c), c = 'a', c = 'a\0\0' from t;
+--------+---------+-------------+
| HEX(c) | c = 'a' | c = 'a\0\0' |
+--------+---------+-------------+
| 610000 |       0 |           1 |
+--------+---------+-------------+
1 row in set (0.09 sec)
```

If the value retrieved must be the same as the value specified for storage with no padding, it might be preferable to use VARBINARY or one of the BLOB data types instead.

11.4.3 The BLOB and TEXT Types

A BLOB is a binary large object that can hold a variable amount of data. The four BLOB types are TINYBLOB, BLOB, MEDIUMBLOB, and LONGBLOB. These differ only in the maximum length of the values they can hold. The four TEXT types are TINYTEXT, TEXT, MEDIUMTEXT, and LONGTEXT. These correspond to the four BLOB types and have the same maximum lengths and storage requirements. See Section 11.6, “Data Type Storage Requirements”.

BLOB values are treated as binary strings (byte strings). They have no character set, and sorting and comparison are based on the numeric values of the bytes in column values. TEXT values are treated as nonbinary strings (character strings). They have a character set, and values are sorted and compared based on the collation of the character set.

If strict SQL mode is not enabled and you assign a value to a BLOB or TEXT column that exceeds the column's maximum length, the value is truncated to fit and a warning is generated. For truncation of nonspace characters, you can cause an error to occur (rather than a warning) and suppress insertion of the value by using strict SQL mode. See Section 5.1.7, “Server SQL Modes”.

Truncation of excess trailing spaces from values to be inserted into TEXT columns always generates a warning, regardless of the SQL mode.

For `TEXT` and `BLOB` columns, there is no padding on insert and no bytes are stripped on select.

If a `TEXT` column is indexed, index entry comparisons are space-padded at the end. This means that, if the index requires unique values, duplicate-key errors will occur for values that differ only in the number of trailing spaces. For example, if a table contains `'a'`, an attempt to store `'a  '` causes a duplicate-key error. This is not true for `BLOB` columns.

In most respects, you can regard a `BLOB` column as a `VARBINARY` column that can be as large as you like. Similarly, you can regard a `TEXT` column as a `VARCHAR` column. `BLOB` and `TEXT` differ from `VARBINARY` and `VARCHAR` in the following ways:

- For indexes on `BLOB` and `TEXT` columns, you must specify an index prefix length. For `CHAR` and `VARCHAR`, a prefix length is optional. See Section 8.3.4, “Column Indexes”.
- `BLOB` and `TEXT` columns cannot have `DEFAULT` values.

If you use the `BINARY` attribute with a `TEXT` data type, the column is assigned the binary collation of the column character set.

`LONG` and `LONG VARCHAR` map to the `MEDIUMTEXT` data type. This is a compatibility feature.

MySQL Connector/ODBC defines `BLOB` values as `LONGVARBINARY` and `TEXT` values as `LONGVARCHAR`.

Because `BLOB` and `TEXT` values can be extremely long, you might encounter some constraints in using them:

- Only the first `max_sort_length` bytes of the column are used when sorting. The default value of `max_sort_length` is 1024. You can make more bytes significant in sorting or grouping by increasing the value of `max_sort_length` at server startup or runtime. Any client can change the value of its session `max_sort_length` variable:

  ```
  mysql> SET max_sort_length = 2000;
  mysql> SELECT id, comment FROM t
      -> ORDER BY comment;
  ```

- Instances of `BLOB` or `TEXT` columns in the result of a query that is processed using a temporary table causes the server to use a table on disk rather than in memory because the `MEMORY` storage engine does not support those data types (see Section 8.4.4, “How MySQL Uses Internal Temporary Tables”). Use of disk incurs a performance penalty, so include `BLOB` or `TEXT` columns in the query result only if they are really needed. For example, avoid using `SELECT *`, which selects all columns.
- The maximum size of a `BLOB` or `TEXT` object is determined by its type, but the largest value you actually can transmit between the client and server is determined by the amount of available memory and the size of the communications buffers. You can change the message buffer size by changing the value of the `max_allowed_packet` variable, but you must do so for both the server and your client program. For example, both `mysql` and `mysqldump` enable you to change the client-side `max_allowed_packet` value. See Section 8.11.2, “Tuning Server Parameters”, Section 4.5.1, “`mysql` — The MySQL Command-Line Tool”, and Section 4.5.4, “`mysqldump` — A Database Backup Program”. You may also want to compare the packet sizes and the size of the data objects you are storing with the storage requirements, see Section 11.6, “Data Type Storage Requirements”

Each `BLOB` or `TEXT` value is represented internally by a separately allocated object. This is in contrast to all other data types, for which storage is allocated once per column when the table is opened.

In some cases, it may be desirable to store binary data such as media files in `BLOB` or `TEXT` columns. You may find MySQL's string handling functions useful for working with such data. See Section 12.5, “String

Functions”. For security and other reasons, it is usually preferable to do so using application code rather than giving application users the FILE privilege. You can discuss specifics for various languages and platforms in the MySQL Forums (http://forums.mysql.com/).

11.4.4 The ENUM Type

An ENUM is a string object with a value chosen from a list of permitted values that are enumerated explicitly in the column specification at table creation time. It has these advantages:

- Compact data storage in situations where a column has a limited set of possible values. The strings you specify as input values are automatically encoded as numbers. See Section 11.6, “Data Type Storage Requirements” for the storage requirements for ENUM types.

- Readable queries and output. The numbers are translated back to the corresponding strings in query results.

and these potential issues to consider:

- If you make enumeration values that look like numbers, it is easy to mix up the literal values with their internal index numbers, as explained in Enumeration Limitations.

- Using ENUM columns in ORDER BY clauses requires extra care, as explained in Enumeration Sorting.

Creating and Using ENUM Columns

An enumeration value must be a quoted string literal. For example, you can create a table with an ENUM column like this:

```
CREATE TABLE shirts (
    name VARCHAR(40),
    size ENUM('x-small', 'small', 'medium', 'large', 'x-large')
);
INSERT INTO shirts (name, size) VALUES ('dress shirt','large'), ('t-shirt','medium'),
  ('polo shirt','small');
SELECT name, size FROM shirts WHERE size = 'medium';
+---------+--------+
| name    | size   |
+---------+--------+
| t-shirt | medium |
+---------+--------+
UPDATE shirts SET size = 'small' WHERE size = 'large';
COMMIT;
```

Inserting 1 million rows into this table with a value of 'medium' would require 1 million bytes of storage, as opposed to 6 million bytes if you stored the actual string 'medium' in a VARCHAR column.

Index Values for Enumeration Literals

Each enumeration value has an index:

- The elements listed in the column specification are assigned index numbers, beginning with 1.

- The index value of the empty string error value is 0. This means that you can use the following SELECT statement to find rows into which invalid ENUM values were assigned:

```
mysql> SELECT * FROM tbl_name WHERE enum_col=0;
```

- The index of the NULL value is NULL.

- The term “index” here refers to a position within the list of enumeration values. It has nothing to do with table indexes.

For example, a column specified as `ENUM('Mercury', 'Venus', 'Earth')` can have any of the values shown here. The index of each value is also shown.

Value	Index
`NULL`	`NULL`
`''`	0
`'Mercury'`	1
`'Venus'`	2
`'Earth'`	3

An `ENUM` column can have a maximum of 65,535 distinct elements. (The practical limit is less than 3000.) A table can have no more than 255 unique element list definitions among its `ENUM` and `SET` columns considered as a group. For more information on these limits, see Section E.10.5, “Limits Imposed by `.frm` File Structure”.

If you retrieve an `ENUM` value in a numeric context, the column value's index is returned. For example, you can retrieve numeric values from an `ENUM` column like this:

```
mysql> SELECT enum_col+0 FROM tbl_name;
```

Functions such as `SUM()` or `AVG()` that expect a numeric argument cast the argument to a number if necessary. For `ENUM` values, the index number is used in the calculation.

Handling of Enumeration Literals

Trailing spaces are automatically deleted from `ENUM` member values in the table definition when a table is created.

When retrieved, values stored into an `ENUM` column are displayed using the lettercase that was used in the column definition. Note that `ENUM` columns can be assigned a character set and collation. For binary or case-sensitive collations, lettercase is taken into account when assigning values to the column.

If you store a number into an `ENUM` column, the number is treated as the index into the possible values, and the value stored is the enumeration member with that index. (However, this does *not* work with `LOAD DATA`, which treats all input as strings.) If the numeric value is quoted, it is still interpreted as an index if there is no matching string in the list of enumeration values. For these reasons, it is not advisable to define an `ENUM` column with enumeration values that look like numbers, because this can easily become confusing. For example, the following column has enumeration members with string values of `'0'`, `'1'`, and `'2'`, but numeric index values of `1`, `2`, and `3`:

```
numbers ENUM('0','1','2')
```

If you store `2`, it is interpreted as an index value, and becomes `'1'` (the value with index 2). If you store `'2'`, it matches an enumeration value, so it is stored as `'2'`. If you store `'3'`, it does not match any enumeration value, so it is treated as an index and becomes `'2'` (the value with index 3).

```
mysql> INSERT INTO t (numbers) VALUES(2),('2'),('3');
mysql> SELECT * FROM t;
+---------+
| numbers |
+---------+
| 1       |
| 2       |
| 2       |
+---------+
```

To determine all possible values for an ENUM column, use SHOW COLUMNS FROM *tbl_name* LIKE '*enum_col*' and parse the ENUM definition in the Type column of the output.

In the C API, ENUM values are returned as strings. For information about using result set metadata to distinguish them from other strings, see Section 21.8.5, "C API Data Structures".

Empty or NULL Enumeration Values

An enumeration value can also be the empty string ('') or NULL under certain circumstances:

- If you insert an invalid value into an ENUM (that is, a string not present in the list of permitted values), the empty string is inserted instead as a special error value. This string can be distinguished from a "normal" empty string by the fact that this string has the numeric value 0. See Index Values for Enumeration Literals for details about the numeric indexes for the enumeration values.

  If strict SQL mode is enabled, attempts to insert invalid ENUM values result in an error.

- If an ENUM column is declared to permit NULL, the NULL value is a valid value for the column, and the default value is NULL. If an ENUM column is declared NOT NULL, its default value is the first element of the list of permitted values.

Enumeration Sorting

ENUM values are sorted based on their index numbers, which depend on the order in which the enumeration members were listed in the column specification. For example, 'b' sorts before 'a' for ENUM('b', 'a'). The empty string sorts before nonempty strings, and NULL values sort before all other enumeration values.

To prevent unexpected results when using the ORDER BY clause on an ENUM column, use one of these techniques:

- Specify the ENUM list in alphabetic order.

- Make sure that the column is sorted lexically rather than by index number by coding ORDER BY CAST(*col* AS CHAR) or ORDER BY CONCAT(*col*).

Enumeration Limitations

An enumeration value cannot be an expression, even one that evaluates to a string value.

For example, this CREATE TABLE statement does *not* work because the CONCAT function cannot be used to construct an enumeration value:

```
CREATE TABLE sizes (
    size ENUM('small', CONCAT('med','ium'), 'large')
);
```

You also cannot employ a user variable as an enumeration value. This pair of statements do *not* work:

```
SET @mysize = 'medium';

CREATE TABLE sizes (
    size ENUM('small', @mysize, 'large')
);
```

We strongly recommend that you do *not* use numbers as enumeration values, because it does not save on storage over the appropriate TINYINT or SMALLINT type, and it is easy to mix up the strings and the underlying number values (which might not be the same) if you quote the ENUM values incorrectly. If you do use a number as an enumeration value, always enclose it in quotation marks. If the quotation marks are omitted, the number is regarded as an index. See Handling of Enumeration Literals to see how even a quoted number could be mistakenly used as a numeric index value.

Duplicate values in the definition cause a warning, or an error if strict SQL mode is enabled.

11.4.5 The SET Type

A SET is a string object that can have zero or more values, each of which must be chosen from a list of permitted values specified when the table is created. SET column values that consist of multiple set members are specified with members separated by commas (“,”). A consequence of this is that SET member values should not themselves contain commas.

For example, a column specified as SET('one', 'two') NOT NULL can have any of these values:

```
''
'one'
'two'
'one,two'
```

A SET column can have a maximum of 64 distinct members. A table can have no more than 255 unique element list definitions among its ENUM and SET columns considered as a group. For more information on this limit, see Section E.10.5, “Limits Imposed by .frm File Structure”.

Duplicate values in the definition cause a warning, or an error if strict SQL mode is enabled.

Trailing spaces are automatically deleted from SET member values in the table definition when a table is created.

When retrieved, values stored in a SET column are displayed using the lettercase that was used in the column definition. Note that SET columns can be assigned a character set and collation. For binary or case-sensitive collations, lettercase is taken into account when assigning values to the column.

MySQL stores SET values numerically, with the low-order bit of the stored value corresponding to the first set member. If you retrieve a SET value in a numeric context, the value retrieved has bits set corresponding to the set members that make up the column value. For example, you can retrieve numeric values from a SET column like this:

```
mysql> SELECT set_col+0 FROM tbl_name;
```

If a number is stored into a SET column, the bits that are set in the binary representation of the number determine the set members in the column value. For a column specified as SET('a','b','c','d'), the members have the following decimal and binary values.

SET Member	Decimal Value	Binary Value
'a'	1	0001

SET Member	Decimal Value	Binary Value
'b'	2	0010
'c'	4	0100
'd'	8	1000

If you assign a value of 9 to this column, that is 1001 in binary, so the first and fourth SET value members 'a' and 'd' are selected and the resulting value is 'a,d'.

For a value containing more than one SET element, it does not matter what order the elements are listed in when you insert the value. It also does not matter how many times a given element is listed in the value. When the value is retrieved later, each element in the value appears once, with elements listed according to the order in which they were specified at table creation time. For example, suppose that a column is specified as SET('a','b','c','d'):

```
mysql> CREATE TABLE myset (col SET('a', 'b', 'c', 'd'));
```

If you insert the values 'a,d', 'd,a', 'a,d,d', 'a,d,a', and 'd,a,d':

```
mysql> INSERT INTO myset (col) VALUES
-> ('a,d'), ('d,a'), ('a,d,a'), ('a,d,d'), ('d,a,d');
Query OK, 5 rows affected (0.01 sec)
Records: 5  Duplicates: 0  Warnings: 0
```

Then all these values appear as 'a,d' when retrieved:

```
mysql> SELECT col FROM myset;
+------+
| col  |
+------+
| a,d  |
| a,d  |
| a,d  |
| a,d  |
| a,d  |
+------+
5 rows in set (0.04 sec)
```

If you set a SET column to an unsupported value, the value is ignored and a warning is issued:

```
mysql> INSERT INTO myset (col) VALUES ('a,d,d,s');
Query OK, 1 row affected, 1 warning (0.03 sec)

mysql> SHOW WARNINGS;
+---------+------+------------------------------------------+
| Level   | Code | Message                                  |
+---------+------+------------------------------------------+
| Warning | 1265 | Data truncated for column 'col' at row 1 |
+---------+------+------------------------------------------+
1 row in set (0.04 sec)

mysql> SELECT col FROM myset;
+------+
| col  |
+------+
| a,d  |
| a,d  |
| a,d  |
| a,d  |
| a,d  |
| a,d  |
+------+
6 rows in set (0.01 sec)
```

If strict SQL mode is enabled, attempts to insert invalid `SET` values result in an error.

`SET` values are sorted numerically. `NULL` values sort before non-`NULL` `SET` values.

Functions such as `SUM()` or `AVG()` that expect a numeric argument cast the argument to a number if necessary. For `SET` values, the cast operation causes the numeric value to be used.

Normally, you search for `SET` values using the `FIND_IN_SET()` function or the `LIKE` operator:

```
mysql> SELECT * FROM tbl_name WHERE FIND_IN_SET('value',set_col)>0;
mysql> SELECT * FROM tbl_name WHERE set_col LIKE '%value%';
```

The first statement finds rows where *`set_col`* contains the *`value`* set member. The second is similar, but not the same: It finds rows where *`set_col`* contains *`value`* anywhere, even as a substring of another set member.

The following statements also are permitted:

```
mysql> SELECT * FROM tbl_name WHERE set_col & 1;
mysql> SELECT * FROM tbl_name WHERE set_col = 'val1,val2';
```

The first of these statements looks for values containing the first set member. The second looks for an exact match. Be careful with comparisons of the second type. Comparing set values to `'val1,val2'` returns different results than comparing values to `'val2,val1'`. You should specify the values in the same order they are listed in the column definition.

To determine all possible values for a `SET` column, use `SHOW COLUMNS FROM tbl_name LIKE set_col` and parse the `SET` definition in the `Type` column of the output.

In the C API, `SET` values are returned as strings. For information about using result set metadata to distinguish them from other strings, see Section 21.8.5, “C API Data Structures”.

11.5 Data Type Default Values

The `DEFAULT value` clause in a data type specification indicates a default value for a column. With one exception, the default value must be a constant; it cannot be a function or an expression. This means, for example, that you cannot set the default for a date column to be the value of a function such as `NOW()` or `CURRENT_DATE`. The exception is that you can specify `CURRENT_TIMESTAMP` as the default for `TIMESTAMP` and `DATETIME` columns. See Section 11.3.5, “Automatic Initialization and Updating for `TIMESTAMP` and `DATETIME`”.

`BLOB` and `TEXT` columns cannot be assigned a default value.

If a column definition includes no explicit `DEFAULT` value, MySQL determines the default value as follows:

If the column can take `NULL` as a value, the column is defined with an explicit `DEFAULT NULL` clause.

If the column cannot take `NULL` as the value, MySQL defines the column with no explicit `DEFAULT` clause. Exception: If the column is defined as part of a `PRIMARY KEY` but not explicitly as `NOT NULL`, MySQL creates it as a NOT NULL column (because PRIMARY KEY columns must be NOT NULL). Before MySQL 5.7.3, the column is also assigned a DEFAULT clause using the implicit default value. To prevent this, include an explicit NOT NULL in the definition of any PRIMARY KEY column.

For data entry into a NOT NULL column that has no explicit DEFAULT clause, if an INSERT or REPLACE statement includes no value for the column, or an UPDATE statement sets the column to NULL, MySQL handles the column according to the SQL mode in effect at the time:

- If strict SQL mode is enabled, an error occurs for transactional tables and the statement is rolled back. For nontransactional tables, an error occurs, but if this happens for the second or subsequent row of a multiple-row statement, the preceding rows will have been inserted.

- If strict mode is not enabled, MySQL sets the column to the implicit default value for the column data type.

Suppose that a table t is defined as follows:

```
CREATE TABLE t (i INT NOT NULL);
```

In this case, i has no explicit default, so in strict mode each of the following statements produce an error and no row is inserted. When not using strict mode, only the third statement produces an error; the implicit default is inserted for the first two statements, but the third fails because DEFAULT(i) cannot produce a value:

```
INSERT INTO t VALUES();
INSERT INTO t VALUES(DEFAULT);
INSERT INTO t VALUES(DEFAULT(i));
```

See Section 5.1.7, “Server SQL Modes”.

For a given table, you can use the SHOW CREATE TABLE statement to see which columns have an explicit DEFAULT clause.

Implicit defaults are defined as follows:

- For numeric types, the default is 0, with the exception that for integer or floating-point types declared with the AUTO_INCREMENT attribute, the default is the next value in the sequence.

- For date and time types other than TIMESTAMP, the default is the appropriate “zero” value for the type. This is also true for TIMESTAMP if the explicit_defaults_for_timestamp system variable is enabled (see Section 5.1.4, “Server System Variables”). Otherwise, for the first TIMESTAMP column in a table, the default value is the current date and time. See Section 11.3, “Date and Time Types”.

- For string types other than ENUM, the default value is the empty string. For ENUM, the default is the first enumeration value.

SERIAL DEFAULT VALUE in the definition of an integer column is an alias for NOT NULL AUTO_INCREMENT UNIQUE.

11.6 Data Type Storage Requirements

The storage requirements for table data on disk depend on several factors. Different storage engines represent data types and store raw data differently. Table data might be compressed, either for a column or an entire row, complicating the calculation of storage requirements for a table or column.

Despite differences in storage layout on disk, the internal MySQL APIs that communicate and exchange information about table rows use a consistent data structure that applies across all storage engines.

This section includes guidelines and information for the storage requirements for each data type supported by MySQL, including the internal format and size for storage engines that use a fixed-size representation for data types. Information is listed by category or storage engine.

The internal representation of a table has a maximum row size of 65,535 bytes, even if the storage engine is capable of supporting larger rows. This figure excludes BLOB or TEXT columns, which contribute only 9 to 12 bytes toward this size. For BLOB and TEXT data, the information is stored internally in a different area of memory than the row buffer. Different storage engines handle the allocation and storage of this data in different ways, according to the method they use for handling the corresponding types. For more information, see Chapter 14, *Storage Engines*, and Section E.10.4, “Limits on Table Column Count and Row Size”.

Storage Requirements for InnoDB Tables

See Physical Row Structure for information about storage requirements for InnoDB tables.

Storage Requirements for Numeric Types

Data Type	Storage Required
TINYINT	1 byte
SMALLINT	2 bytes
MEDIUMINT	3 bytes
INT, INTEGER	4 bytes
BIGINT	8 bytes
FLOAT(*p*)	4 bytes if 0 <= *p* <= 24, 8 bytes if 25 <= *p* <= 53
FLOAT	4 bytes
DOUBLE [PRECISION], REAL	8 bytes
DECIMAL(*M*,*D*), NUMERIC(*M*,*D*)	Varies; see following discussion
BIT(*M*)	approximately (*M*+7)/8 bytes

Values for DECIMAL (and NUMERIC) columns are represented using a binary format that packs nine decimal (base 10) digits into four bytes. Storage for the integer and fractional parts of each value are determined separately. Each multiple of nine digits requires four bytes, and the “leftover” digits require some fraction of four bytes. The storage required for excess digits is given by the following table.

Leftover Digits	Number of Bytes
0	0
1	1
2	1
3	2
4	2
5	3
6	3
7	4

Leftover Digits	Number of Bytes
8	4

Storage Requirements for Date and Time Types

For `TIME`, `DATETIME`, and `TIMESTAMP` columns, the storage required for tables created before MySQL 5.6.4 differs from tables created from 5.6.4 on. This is due to a change in 5.6.4 that permits these types to have a fractional part, which requires from 0 to 3 bytes.

Data Type	Storage Required Before MySQL 5.6.4	Storage Required as of MySQL 5.6.4
`YEAR`	1 byte	1 byte
`DATE`	3 bytes	3 bytes
`TIME`	3 bytes	3 bytes + fractional seconds storage
`DATETIME`	8 bytes	5 bytes + fractional seconds storage
`TIMESTAMP`	4 bytes	4 bytes + fractional seconds storage

As of MySQL 5.6.4, storage for `YEAR` and `DATE` remains unchanged. However, `TIME`, `DATETIME`, and `TIMESTAMP` are represented differently. `DATETIME` is packed more efficiently, requiring 5 rather than 8 bytes for the nonfractional part, and all three parts have a fractional part that requires from 0 to 3 bytes, depending on the fractional seconds precision of stored values.

Fractional Seconds Precision	Storage Required
0	0 bytes
1, 2	1 byte
3, 4	2 bytes
5, 6	3 bytes

For example, `TIME(0)`, `TIME(2)`, `TIME(4)`, and `TIME(6)` use 3, 4, 5, and 6 bytes, respectively. `TIME` and `TIME(0)` are equivalent and require the same storage.

For details about internal representation of temporal values, see MySQL Internals: Important Algorithms and Structures.

Storage Requirements for String Types

In the following table, *`M`* represents the declared column length in characters for nonbinary string types and bytes for binary string types. *`L`* represents the actual length in bytes of a given string value.

Data Type	Storage Required
`CHAR(M)`	*`M`* × *`w`* bytes, 0 `<=` *`M`* `<=` 255, where *`w`* is the number of bytes required for the maximum-length character in the character set. See Physical Row Structure for information about `CHAR` data type storage requirements for `InnoDB` tables.
`BINARY(M)`	*`M`* bytes, 0 `<=` *`M`* `<=` 255
`VARCHAR(M)`, `VARBINARY(M)`	*`L`* + 1 bytes if column values require 0 – 255 bytes, *`L`* + 2 bytes if values may require more than 255 bytes
`TINYBLOB`, `TINYTEXT`	*`L`* + 1 bytes, where *`L`* < 2^8
`BLOB`, `TEXT`	*`L`* + 2 bytes, where *`L`* < 2^{16}

Data Type	Storage Required
MEDIUMBLOB, MEDIUMTEXT	*L* + 3 bytes, where *L* < 2^{24}
LONGBLOB, LONGTEXT	*L* + 4 bytes, where *L* < 2^{32}
ENUM('*value1*','*value2*',...)	1 or 2 bytes, depending on the number of enumeration values (65,535 values maximum)
SET('*value1*','*value2*',...)	1, 2, 3, 4, or 8 bytes, depending on the number of set members (64 members maximum)

Variable-length string types are stored using a length prefix plus data. The length prefix requires from one to four bytes depending on the data type, and the value of the prefix is *L* (the byte length of the string). For example, storage for a MEDIUMTEXT value requires *L* bytes to store the value plus three bytes to store the length of the value.

To calculate the number of bytes used to store a particular CHAR, VARCHAR, or TEXT column value, you must take into account the character set used for that column and whether the value contains multi-byte characters. In particular, when using the utf8 (or utf8mb4) Unicode character set, you must keep in mind that not all characters use the same number of bytes and can require up to three (four) bytes per character. For a breakdown of the storage used for different categories of utf8 or utf8mb4 characters, see Section 10.1.10, “Unicode Support”.

VARCHAR, VARBINARY, and the BLOB and TEXT types are variable-length types. For each, the storage requirements depend on these factors:

- The actual length of the column value
- The column's maximum possible length
- The character set used for the column, because some character sets contain multi-byte characters

For example, a VARCHAR(255) column can hold a string with a maximum length of 255 characters. Assuming that the column uses the latin1 character set (one byte per character), the actual storage required is the length of the string (*L*), plus one byte to record the length of the string. For the string 'abcd', *L* is 4 and the storage requirement is five bytes. If the same column is instead declared to use the ucs2 double-byte character set, the storage requirement is 10 bytes: The length of 'abcd' is eight bytes and the column requires two bytes to store lengths because the maximum length is greater than 255 (up to 510 bytes).

The effective maximum number of *bytes* that can be stored in a VARCHAR or VARBINARY column is subject to the maximum row size of 65,535 bytes, which is shared among all columns. For a VARCHAR column that stores multi-byte characters, the effective maximum number of *characters* is less. For example, utf8 characters can require up to three bytes per character, so a VARCHAR column that uses the utf8 character set can be declared to be a maximum of 21,844 characters. See Section E.10.4, “Limits on Table Column Count and Row Size”.

The size of an ENUM object is determined by the number of different enumeration values. One byte is used for enumerations with up to 255 possible values. Two bytes are used for enumerations having between 256 and 65,535 possible values. See Section 11.4.4, “The ENUM Type”.

The size of a SET object is determined by the number of different set members. If the set size is *N*, the object occupies (*N*+7)/8 bytes, rounded up to 1, 2, 3, 4, or 8 bytes. A SET can have a maximum of 64 members. See Section 11.4.5, “The SET Type”.

11.7 Choosing the Right Type for a Column

For optimum storage, you should try to use the most precise type in all cases. For example, if an integer column is used for values in the range from 1 to 99999, MEDIUMINT UNSIGNED is the best type. Of the types that represent all the required values, this type uses the least amount of storage.

All basic calculations (+, -, *, and /) with DECIMAL columns are done with precision of 65 decimal (base 10) digits. See Section 11.1.1, “Numeric Type Overview”.

If accuracy is not too important or if speed is the highest priority, the DOUBLE type may be good enough. For high precision, you can always convert to a fixed-point type stored in a BIGINT. This enables you to do all calculations with 64-bit integers and then convert results back to floating-point values as necessary.

PROCEDURE ANALYSE can be used to obtain suggestions for optimal column data types. For more information, see Section 8.4.2.4, “Using PROCEDURE ANALYSE”.

11.8 Using Data Types from Other Database Engines

To facilitate the use of code written for SQL implementations from other vendors, MySQL maps data types as shown in the following table. These mappings make it easier to import table definitions from other database systems into MySQL.

Other Vendor Type	MySQL Type
BOOL	TINYINT
BOOLEAN	TINYINT
CHARACTER VARYING(*M*)	VARCHAR(*M*)
FIXED	DECIMAL
FLOAT4	FLOAT
FLOAT8	DOUBLE
INT1	TINYINT
INT2	SMALLINT
INT3	MEDIUMINT
INT4	INT
INT8	BIGINT
LONG VARBINARY	MEDIUMBLOB
LONG VARCHAR	MEDIUMTEXT
LONG	MEDIUMTEXT
MIDDLEINT	MEDIUMINT
NUMERIC	DECIMAL

Data type mapping occurs at table creation time, after which the original type specifications are discarded. If you create a table with types used by other vendors and then issue a DESCRIBE *tbl_name* statement, MySQL reports the table structure using the equivalent MySQL types. For example:

```
mysql> CREATE TABLE t (a BOOL, b FLOAT8, c LONG VARCHAR, d NUMERIC);
Query OK, 0 rows affected (0.00 sec)

mysql> DESCRIBE t;
+-------+---------------+------+-----+---------+-------+
| Field | Type          | Null | Key | Default | Extra |
+-------+---------------+------+-----+---------+-------+
| a     | tinyint(1)    | YES  |     | NULL    |       |
| b     | double        | YES  |     | NULL    |       |
| c     | mediumtext    | YES  |     | NULL    |       |
| d     | decimal(10,0) | YES  |     | NULL    |       |
+-------+---------------+------+-----+---------+-------+
4 rows in set (0.01 sec)
```

Chapter 12 Functions and Operators

Table of Contents

Expressions can be used at several points in SQL statements, such as in the ORDER BY or HAVING clauses of SELECT statements, in the WHERE clause of a SELECT, DELETE, or UPDATE statement, or in SET statements. Expressions can be written using literal values, column values, NULL, built-in functions, stored functions, user-defined functions, and operators. This chapter describes the functions and operators that are permitted for writing expressions in MySQL. Instructions for writing stored functions and user-defined functions are given in Section 18.2, “Using Stored Routines (Procedures and Functions)”, and Section 22.3, “Adding New Functions to MySQL”. See Section 9.2.4, “Function Name Parsing and Resolution”, for the rules describing how the server interprets references to different kinds of functions.

An expression that contains NULL always produces a NULL value unless otherwise indicated in the documentation for a particular function or operator.

Note

By default, there must be no whitespace between a function name and the parenthesis following it. This helps the MySQL parser distinguish between function calls and references to tables or columns that happen to have the same name as a function. However, spaces around function arguments are permitted.

You can tell the MySQL server to accept spaces after function names by starting it with the --sql-mode=IGNORE_SPACE option. (See Section 5.1.7, “Server SQL Modes”.) Individual client programs can request this behavior by using the CLIENT_IGNORE_SPACE option for mysql_real_connect(). In either case, all function names become reserved words.

For the sake of brevity, most examples in this chapter display the output from the mysql program in abbreviated form. Rather than showing examples in this format:

```
mysql> SELECT MOD(29,9);
+-----------+
| mod(29,9) |
+-----------+
|         2 |
+-----------+
1 rows in set (0.00 sec)
```

This format is used instead:

```
mysql> SELECT MOD(29,9);
        -> 2
```

12.1 Function and Operator Reference

Table 12.1 Functions/Operators

Name	Description
ABS()	Return the absolute value
ACOS()	Return the arc cosine
ADDDATE()	Add time values (intervals) to a date value
ADDTIME()	Add time
AES_DECRYPT()	Decrypt using AES
AES_ENCRYPT()	Encrypt using AES
AND, &&	Logical AND

Name	Description
ASCII()	Return numeric value of left-most character
ASIN()	Return the arc sine
=	Assign a value (as part of a SET statement, or as part of the SET clause in an UPDATE statement)
:=	Assign a value
ATAN2(), ATAN()	Return the arc tangent of the two arguments
ATAN()	Return the arc tangent
AVG()	Return the average value of the argument
BENCHMARK()	Repeatedly execute an expression
BETWEEN ... AND ...	Check whether a value is within a range of values
BIN()	Return a string containing binary representation of a number
BINARY	Cast a string to a binary string
BIT_AND()	Return bitwise and
BIT_COUNT()	Return the number of bits that are set
BIT_LENGTH()	Return length of argument in bits
BIT_OR()	Return bitwise or
BIT_XOR()	Return bitwise xor
&	Bitwise AND
~	Invert bits
|	Bitwise OR
^	Bitwise XOR
CASE	Case operator
CAST()	Cast a value as a certain type
CEIL()	Return the smallest integer value not less than the argument
CEILING()	Return the smallest integer value not less than the argument
CHAR_LENGTH()	Return number of characters in argument
CHAR()	Return the character for each integer passed
CHARACTER_LENGTH()	Synonym for CHAR_LENGTH()
CHARSET()	Return the character set of the argument
COALESCE()	Return the first non-NULL argument
COERCIBILITY()	Return the collation coercibility value of the string argument
COLLATION()	Return the collation of the string argument
COMPRESS()	Return result as a binary string
CONCAT_WS()	Return concatenate with separator
CONCAT()	Return concatenated string
CONNECTION_ID()	Return the connection ID (thread ID) for the connection
CONV()	Convert numbers between different number bases
CONVERT_TZ()	Convert from one timezone to another

Name	Description
CONVERT()	Cast a value as a certain type
COS()	Return the cosine
COT()	Return the cotangent
COUNT(DISTINCT)	Return the count of a number of different values
COUNT()	Return a count of the number of rows returned
CRC32()	Compute a cyclic redundancy check value
CURDATE()	Return the current date
CURRENT_DATE(), CURRENT_DATE	Synonyms for CURDATE()
CURRENT_TIME(), CURRENT_TIME	Synonyms for CURTIME()
CURRENT_TIMESTAMP(), CURRENT_TIMESTAMP	Synonyms for NOW()
CURRENT_USER(), CURRENT_USER	The authenticated user name and host name
CURTIME()	Return the current time
DATABASE()	Return the default (current) database name
DATE_ADD()	Add time values (intervals) to a date value
DATE_FORMAT()	Format date as specified
DATE_SUB()	Subtract a time value (interval) from a date
DATE()	Extract the date part of a date or datetime expression
DATEDIFF()	Subtract two dates
DAY()	Synonym for DAYOFMONTH()
DAYNAME()	Return the name of the weekday
DAYOFMONTH()	Return the day of the month (0-31)
DAYOFWEEK()	Return the weekday index of the argument
DAYOFYEAR()	Return the day of the year (1-366)
DECODE()	Decodes a string encrypted using ENCODE()
DEFAULT()	Return the default value for a table column
DEGREES()	Convert radians to degrees
DES_DECRYPT()	Decrypt a string
DES_ENCRYPT()	Encrypt a string
DIV	Integer division
/	Division operator
ELT()	Return string at index number
ENCODE()	Encode a string
ENCRYPT()	Encrypt a string
<=>	NULL-safe equal to operator
=	Equal operator
EXP()	Raise to the power of

Name	Description
`EXPORT_SET()`	Return a string such that for every bit set in the value bits, you get an on string and for every unset bit, you get an off string
`EXTRACT()`	Extract part of a date
`ExtractValue()`	Extracts a value from an XML string using XPath notation
`FIELD()`	Return the index (position) of the first argument in the subsequent arguments
`FIND_IN_SET()`	Return the index position of the first argument within the second argument
`FLOOR()`	Return the largest integer value not greater than the argument
`FORMAT()`	Return a number formatted to specified number of decimal places
`FOUND_ROWS()`	For a SELECT with a LIMIT clause, the number of rows that would be returned were there no LIMIT clause
`FROM_BASE64()`	Decode to a base-64 string and return result
`FROM_DAYS()`	Convert a day number to a date
`FROM_UNIXTIME()`	Format UNIX timestamp as a date
`GET_FORMAT()`	Return a date format string
`GET_LOCK()`	Get a named lock
`>=`	Greater than or equal operator
`>`	Greater than operator
`GREATEST()`	Return the largest argument
`GROUP_CONCAT()`	Return a concatenated string
`GTID_SUBSET()`	Return true if all GTIDs in subset are also in set; otherwise false.
`GTID_SUBTRACT()`	Return all GTIDs in set that are not in subset.
`HEX()`	Return a hexadecimal representation of a decimal or string value
`HOUR()`	Extract the hour
`IF()`	If/else construct
`IFNULL()`	Null if/else construct
`IN()`	Check whether a value is within a set of values
`INET_ATON()`	Return the numeric value of an IP address
`INET_NTOA()`	Return the IP address from a numeric value
`INET6_ATON()`	Return the numeric value of an IPv6 address
`INET6_NTOA()`	Return the IPv6 address from a numeric value
`INSERT()`	Insert a substring at the specified position up to the specified number of characters
`INSTR()`	Return the index of the first occurrence of substring
`INTERVAL()`	Return the index of the argument that is less than the first argument

Name	Description
IS_FREE_LOCK()	Checks whether the named lock is free
IS_IPV4_COMPAT()	Return true if argument is an IPv4-compatible address
IS_IPV4_MAPPED()	Return true if argument is an IPv4-mapped address
IS_IPV4()	Return true if argument is an IPv4 address
IS_IPV6()	Return true if argument is an IPv6 address
IS NOT NULL	NOT NULL value test
IS NOT	Test a value against a boolean
IS NULL	NULL value test
IS_USED_LOCK()	Checks whether the named lock is in use. Return connection identifier if true.
IS	Test a value against a boolean
ISNULL()	Test whether the argument is NULL
LAST_DAY	Return the last day of the month for the argument
LAST_INSERT_ID()	Value of the AUTOINCREMENT column for the last INSERT
LCASE()	Synonym for LOWER()
LEAST()	Return the smallest argument
<<	Left shift
LEFT()	Return the leftmost number of characters as specified
LENGTH()	Return the length of a string in bytes
<=	Less than or equal operator
<	Less than operator
LIKE	Simple pattern matching
LN()	Return the natural logarithm of the argument
LOAD_FILE()	Load the named file
LOCALTIME(), LOCALTIME	Synonym for NOW()
LOCALTIMESTAMP, LOCALTIMESTAMP()	Synonym for NOW()
LOCATE()	Return the position of the first occurrence of substring
LOG10()	Return the base-10 logarithm of the argument
LOG2()	Return the base-2 logarithm of the argument
LOG()	Return the natural logarithm of the first argument
LOWER()	Return the argument in lowercase
LPAD()	Return the string argument, left-padded with the specified string
LTRIM()	Remove leading spaces
MAKE_SET()	Return a set of comma-separated strings that have the corresponding bit in bits set
MAKEDATE()	Create a date from the year and day of year
MAKETIME()	Create time from hour, minute, second

Name	Description
MASTER_POS_WAIT()	Block until the slave has read and applied all updates up to the specified position
MATCH [1271]	Perform full-text search
MAX()	Return the maximum value
MD5()	Calculate MD5 checksum
MICROSECOND()	Return the microseconds from argument
MID()	Return a substring starting from the specified position
MIN()	Return the minimum value
-	Minus operator
MINUTE()	Return the minute from the argument
MOD()	Return the remainder
% or MOD	Modulo operator
MONTH()	Return the month from the date passed
MONTHNAME()	Return the name of the month
NAME_CONST()	Causes the column to have the given name
NOT BETWEEN ... AND ...	Check whether a value is not within a range of values
!=, <>	Not equal operator
NOT IN()	Check whether a value is not within a set of values
NOT LIKE	Negation of simple pattern matching
NOT REGEXP	Negation of REGEXP
NOT, !	Negates value
NOW()	Return the current date and time
NULLIF()	Return NULL if expr1 = expr2
OCT()	Return a string containing octal representation of a number
OCTET_LENGTH()	Synonym for LENGTH()
OLD_PASSWORD()	Return the value of the pre-4.1 implementation of PASSWORD
||, OR	Logical OR
ORD()	Return character code for leftmost character of the argument
PASSWORD()	Calculate and return a password string
PERIOD_ADD()	Add a period to a year-month
PERIOD_DIFF()	Return the number of months between periods
PI()	Return the value of pi
+	Addition operator
POSITION()	Synonym for LOCATE()
POW()	Return the argument raised to the specified power
POWER()	Return the argument raised to the specified power
PROCEDURE ANALYSE()	Analyze the results of a query
QUARTER()	Return the quarter from a date argument

Name	Description
QUOTE()	Escape the argument for use in an SQL statement
RADIANS()	Return argument converted to radians
RAND()	Return a random floating-point value
RANDOM_BYTES()	Return a random byte vector
REGEXP	Pattern matching using regular expressions
RELEASE_LOCK()	Releases the named lock
REPEAT()	Repeat a string the specified number of times
REPLACE()	Replace occurrences of a specified string
REVERSE()	Reverse the characters in a string
>>	Right shift
RIGHT()	Return the specified rightmost number of characters
RLIKE	Synonym for REGEXP
ROUND()	Round the argument
ROW_COUNT()	The number of rows updated
RPAD()	Append string the specified number of times
RTRIM()	Remove trailing spaces
SCHEMA()	Synonym for DATABASE()
SEC_TO_TIME()	Converts seconds to 'HH:MM:SS' format
SECOND()	Return the second (0-59)
SESSION_USER()	Synonym for USER()
SHA1(), SHA()	Calculate an SHA-1 160-bit checksum
SHA2()	Calculate an SHA-2 checksum
SIGN()	Return the sign of the argument
SIN()	Return the sine of the argument
SLEEP()	Sleep for a number of seconds
SOUNDEX()	Return a soundex string
SOUNDS LIKE	Compare sounds
SPACE()	Return a string of the specified number of spaces
SQRT()	Return the square root of the argument
STD()	Return the population standard deviation
STDDEV_POP()	Return the population standard deviation
STDDEV_SAMP()	Return the sample standard deviation
STDDEV()	Return the population standard deviation
STR_TO_DATE()	Convert a string to a date
STRCMP()	Compare two strings
SUBDATE()	Synonym for DATE_SUB() when invoked with three arguments
SUBSTR()	Return the substring as specified

Name	Description
SUBSTRING_INDEX()	Return a substring from a string before the specified number of occurrences of the delimiter
SUBSTRING()	Return the substring as specified
SUBTIME()	Subtract times
SUM()	Return the sum
SYSDATE()	Return the time at which the function executes
SYSTEM_USER()	Synonym for USER()
TAN()	Return the tangent of the argument
TIME_FORMAT()	Format as time
TIME_TO_SEC()	Return the argument converted to seconds
TIME()	Extract the time portion of the expression passed
TIMEDIFF()	Subtract time
*	Multiplication operator
TIMESTAMP()	With a single argument, this function returns the date or datetime expression; with two arguments, the sum of the arguments
TIMESTAMPADD()	Add an interval to a datetime expression
TIMESTAMPDIFF()	Subtract an interval from a datetime expression
TO_BASE64()	Return the argument converted to a base-64 string
TO_DAYS()	Return the date argument converted to days
TO_SECONDS()	Return the date or datetime argument converted to seconds since Year 0
TRIM()	Remove leading and trailing spaces
TRUNCATE()	Truncate to specified number of decimal places
UCASE()	Synonym for UPPER()
-	Change the sign of the argument
UNCOMPRESS()	Uncompress a string compressed
UNCOMPRESSED_LENGTH()	Return the length of a string before compression
UNHEX()	Return a string containing hex representation of a number
UNIX_TIMESTAMP()	Return a UNIX timestamp
UpdateXML()	Return replaced XML fragment
UPPER()	Convert to uppercase
USER()	The user name and host name provided by the client
UTC_DATE()	Return the current UTC date
UTC_TIME()	Return the current UTC time
UTC_TIMESTAMP()	Return the current UTC date and time
UUID_SHORT()	Return an integer-valued universal identifier
UUID()	Return a Universal Unique Identifier (UUID)
VALIDATE_PASSWORD_STRENGTH()	Determine strength of password

Name	Description
VALUES()	Defines the values to be used during an INSERT
VAR_POP()	Return the population standard variance
VAR_SAMP()	Return the sample variance
VARIANCE()	Return the population standard variance
VERSION()	Returns a string that indicates the MySQL server version
WAIT_UNTIL_SQL_THREAD_AFTER_GTIDS()	Wait until the slave SQL thread has executed all the given GTIDs. Returns: the number of events that were executed (or NULL, if GTID mode is not enabled).
WEEK()	Return the week number
WEEKDAY()	Return the weekday index
WEEKOFYEAR()	Return the calendar week of the date (0-53)
WEIGHT_STRING()	Return the weight string for a string
XOR	Logical XOR
YEAR()	Return the year
YEARWEEK()	Return the year and week

12.2 Type Conversion in Expression Evaluation

When an operator is used with operands of different types, type conversion occurs to make the operands compatible. Some conversions occur implicitly. For example, MySQL automatically converts numbers to strings as necessary, and vice versa.

```
mysql> SELECT 1+'1';
        -> 2
mysql> SELECT CONCAT(2,' test');
        -> '2 test'
```

It is also possible to convert a number to a string explicitly using the CAST() function. Conversion occurs implicitly with the CONCAT() function because it expects string arguments.

```
mysql> SELECT 38.8, CAST(38.8 AS CHAR);
        -> 38.8, '38.8'
mysql> SELECT 38.8, CONCAT(38.8);
        -> 38.8, '38.8'
```

See later in this section for information about the character set of implicit number-to-string conversions, and for modified rules that apply to CREATE TABLE ... SELECT statements.

The following rules describe how conversion occurs for comparison operations:

- If one or both arguments are NULL, the result of the comparison is NULL, except for the NULL-safe <=> equality comparison operator. For NULL <=> NULL, the result is true. No conversion is needed.
- If both arguments in a comparison operation are strings, they are compared as strings.
- If both arguments are integers, they are compared as integers.
- Hexadecimal values are treated as binary strings if not compared to a number.
- If one of the arguments is a TIMESTAMP or DATETIME column and the other argument is a constant, the constant is converted to a timestamp before the comparison is performed. This is done to be more

ODBC-friendly. Note that this is not done for the arguments to `IN()`! To be safe, always use complete datetime, date, or time strings when doing comparisons. For example, to achieve best results when using `BETWEEN` with date or time values, use `CAST()` to explicitly convert the values to the desired data type.

A single-row subquery from a table or tables is not considered a constant. For example, if a subquery returns an integer to be compared to a `DATETIME` value, the comparison is done as two integers. The integer is not converted to a temporal value. To compare the operands as `DATETIME` values, use `CAST()` to explicitly convert the subquery value to `DATETIME`.

- If one of the arguments is a decimal value, comparison depends on the other argument. The arguments are compared as decimal values if the other argument is a decimal or integer value, or as floating-point values if the other argument is a floating-point value.

- In all other cases, the arguments are compared as floating-point (real) numbers.

For information about conversion of values from one temporal type to another, see Section 11.3.7, “Conversion Between Date and Time Types”.

The following examples illustrate conversion of strings to numbers for comparison operations:

```
mysql> SELECT 1 > '6x';
        -> 0
mysql> SELECT 7 > '6x';
        -> 1
mysql> SELECT 0 > 'x6';
        -> 0
mysql> SELECT 0 = 'x6';
        -> 1
```

For comparisons of a string column with a number, MySQL cannot use an index on the column to look up the value quickly. If *`str_col`* is an indexed string column, the index cannot be used when performing the lookup in the following statement:

```
SELECT * FROM tbl_name WHERE str_col=1;
```

The reason for this is that there are many different strings that may convert to the value `1`, such as `'1'`, `' 1'`, or `'1a'`.

Comparisons that use floating-point numbers (or values that are converted to floating-point numbers) are approximate because such numbers are inexact. This might lead to results that appear inconsistent:

```
mysql> SELECT '18015376320243458' = 18015376320243458;
        -> 1
mysql> SELECT '18015376320243459' = 18015376320243459;
        -> 0
```

Such results can occur because the values are converted to floating-point numbers, which have only 53 bits of precision and are subject to rounding:

```
mysql> SELECT '18015376320243459'+0.0;
        -> 1.8015376320243e+16
```

Furthermore, the conversion from string to floating-point and from integer to floating-point do not necessarily occur the same way. The integer may be converted to floating-point by the CPU, whereas the string is converted digit by digit in an operation that involves floating-point multiplications.

The results shown will vary on different systems, and can be affected by factors such as computer architecture or the compiler version or optimization level. One way to avoid such problems is to use CAST() so that a value is not converted implicitly to a float-point number:

```
mysql> SELECT CAST('18015376320243459' AS UNSIGNED) = 18015376320243459;
        -> 1
```

For more information about floating-point comparisons, see Section C.5.5.8, “Problems with Floating-Point Values”.

In MySQL 5.7, the server includes dtoa, a conversion library that provides the basis for improved conversion between string or DECIMAL values and approximate-value (FLOAT/DOUBLE) numbers:

- Consistent conversion results across platforms, which eliminates, for example, Unix versus Windows conversion differences.

- Accurate representation of values in cases where results previously did not provide sufficient precision, such as for values close to IEEE limits.

- Conversion of numbers to string format with the best possible precision. The precision of dtoa is always the same or better than that of the standard C library functions.

Because the conversions produced by this library differ in some cases from non-dtoa results, the potential exists for incompatibilities in applications that rely on previous results. For example, applications that depend on a specific exact result from previous conversions might need adjustment to accommodate additional precision.

The dtoa library provides conversions with the following properties. *D* represents a value with a DECIMAL or string representation, and *F* represents a floating-point number in native binary (IEEE) format.

- *F* -> *D* conversion is done with the best possible precision, returning *D* as the shortest string that yields *F* when read back in and rounded to the nearest value in native binary format as specified by IEEE.

- *D* -> *F* conversion is done such that *F* is the nearest native binary number to the input decimal string *D*.

These properties imply that *F* -> *D* -> *F* conversions are lossless unless *F* is -inf, +inf, or NaN. The latter values are not supported because the SQL standard defines them as invalid values for FLOAT or DOUBLE.

For *D* -> *F* -> *D* conversions, a sufficient condition for losslessness is that *D* uses 15 or fewer digits of precision, is not a denormal value, -inf, +inf, or NaN. In some cases, the conversion is lossless even if *D* has more than 15 digits of precision, but this is not always the case.

In MySQL 5.7, implicit conversion of a numeric or temporal value to string produces a value that has a character set and collation determined by the character_set_connection and collation_connection system variables. (These variables commonly are set with SET NAMES. For information about connection character sets, see Section 10.1.4, “Connection Character Sets and Collations”.)

This means that such a conversion results in a character (nonbinary) string (a CHAR, VARCHAR, or LONGTEXT value), except in the case that the connection character set is set to binary. In that case, the conversion result is a binary string (a BINARY, VARBINARY, or LONGBLOB value).

For integer expressions, the preceding remarks about expression *evaluation* apply somewhat differently for expression *assignment*; for example, in a statement such as this:

```
CREATE TABLE t SELECT integer_expr;
```

In this case, the table in the column resulting from the expression has type INT or BIGINT depending on the length of the integer expression. If the maximum length of the expression does no fit in an INT, BIGINT is used instead. The length is taken from the max_length value of the SELECT result set metadata (see Section 21.8.5, “C API Data Structures”). This means that you can force a BIGINT rather than INT by use of a sufficiently long expression:

```
CREATE TABLE t SELECT 000000000000000000000;
```

12.3 Operators

Table 12.2 Operators

Name	Description
AND, &&	Logical AND
=	Assign a value (as part of a SET statement, or as part of the SET clause in an UPDATE statement)
:=	Assign a value
BETWEEN ... AND ...	Check whether a value is within a range of values
BINARY	Cast a string to a binary string
&	Bitwise AND
~	Invert bits
|	Bitwise OR
^	Bitwise XOR
CASE	Case operator
DIV	Integer division
/	Division operator
<=>	NULL-safe equal to operator
=	Equal operator
>=	Greater than or equal operator
>	Greater than operator
IS NOT NULL	NOT NULL value test
IS NOT	Test a value against a boolean
IS NULL	NULL value test
IS	Test a value against a boolean
<<	Left shift
<=	Less than or equal operator
<	Less than operator
LIKE	Simple pattern matching
-	Minus operator
% or MOD	Modulo operator
NOT BETWEEN ... AND ...	Check whether a value is not within a range of values
!=, <>	Not equal operator
NOT LIKE	Negation of simple pattern matching

Name	Description
NOT REGEXP	Negation of REGEXP
NOT, !	Negates value
||, OR	Logical OR
+	Addition operator
REGEXP	Pattern matching using regular expressions
>>	Right shift
RLIKE	Synonym for REGEXP
SOUNDS LIKE	Compare sounds
*	Multiplication operator
-	Change the sign of the argument
XOR	Logical XOR

12.3.1 Operator Precedence

Operator precedences are shown in the following list, from highest precedence to the lowest. Operators that are shown together on a line have the same precedence.

```
INTERVAL
BINARY, COLLATE
!
- (unary minus), ~ (unary bit inversion)
^
*, /, DIV, %, MOD
-, +
<<, >>
&
|
= (comparison), <=>, >=, >, <=, <, <>, !=, IS, LIKE, REGEXP, IN
BETWEEN, CASE, WHEN, THEN, ELSE
NOT
&&, AND
XOR
||, OR
= (assignment), :=
```

The precedence of = depends on whether it is used as a comparison operator (=) or as an assignment operator (=). When used as a comparison operator, it has the same precedence as <=>, >=, >, <=, <, <>, !=, IS, LIKE, REGEXP, and IN. When used as an assignment operator, it has the same precedence as :=. Section 13.7.4, "SET Syntax", and Section 9.4, "User-Defined Variables", explain how MySQL determines which interpretation of = should apply.

The meaning of some operators depends on the SQL mode:

- By default, || is a logical OR operator. With PIPES_AS_CONCAT enabled, || is string concatenation, with a precedence between ^ and the unary operators.
- By default, ! has a higher precedence than NOT. With HIGH_NOT_PRECEDENCE enabled, ! and NOT have the same precedence.

See Section 5.1.7, "Server SQL Modes".

The precedence of operators determines the order of evaluation of terms in an expression. To override this order and group terms explicitly, use parentheses. For example:

```
mysql> SELECT 1+2*3;
        -> 7
mysql> SELECT (1+2)*3;
        -> 9
```

12.3.2 Comparison Functions and Operators

Table 12.3 Comparison Operators

Name	Description
BETWEEN ... AND ...	Check whether a value is within a range of values
COALESCE()	Return the first non-NULL argument
<=>	NULL-safe equal to operator
=	Equal operator
>=	Greater than or equal operator
>	Greater than operator
GREATEST()	Return the largest argument
IN()	Check whether a value is within a set of values
INTERVAL()	Return the index of the argument that is less than the first argument
IS NOT NULL	NOT NULL value test
IS NOT	Test a value against a boolean
IS NULL	NULL value test
IS	Test a value against a boolean
ISNULL()	Test whether the argument is NULL
LEAST()	Return the smallest argument
<=	Less than or equal operator
<	Less than operator
LIKE	Simple pattern matching
NOT BETWEEN ... AND ...	Check whether a value is not within a range of values
!=, <>	Not equal operator
NOT IN()	Check whether a value is not within a set of values
NOT LIKE	Negation of simple pattern matching
STRCMP()	Compare two strings

Comparison operations result in a value of 1 (TRUE), 0 (FALSE), or NULL. These operations work for both numbers and strings. Strings are automatically converted to numbers and numbers to strings as necessary.

The following relational comparison operators can be used to compare not only scalar operands, but row operands:

```
=  >  <  >=  <=  <>  !=
```

For examples of row comparisons, see Section 13.2.10.5, “Row Subqueries”.

Some of the functions in this section return values other than `1` (`TRUE`), `0` (`FALSE`), or `NULL`. For example, `LEAST()` and `GREATEST()`. However, the value they return is based on comparison operations performed according to the rules described in Section 12.2, “Type Conversion in Expression Evaluation”.

To convert a value to a specific type for comparison purposes, you can use the `CAST()` function. String values can be converted to a different character set using `CONVERT()`. See Section 12.10, “Cast Functions and Operators”.

By default, string comparisons are not case sensitive and use the current character set. The default is `latin1` (cp1252 West European), which also works well for English.

- `=`

  Equal:

  ```
  mysql> SELECT 1 = 0;
          -> 0
  mysql> SELECT '0' = 0;
          -> 1
  mysql> SELECT '0.0' = 0;
          -> 1
  mysql> SELECT '0.01' = 0;
          -> 0
  mysql> SELECT '.01' = 0.01;
          -> 1
  ```

- `<=>`

  `NULL`-safe equal. This operator performs an equality comparison like the `=` operator, but returns `1` rather than `NULL` if both operands are `NULL`, and `0` rather than `NULL` if one operand is `NULL`.

  ```
  mysql> SELECT 1 <=> 1, NULL <=> NULL, 1 <=> NULL;
          -> 1, 1, 0
  mysql> SELECT 1 = 1, NULL = NULL, 1 = NULL;
          -> 1, NULL, NULL
  ```

- `<>`, `!=`

  Not equal:

  ```
  mysql> SELECT '.01' <> '0.01';
          -> 1
  mysql> SELECT .01 <> '0.01';
          -> 0
  mysql> SELECT 'zapp' <> 'zappp';
          -> 1
  ```

- `<=`

  Less than or equal:

  ```
  mysql> SELECT 0.1 <= 2;
          -> 1
  ```

- `<`

  Less than:

```
mysql> SELECT 2 < 2;
        -> 0
```

- `>=`

  Greater than or equal:

  ```
  mysql> SELECT 2 >= 2;
          -> 1
  ```

- `>`

  Greater than:

  ```
  mysql> SELECT 2 > 2;
          -> 0
  ```

- `IS boolean_value`

  Tests a value against a boolean value, where *`boolean_value`* can be `TRUE`, `FALSE`, or `UNKNOWN`.

  ```
  mysql> SELECT 1 IS TRUE, 0 IS FALSE, NULL IS UNKNOWN;
          -> 1, 1, 1
  ```

- `IS NOT boolean_value`

  Tests a value against a boolean value, where *`boolean_value`* can be `TRUE`, `FALSE`, or `UNKNOWN`.

  ```
  mysql> SELECT 1 IS NOT UNKNOWN, 0 IS NOT UNKNOWN, NULL IS NOT UNKNOWN;
          -> 1, 1, 0
  ```

- `IS NULL`

  Tests whether a value is `NULL`.

  ```
  mysql> SELECT 1 IS NULL, 0 IS NULL, NULL IS NULL;
          -> 0, 0, 1
  ```

  To work well with ODBC programs, MySQL supports the following extra features when using `IS NULL`:

  - If `sql_auto_is_null` variable is set to 1, then after a statement that successfully inserts an automatically generated `AUTO_INCREMENT` value, you can find that value by issuing a statement of the following form:

    ```
    SELECT * FROM tbl_name WHERE auto_col IS NULL
    ```

    If the statement returns a row, the value returned is the same as if you invoked the `LAST_INSERT_ID()` function. For details, including the return value after a multiple-row insert, see Section 12.14, “Information Functions”. If no `AUTO_INCREMENT` value was successfully inserted, the `SELECT` statement returns no row.

    The behavior of retrieving an `AUTO_INCREMENT` value by using an `IS NULL` comparison can be disabled by setting `sql_auto_is_null = 0`. See Section 5.1.4, “Server System Variables”.

    The default value of `sql_auto_is_null` is 0 in MySQL 5.7.

- For DATE and DATETIME columns that are declared as NOT NULL, you can find the special date '0000-00-00' by using a statement like this:

  ```
  SELECT * FROM tbl_name WHERE date_column IS NULL
  ```

  This is needed to get some ODBC applications to work because ODBC does not support a '0000-00-00' date value.

  See Obtaining Auto-Increment Values, and the description for the FLAG_AUTO_IS_NULL option at Connector/ODBC Connection Parameters.

- IS NOT NULL

  Tests whether a value is not NULL.

  ```
  mysql> SELECT 1 IS NOT NULL, 0 IS NOT NULL, NULL IS NOT NULL;
          -> 1, 1, 0
  ```

- *expr* BETWEEN *min* AND *max*

  If *expr* is greater than or equal to *min* and *expr* is less than or equal to *max*, BETWEEN returns 1, otherwise it returns 0. This is equivalent to the expression (*min* <= *expr* AND *expr* <= *max*) if all the arguments are of the same type. Otherwise type conversion takes place according to the rules described in Section 12.2, “Type Conversion in Expression Evaluation”, but applied to all the three arguments.

  ```
  mysql> SELECT 2 BETWEEN 1 AND 3, 2 BETWEEN 3 and 1;
          -> 1, 0
  mysql> SELECT 1 BETWEEN 2 AND 3;
          -> 0
  mysql> SELECT 'b' BETWEEN 'a' AND 'c';
          -> 1
  mysql> SELECT 2 BETWEEN 2 AND '3';
          -> 1
  mysql> SELECT 2 BETWEEN 2 AND 'x-3';
          -> 0
  ```

  For best results when using BETWEEN with date or time values, use CAST() to explicitly convert the values to the desired data type. Examples: If you compare a DATETIME to two DATE values, convert the DATE values to DATETIME values. If you use a string constant such as '2001-1-1' in a comparison to a DATE, cast the string to a DATE.

- *expr* NOT BETWEEN *min* AND *max*

  This is the same as NOT (*expr* BETWEEN *min* AND *max*).

- COALESCE(*value*,...)

  Returns the first non-NULL value in the list, or NULL if there are no non-NULL values.

  ```
  mysql> SELECT COALESCE(NULL,1);
          -> 1
  mysql> SELECT COALESCE(NULL,NULL,NULL);
          -> NULL
  ```

- GREATEST(*value1*,*value2*,...)

With two or more arguments, returns the largest (maximum-valued) argument. The arguments are compared using the same rules as for `LEAST()`.

```
mysql> SELECT GREATEST(2,0);
        -> 2
mysql> SELECT GREATEST(34.0,3.0,5.0,767.0);
        -> 767.0
mysql> SELECT GREATEST('B','A','C');
        -> 'C'
```

`GREATEST()` returns `NULL` if any argument is `NULL`.

- `expr IN (value,...)`

  Returns `1` if *`expr`* is equal to any of the values in the `IN` list, else returns `0`. If all values are constants, they are evaluated according to the type of *`expr`* and sorted. The search for the item then is done using a binary search. This means `IN` is very quick if the `IN` value list consists entirely of constants. Otherwise, type conversion takes place according to the rules described in Section 12.2, "Type Conversion in Expression Evaluation", but applied to all the arguments.

  ```
  mysql> SELECT 2 IN (0,3,5,7);
          -> 0
  mysql> SELECT 'wefwf' IN ('wee','wefwf','weg');
          -> 1
  ```

  You should never mix quoted and unquoted values in an `IN` list because the comparison rules for quoted values (such as strings) and unquoted values (such as numbers) differ. Mixing types may therefore lead to inconsistent results. For example, do not write an `IN` expression like this:

  ```
  SELECT val1 FROM tbl1 WHERE val1 IN (1,2,'a');
  ```

  Instead, write it like this:

  ```
  SELECT val1 FROM tbl1 WHERE val1 IN ('1','2','a');
  ```

  The number of values in the `IN` list is only limited by the `max_allowed_packet` value.

  To comply with the SQL standard, `IN` returns `NULL` not only if the expression on the left hand side is `NULL`, but also if no match is found in the list and one of the expressions in the list is `NULL`.

  `IN()` syntax can also be used to write certain types of subqueries. See Section 13.2.10.3, "Subqueries with `ANY`, `IN`, or `SOME`".

- `expr NOT IN (value,...)`

  This is the same as `NOT (expr IN (value,...))`.

- `ISNULL(expr)`

  If *`expr`* is `NULL`, `ISNULL()` returns `1`, otherwise it returns `0`.

  ```
  mysql> SELECT ISNULL(1+1);
          -> 0
  mysql> SELECT ISNULL(1/0);
          -> 1
  ```

`ISNULL()` can be used instead of `=` to test whether a value is `NULL`. (Comparing a value to `NULL` using `=` always yields false.)

The `ISNULL()` function shares some special behaviors with the `IS NULL` comparison operator. See the description of `IS NULL`.

- `INTERVAL(N,N1,N2,N3,...)`

  Returns `0` if *`N`* < *`N1`*, `1` if *`N`* < *`N2`* and so on or `-1` if *`N`* is `NULL`. All arguments are treated as integers. It is required that *`N1`* < *`N2`* < *`N3`* < `...` < *`Nn`* for this function to work correctly. This is because a binary search is used (very fast).

  ```
  mysql> SELECT INTERVAL(23, 1, 15, 17, 30, 44, 200);
          -> 3
  mysql> SELECT INTERVAL(10, 1, 10, 100, 1000);
          -> 2
  mysql> SELECT INTERVAL(22, 23, 30, 44, 200);
          -> 0
  ```

- `LEAST(value1,value2,...)`

  With two or more arguments, returns the smallest (minimum-valued) argument. The arguments are compared using the following rules:

  - If any argument is `NULL`, the result is `NULL`. No comparison is needed.
  - If the return value is used in an `INTEGER` context or all arguments are integer-valued, they are compared as integers.
  - If the return value is used in a `REAL` context or all arguments are real-valued, they are compared as reals.
  - If the arguments comprise a mix of numbers and strings, they are compared as numbers.
  - If any argument is a nonbinary (character) string, the arguments are compared as nonbinary strings.
  - In all other cases, the arguments are compared as binary strings.

  ```
  mysql> SELECT LEAST(2,0);
          -> 0
  mysql> SELECT LEAST(34.0,3.0,5.0,767.0);
          -> 3.0
  mysql> SELECT LEAST('B','A','C');
          -> 'A'
  ```

  Note that the preceding conversion rules can produce strange results in some borderline cases:

  ```
  mysql> SELECT CAST(LEAST(3600, 9223372036854775808.0) as SIGNED);
          -> -9223372036854775808
  ```

  This happens because MySQL reads `9223372036854775808.0` in an integer context. The integer representation is not good enough to hold the value, so it wraps to a signed integer.

12.3.3 Logical Operators

Table 12.4 Logical Operators

Name	Description
AND, &&	Logical AND
NOT, !	Negates value
||, OR	Logical OR
XOR	Logical XOR

In SQL, all logical operators evaluate to TRUE, FALSE, or NULL (UNKNOWN). In MySQL, these are implemented as 1 (TRUE), 0 (FALSE), and NULL. Most of this is common to different SQL database servers, although some servers may return any nonzero value for TRUE.

MySQL evaluates any nonzero, non-NULL value to TRUE. For example, the following statements all assess to TRUE:

```
mysql> SELECT 10 IS TRUE;
-> 1
mysql> SELECT -10 IS TRUE;
-> 1
mysql> SELECT 'string' IS NOT NULL;
-> 1
```

- NOT, !

  Logical NOT. Evaluates to 1 if the operand is 0, to 0 if the operand is nonzero, and NOT NULL returns NULL.

  ```
  mysql> SELECT NOT 10;
          -> 0
  mysql> SELECT NOT 0;
          -> 1
  mysql> SELECT NOT NULL;
          -> NULL
  mysql> SELECT ! (1+1);
          -> 0
  mysql> SELECT ! 1+1;
          -> 1
  ```

  The last example produces 1 because the expression evaluates the same way as (!1)+1.

- AND, &&

  Logical AND. Evaluates to 1 if all operands are nonzero and not NULL, to 0 if one or more operands are 0, otherwise NULL is returned.

  ```
  mysql> SELECT 1 && 1;
          -> 1
  mysql> SELECT 1 && 0;
          -> 0
  mysql> SELECT 1 && NULL;
          -> NULL
  mysql> SELECT 0 && NULL;
          -> 0
  mysql> SELECT NULL && 0;
          -> 0
  ```

- OR, ||

Logical OR. When both operands are non-NULL, the result is 1 if any operand is nonzero, and 0 otherwise. With a NULL operand, the result is 1 if the other operand is nonzero, and NULL otherwise. If both operands are NULL, the result is NULL.

```
mysql> SELECT 1 || 1;
        -> 1
mysql> SELECT 1 || 0;
        -> 1
mysql> SELECT 0 || 0;
        -> 0
mysql> SELECT 0 || NULL;
        -> NULL
mysql> SELECT 1 || NULL;
        -> 1
```

- XOR

  Logical XOR. Returns NULL if either operand is NULL. For non-NULL operands, evaluates to 1 if an odd number of operands is nonzero, otherwise 0 is returned.

```
mysql> SELECT 1 XOR 1;
        -> 0
mysql> SELECT 1 XOR 0;
        -> 1
mysql> SELECT 1 XOR NULL;
        -> NULL
mysql> SELECT 1 XOR 1 XOR 1;
        -> 1
```

  a XOR b is mathematically equal to (a AND (NOT b)) OR ((NOT a) and b).

12.3.4 Assignment Operators

Table 12.5 Assignment Operators

Name	Description
=	Assign a value (as part of a SET statement, or as part of the SET clause in an UPDATE statement)
:=	Assign a value

- :=

  Assignment operator. Causes the user variable on the left hand side of the operator to take on the value to its right. The value on the right hand side may be a literal value, another variable storing a value, or any legal expression that yields a scalar value, including the result of a query (provided that this value is a scalar value). You can perform multiple assignments in the same SET statement. You can perform multiple assignments in the same statement-

  Unlike =, the := operator is never interpreted as a comparison operator. This means you can use := in any valid SQL statement (not just in SET statements) to assign a value to a variable.

```
mysql> SELECT @var1, @var2;
        -> NULL, NULL
mysql> SELECT @var1 := 1, @var2;
        -> 1, NULL
mysql> SELECT @var1, @var2;
        -> 1, NULL
```

```
mysql> SELECT @var1, @var2 := @var1;
        -> 1, 1
mysql> SELECT @var1, @var2;
        -> 1, 1

mysql> SELECT @var1:=COUNT(*) FROM t1;
        -> 4
mysql> SELECT @var1;
        -> 4
```

You can make value assignments using `:=` in other statements besides `SELECT`, such as `UPDATE`, as shown here:

```
mysql> SELECT @var1;
        -> 4
mysql> SELECT * FROM t1;
        -> 1, 3, 5, 7

mysql> UPDATE t1 SET c1 = 2 WHERE c1 = @var1:= 1;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT @var1;
        -> 1
mysql> SELECT * FROM t1;
        -> 2, 3, 5, 7
```

While it is also possible both to set and to read the value of the same variable in a single SQL statement using the `:=` operator, this is not recommended. Section 9.4, “User-Defined Variables”, explains why you should avoid doing this.

- =

  This operator is used to perform value assignments in two cases, described in the next two paragraphs.

  Within a `SET` statement, = is treated as an assignment operator that causes the user variable on the left hand side of the operator to take on the value to its right. (In other words, when used in a `SET` statement, = is treated identically to `:=`.) The value on the right hand side may be a literal value, another variable storing a value, or any legal expression that yields a scalar value, including the result of a query (provided that this value is a scalar value). You can perform multiple assignments in the same `SET` statement.

  In the `SET` clause of an `UPDATE` statement, = also acts as an assignment operator; in this case, however, it causes the column named on the left hand side of the operator to assume the value given to the right, provided any `WHERE` conditions that are part of the `UPDATE` are met. You can make multiple assignments in the same `SET` clause of an `UPDATE` statement.

  In any other context, = is treated as a comparison operator.

```
mysql> SELECT @var1, @var2;
        -> NULL, NULL
mysql> SELECT @var1 := 1, @var2;
        -> 1, NULL
mysql> SELECT @var1, @var2;
        -> 1, NULL
mysql> SELECT @var1, @var2 := @var1;
        -> 1, 1
mysql> SELECT @var1, @var2;
        -> 1, 1
```

For more information, see Section 13.7.4, “`SET` Syntax”, Section 13.2.11, “`UPDATE` Syntax”, and Section 13.2.10, “Subquery Syntax”.

12.4 Control Flow Functions

Table 12.6 Flow Control Operators

Name	Description
`CASE`	Case operator
`IF()`	If/else construct
`IFNULL()`	Null if/else construct
`NULLIF()`	Return NULL if expr1 = expr2

- `CASE value WHEN [compare_value] THEN result [WHEN [compare_value] THEN result ...] [ELSE result] END`

  `CASE WHEN [condition] THEN result [WHEN [condition] THEN result ...] [ELSE result] END`

  The first version returns the *`result`* where *`value=compare_value`*. The second version returns the result for the first condition that is true. If there was no matching result value, the result after `ELSE` is returned, or `NULL` if there is no `ELSE` part.

```
mysql> SELECT CASE 1 WHEN 1 THEN 'one'
    ->     WHEN 2 THEN 'two' ELSE 'more' END;
        -> 'one'
mysql> SELECT CASE WHEN 1>0 THEN 'true' ELSE 'false' END;
        -> 'true'
mysql> SELECT CASE BINARY 'B'
    ->     WHEN 'a' THEN 1 WHEN 'b' THEN 2 END;
        -> NULL
```

  The return type of a `CASE` expression is the compatible aggregated type of all return values, but also depends on the context in which it is used. If used in a string context, the result is returned as a string. If used in a numeric context, the result is returned as a decimal, real, or integer value.

  Note

  The syntax of the `CASE` *expression* shown here differs slightly from that of the SQL `CASE` *statement* described in Section 13.6.5.1, “`CASE` Syntax”, for use inside stored programs. The `CASE` statement cannot have an `ELSE NULL` clause, and it is terminated with `END CASE` instead of `END`.

- `IF(expr1,expr2,expr3)`

  If *`expr1`* is `TRUE` (*`expr1`* `<> 0` and *`expr1`* `<> NULL`) then `IF()` returns *`expr2`*; otherwise it returns *`expr3`*. `IF()` returns a numeric or string value, depending on the context in which it is used.

```
mysql> SELECT IF(1>2,2,3);
        -> 3
mysql> SELECT IF(1<2,'yes','no');
        -> 'yes'
mysql> SELECT IF(STRCMP('test','test1'),'no','yes');
        -> 'no'
```

If only one of *`expr2`* or *`expr3`* is explicitly `NULL`, the result type of the `IF()` function is the type of the non-`NULL` expression.

The default return type of `IF()` (which may matter when it is stored into a temporary table) is calculated as follows.

Expression	Return Value
`expr2` or *`expr3`* returns a string	string
`expr2` or *`expr3`* returns a floating-point value	floating-point
`expr2` or *`expr3`* returns an integer	integer

If *`expr2`* and *`expr3`* are both strings, the result is case sensitive if either string is case sensitive.

> **Note**
>
> There is also an `IF` *statement*, which differs from the `IF()` *function* described here. See Section 13.6.5.2, “`IF` Syntax”.

- `IFNULL(`*`expr1`*`,`*`expr2`*`)`

  If *`expr1`* is not `NULL`, `IFNULL()` returns *`expr1`*; otherwise it returns *`expr2`*. `IFNULL()` returns a numeric or string value, depending on the context in which it is used.

```
mysql> SELECT IFNULL(1,0);
        -> 1
mysql> SELECT IFNULL(NULL,10);
        -> 10
mysql> SELECT IFNULL(1/0,10);
        -> 10
mysql> SELECT IFNULL(1/0,'yes');
        -> 'yes'
```

  The default result value of `IFNULL(`*`expr1`*`,`*`expr2`*`)` is the more “general” of the two expressions, in the order `STRING`, `REAL`, or `INTEGER`. Consider the case of a table based on expressions or where MySQL must internally store a value returned by `IFNULL()` in a temporary table:

```
mysql> CREATE TABLE tmp SELECT IFNULL(1,'test') AS test;
mysql> DESCRIBE tmp;
+-------+--------------+------+-----+---------+-------+
| Field | Type         | Null | Key | Default | Extra |
+-------+--------------+------+-----+---------+-------+
| test  | varbinary(4) | NO   |     |         |       |
+-------+--------------+------+-----+---------+-------+
```

  In this example, the type of the `test` column is `VARBINARY(4)`.

- `NULLIF(`*`expr1`*`,`*`expr2`*`)`

  Returns `NULL` if *`expr1`*` = `*`expr2`* is true, otherwise returns *`expr1`*. This is the same as `CASE WHEN `*`expr1`*` = `*`expr2`*` THEN NULL ELSE `*`expr1`*` END`.

```
mysql> SELECT NULLIF(1,1);
        -> NULL
mysql> SELECT NULLIF(1,2);
```

```
        -> 1
```

Note that MySQL evaluates *expr1* twice if the arguments are not equal.

12.5 String Functions

Table 12.7 String Operators

Name	Description
ASCII()	Return numeric value of left-most character
BIN()	Return a string containing binary representation of a number
BIT_LENGTH()	Return length of argument in bits
CHAR_LENGTH()	Return number of characters in argument
CHAR()	Return the character for each integer passed
CHARACTER_LENGTH()	Synonym for CHAR_LENGTH()
CONCAT_WS()	Return concatenate with separator
CONCAT()	Return concatenated string
ELT()	Return string at index number
EXPORT_SET()	Return a string such that for every bit set in the value bits, you get an on string and for every unset bit, you get an off string
FIELD()	Return the index (position) of the first argument in the subsequent arguments
FIND_IN_SET()	Return the index position of the first argument within the second argument
FORMAT()	Return a number formatted to specified number of decimal places
FROM_BASE64()	Decode to a base-64 string and return result
HEX()	Return a hexadecimal representation of a decimal or string value
INSERT()	Insert a substring at the specified position up to the specified number of characters
INSTR()	Return the index of the first occurrence of substring
LCASE()	Synonym for LOWER()
LEFT()	Return the leftmost number of characters as specified
LENGTH()	Return the length of a string in bytes
LIKE	Simple pattern matching
LOAD_FILE()	Load the named file
LOCATE()	Return the position of the first occurrence of substring
LOWER()	Return the argument in lowercase
LPAD()	Return the string argument, left-padded with the specified string
LTRIM()	Remove leading spaces
MAKE_SET()	Return a set of comma-separated strings that have the corresponding bit in bits set

Name	Description
MATCH [1271]	Perform full-text search
MID()	Return a substring starting from the specified position
NOT LIKE	Negation of simple pattern matching
NOT REGEXP	Negation of REGEXP
OCT()	Return a string containing octal representation of a number
OCTET_LENGTH()	Synonym for LENGTH()
ORD()	Return character code for leftmost character of the argument
POSITION()	Synonym for LOCATE()
QUOTE()	Escape the argument for use in an SQL statement
REGEXP	Pattern matching using regular expressions
REPEAT()	Repeat a string the specified number of times
REPLACE()	Replace occurrences of a specified string
REVERSE()	Reverse the characters in a string
RIGHT()	Return the specified rightmost number of characters
RLIKE	Synonym for REGEXP
RPAD()	Append string the specified number of times
RTRIM()	Remove trailing spaces
SOUNDEX()	Return a soundex string
SOUNDS LIKE	Compare sounds
SPACE()	Return a string of the specified number of spaces
STRCMP()	Compare two strings
SUBSTR()	Return the substring as specified
SUBSTRING_INDEX()	Return a substring from a string before the specified number of occurrences of the delimiter
SUBSTRING()	Return the substring as specified
TO_BASE64()	Return the argument converted to a base-64 string
TRIM()	Remove leading and trailing spaces
UCASE()	Synonym for UPPER()
UNHEX()	Return a string containing hex representation of a number
UPPER()	Convert to uppercase
WEIGHT_STRING()	Return the weight string for a string

String-valued functions return `NULL` if the length of the result would be greater than the value of the `max_allowed_packet` system variable. See Section 8.11.2, “Tuning Server Parameters”.

For functions that operate on string positions, the first position is numbered 1.

For functions that take length arguments, noninteger arguments are rounded to the nearest integer.

- `ASCII(`*`str`*`)`

Returns the numeric value of the leftmost character of the string `str`. Returns `0` if `str` is the empty string. Returns `NULL` if `str` is `NULL`. `ASCII()` works for 8-bit characters.

```
mysql> SELECT ASCII('2');
        -> 50
mysql> SELECT ASCII(2);
        -> 50
mysql> SELECT ASCII('dx');
        -> 100
```

See also the `ORD()` function.

- `BIN(N)`

  Returns a string representation of the binary value of `N`, where `N` is a longlong (`BIGINT`) number. This is equivalent to `CONV(N,10,2)`. Returns `NULL` if `N` is `NULL`.

  ```
  mysql> SELECT BIN(12);
          -> '1100'
  ```

- `BIT_LENGTH(str)`

  Returns the length of the string `str` in bits.

  ```
  mysql> SELECT BIT_LENGTH('text');
          -> 32
  ```

- `CHAR(N,... [USING charset_name])`

  `CHAR()` interprets each argument `N` as an integer and returns a string consisting of the characters given by the code values of those integers. `NULL` values are skipped.

  ```
  mysql> SELECT CHAR(77,121,83,81,'76');
          -> 'MySQL'
  mysql> SELECT CHAR(77,77.3,'77.3');
          -> 'MMM'
  ```

  `CHAR()` arguments larger than 255 are converted into multiple result bytes. For example, `CHAR(256)` is equivalent to `CHAR(1,0)`, and `CHAR(256*256)` is equivalent to `CHAR(1,0,0)`:

  ```
  mysql> SELECT HEX(CHAR(1,0)), HEX(CHAR(256));
  +----------------+----------------+
  | HEX(CHAR(1,0)) | HEX(CHAR(256)) |
  +----------------+----------------+
  | 0100           | 0100           |
  +----------------+----------------+
  mysql> SELECT HEX(CHAR(1,0,0)), HEX(CHAR(256*256));
  +------------------+--------------------+
  | HEX(CHAR(1,0,0)) | HEX(CHAR(256*256)) |
  +------------------+--------------------+
  | 010000           | 010000             |
  +------------------+--------------------+
  ```

  By default, `CHAR()` returns a binary string. To produce a string in a given character set, use the optional `USING` clause:

```
mysql> SELECT CHARSET(CHAR(0x65)), CHARSET(CHAR(0x65 USING utf8));
+---------------------+--------------------------------+
| CHARSET(CHAR(0x65)) | CHARSET(CHAR(0x65 USING utf8)) |
+---------------------+--------------------------------+
| binary              | utf8                           |
+---------------------+--------------------------------+
```

If `USING` is given and the result string is illegal for the given character set, a warning is issued. Also, if strict SQL mode is enabled, the result from `CHAR()` becomes `NULL`.

- `CHAR_LENGTH(`*`str`*`)`

  Returns the length of the string *`str`*, measured in characters. A multi-byte character counts as a single character. This means that for a string containing five 2-byte characters, `LENGTH()` returns `10`, whereas `CHAR_LENGTH()` returns `5`.

- `CHARACTER_LENGTH(`*`str`*`)`

  `CHARACTER_LENGTH()` is a synonym for `CHAR_LENGTH()`.

- `CONCAT(`*`str1`*`,`*`str2`*`,...)`

  Returns the string that results from concatenating the arguments. May have one or more arguments. If all arguments are nonbinary strings, the result is a nonbinary string. If the arguments include any binary strings, the result is a binary string. A numeric argument is converted to its equivalent nonbinary string form.

  `CONCAT()` returns `NULL` if any argument is `NULL`.

  ```
  mysql> SELECT CONCAT('My', 'S', 'QL');
          -> 'MySQL'
  mysql> SELECT CONCAT('My', NULL, 'QL');
          -> NULL
  mysql> SELECT CONCAT(14.3);
          -> '14.3'
  ```

  For quoted strings, concatenation can be performed by placing the strings next to each other:

  ```
  mysql> SELECT 'My' 'S' 'QL';
          -> 'MySQL'
  ```

- `CONCAT_WS(`*`separator`*`,`*`str1`*`,`*`str2`*`,...)`

  `CONCAT_WS()` stands for Concatenate With Separator and is a special form of `CONCAT()`. The first argument is the separator for the rest of the arguments. The separator is added between the strings to be concatenated. The separator can be a string, as can the rest of the arguments. If the separator is `NULL`, the result is `NULL`.

  ```
  mysql> SELECT CONCAT_WS(',','First name','Second name','Last Name');
          -> 'First name,Second name,Last Name'
  mysql> SELECT CONCAT_WS(',','First name',NULL,'Last Name');
          -> 'First name,Last Name'
  ```

  `CONCAT_WS()` does not skip empty strings. However, it does skip any `NULL` values after the separator argument.

- `ELT(`*`N`*`,`*`str1`*`,`*`str2`*`,`*`str3`*`,...)`

`ELT()` returns the *`N`*th element of the list of strings: *`str1`* if *`N`* = `1`, *`str2`* if *`N`* = `2`, and so on. Returns `NULL` if *`N`* is less than `1` or greater than the number of arguments. `ELT()` is the complement of `FIELD()`.

```
mysql> SELECT ELT(1, 'ej', 'Heja', 'hej', 'foo');
        -> 'ej'
mysql> SELECT ELT(4, 'ej', 'Heja', 'hej', 'foo');
        -> 'foo'
```

- `EXPORT_SET(bits,on,off[,separator[,number_of_bits]])`

  Returns a string such that for every bit set in the value *`bits`*, you get an *`on`* string and for every bit not set in the value, you get an *`off`* string. Bits in *`bits`* are examined from right to left (from low-order to high-order bits). Strings are added to the result from left to right, separated by the *`separator`* string (the default being the comma character "`,`"). The number of bits examined is given by *`number_of_bits`*, which has a default of 64 if not specified. *`number_of_bits`* is silently clipped to 64 if larger than 64. It is treated as an unsigned integer, so a value of –1 is effectively the same as 64.

```
mysql> SELECT EXPORT_SET(5,'Y','N',',',4);
        -> 'Y,N,Y,N'
mysql> SELECT EXPORT_SET(6,'1','0',',',10);
        -> '0,1,1,0,0,0,0,0,0,0'
```

- `FIELD(str,str1,str2,str3,...)`

  Returns the index (position) of *`str`* in the *`str1`*, *`str2`*, *`str3`*, `...` list. Returns `0` if *`str`* is not found.

  If all arguments to `FIELD()` are strings, all arguments are compared as strings. If all arguments are numbers, they are compared as numbers. Otherwise, the arguments are compared as double.

  If *`str`* is `NULL`, the return value is `0` because `NULL` fails equality comparison with any value. `FIELD()` is the complement of `ELT()`.

```
mysql> SELECT FIELD('ej', 'Hej', 'ej', 'Heja', 'hej', 'foo');
        -> 2
mysql> SELECT FIELD('fo', 'Hej', 'ej', 'Heja', 'hej', 'foo');
        -> 0
```

- `FIND_IN_SET(str,strlist)`

  Returns a value in the range of 1 to *`N`* if the string *`str`* is in the string list *`strlist`* consisting of *`N`* substrings. A string list is a string composed of substrings separated by "`,`" characters. If the first argument is a constant string and the second is a column of type `SET`, the `FIND_IN_SET()` function is optimized to use bit arithmetic. Returns `0` if *`str`* is not in *`strlist`* or if *`strlist`* is the empty string. Returns `NULL` if either argument is `NULL`. This function does not work properly if the first argument contains a comma ("`,`") character.

```
mysql> SELECT FIND_IN_SET('b','a,b,c,d');
        -> 2
```

- `FORMAT(X,D[,locale])`

  Formats the number *`X`* to a format like `'#,###,###.##'`, rounded to *`D`* decimal places, and returns the result as a string. If *`D`* is `0`, the result has no decimal point or fractional part.

The optional third parameter enables a locale to be specified to be used for the result number's decimal point, thousands separator, and grouping between separators. Permissible locale values are the same as the legal values for the `lc_time_names` system variable (see Section 10.7, “MySQL Server Locale Support”). If no locale is specified, the default is `'en_US'`.

```
mysql> SELECT FORMAT(12332.123456, 4);
        -> '12,332.1235'
mysql> SELECT FORMAT(12332.1,4);
        -> '12,332.1000'
mysql> SELECT FORMAT(12332.2,0);
        -> '12,332'
mysql> SELECT FORMAT(12332.2,2,'de_DE');
        -> '12.332,20'
```

- `FROM_BASE64(str)`

  Takes a string encoded with the base-64 encoded rules used by `TO_BASE64()` and returns the decoded result as a binary string. The result is `NULL` if the argument is `NULL` or not a valid base-64 string. See the description of `TO_BASE64()` for details about the encoding and decoding rules.

  ```
  mysql> SELECT TO_BASE64('abc'), FROM_BASE64(TO_BASE64('abc'));
          -> 'JWJj', 'abc'
  ```

- `HEX(str)`, `HEX(N)`

  For a string argument *str*, `HEX()` returns a hexadecimal string representation of *str* where each byte of each character in *str* is converted to two hexadecimal digits. (Multi-byte characters therefore become more than two digits.) The inverse of this operation is performed by the `UNHEX()` function.

  For a numeric argument *N*, `HEX()` returns a hexadecimal string representation of the value of *N* treated as a longlong (`BIGINT`) number. This is equivalent to `CONV(N,10,16)`. The inverse of this operation is performed by `CONV(HEX(N),16,10)`.

  ```
  mysql> SELECT 0x616263, HEX('abc'), UNHEX(HEX('abc'));
          -> 'abc', 616263, 'abc'
  mysql> SELECT HEX(255), CONV(HEX(255),16,10);
          -> 'FF', 255
  ```

- `INSERT(str,pos,len,newstr)`

  Returns the string *str*, with the substring beginning at position *pos* and *len* characters long replaced by the string *newstr*. Returns the original string if *pos* is not within the length of the string. Replaces the rest of the string from position *pos* if *len* is not within the length of the rest of the string. Returns `NULL` if any argument is `NULL`.

  ```
  mysql> SELECT INSERT('Quadratic', 3, 4, 'What');
          -> 'QuWhattic'
  mysql> SELECT INSERT('Quadratic', -1, 4, 'What');
          -> 'Quadratic'
  mysql> SELECT INSERT('Quadratic', 3, 100, 'What');
          -> 'QuWhat'
  ```

  This function is multi-byte safe.

- `INSTR(str,substr)`

Returns the position of the first occurrence of substring *`substr`* in string *`str`*. This is the same as the two-argument form of `LOCATE()`, except that the order of the arguments is reversed.

```
mysql> SELECT INSTR('foobarbar', 'bar');
        -> 4
mysql> SELECT INSTR('xbar', 'foobar');
        -> 0
```

This function is multi-byte safe, and is case sensitive only if at least one argument is a binary string.

- `LCASE(`*`str`*`)`

  `LCASE()` is a synonym for `LOWER()`.

  In MySQL 5.7, `LCASE()` used in a view is rewritten as `LOWER()` when storing the view's definition. (Bug #12844279)

- `LEFT(`*`str`*`,`*`len`*`)`

  Returns the leftmost *`len`* characters from the string *`str`*, or `NULL` if any argument is `NULL`.

  ```
  mysql> SELECT LEFT('foobarbar', 5);
          -> 'fooba'
  ```

  This function is multi-byte safe.

- `LENGTH(`*`str`*`)`

  Returns the length of the string *`str`*, measured in bytes. A multi-byte character counts as multiple bytes. This means that for a string containing five 2-byte characters, `LENGTH()` returns `10`, whereas `CHAR_LENGTH()` returns `5`.

  ```
  mysql> SELECT LENGTH('text');
          -> 4
  ```

- `LOAD_FILE(`*`file_name`*`)`

  Reads the file and returns the file contents as a string. To use this function, the file must be located on the server host, you must specify the full path name to the file, and you must have the `FILE` privilege. The file must be readable by all and its size less than `max_allowed_packet` bytes. If the `secure_file_priv` system variable is set to a nonempty directory name, the file to be loaded must be located in that directory.

  If the file does not exist or cannot be read because one of the preceding conditions is not satisfied, the function returns `NULL`.

  The `character_set_filesystem` system variable controls interpretation of file names that are given as literal strings.

  ```
  mysql> UPDATE t
              SET blob_col=LOAD_FILE('/tmp/picture')
              WHERE id=1;
  ```

- `LOCATE(`*`substr`*`,`*`str`*`)`, `LOCATE(`*`substr`*`,`*`str`*`,`*`pos`*`)`

The first syntax returns the position of the first occurrence of substring *`substr`* in string *`str`*. The second syntax returns the position of the first occurrence of substring *`substr`* in string *`str`*, starting at position *`pos`*. Returns `0` if *`substr`* is not in *`str`*.

```
mysql> SELECT LOCATE('bar', 'foobarbar');
        -> 4
mysql> SELECT LOCATE('xbar', 'foobar');
        -> 0
mysql> SELECT LOCATE('bar', 'foobarbar', 5);
        -> 7
```

This function is multi-byte safe, and is case-sensitive only if at least one argument is a binary string.

- `LOWER(`*`str`*`)`

  Returns the string *`str`* with all characters changed to lowercase according to the current character set mapping. The default is `latin1` (cp1252 West European).

  ```
  mysql> SELECT LOWER('QUADRATICALLY');
          -> 'quadratically'
  ```

  `LOWER()` (and `UPPER()`) are ineffective when applied to binary strings (`BINARY`, `VARBINARY`, `BLOB`). To perform lettercase conversion, convert the string to a nonbinary string:

  ```
  mysql> SET @str = BINARY 'New York';
  mysql> SELECT LOWER(@str), LOWER(CONVERT(@str USING latin1));
  +-------------+-----------------------------------+
  | LOWER(@str) | LOWER(CONVERT(@str USING latin1)) |
  +-------------+-----------------------------------+
  | New York    | new york                          |
  +-------------+-----------------------------------+
  ```

  For Unicode character sets, `LOWER()` and `UPPER()` work accounting to Unicode Collation Algorithm (UCA) 5.2.0 for *`xxx`*`_unicode_520_ci` collations and for language-specific collations that are derived from them. For other Unicode collations, `LOWER()` and `UPPER()` work accounting to Unicode Collation Algorithm (UCA) 4.0.0. See Section 10.1.14.1, “Unicode Character Sets”.

  This function is multi-byte safe.

  In previous versions of MySQL, `LOWER()` used within a view was rewritten as `LCASE()` when storing the view's definition. In MySQL 5.7, `LOWER()` is never rewritten in such cases, but `LCASE()` used within views is instead rewritten as `LOWER()`. (Bug #12844279)

- `LPAD(`*`str`*`,`*`len`*`,`*`padstr`*`)`

  Returns the string *`str`*, left-padded with the string *`padstr`* to a length of *`len`* characters. If *`str`* is longer than *`len`*, the return value is shortened to *`len`* characters.

  ```
  mysql> SELECT LPAD('hi',4,'??');
          -> '??hi'
  mysql> SELECT LPAD('hi',1,'??');
          -> 'h'
  ```

- `LTRIM(`*`str`*`)`

  Returns the string *`str`* with leading space characters removed.

```
mysql> SELECT LTRIM('  barbar');
        -> 'barbar'
```

This function is multi-byte safe.

- `MAKE_SET(bits,str1,str2,...)`

  Returns a set value (a string containing substrings separated by “`,`” characters) consisting of the strings that have the corresponding bit in *`bits`* set. *`str1`* corresponds to bit 0, *`str2`* to bit 1, and so on. `NULL` values in *`str1`*, *`str2`*, `...` are not appended to the result.

```
mysql> SELECT MAKE_SET(1,'a','b','c');
        -> 'a'
mysql> SELECT MAKE_SET(1 | 4,'hello','nice','world');
        -> 'hello,world'
mysql> SELECT MAKE_SET(1 | 4,'hello','nice',NULL,'world');
        -> 'hello'
mysql> SELECT MAKE_SET(0,'a','b','c');
        -> ''
```

- `MID(str,pos,len)`

  `MID(str,pos,len)` is a synonym for `SUBSTRING(str,pos,len)`.

- `OCT(N)`

  Returns a string representation of the octal value of *`N`*, where *`N`* is a longlong (`BIGINT`) number. This is equivalent to `CONV(N,10,8)`. Returns `NULL` if *`N`* is `NULL`.

```
mysql> SELECT OCT(12);
        -> '14'
```

- `OCTET_LENGTH(str)`

  `OCTET_LENGTH()` is a synonym for `LENGTH()`.

- `ORD(str)`

  If the leftmost character of the string *`str`* is a multi-byte character, returns the code for that character, calculated from the numeric values of its constituent bytes using this formula:

```
  (1st byte code)
+ (2nd byte code * 256)
+ (3rd byte code * 256^2) ...
```

  If the leftmost character is not a multi-byte character, `ORD()` returns the same value as the `ASCII()` function.

```
mysql> SELECT ORD('2');
        -> 50
```

- `POSITION(substr IN str)`

  `POSITION(substr IN str)` is a synonym for `LOCATE(substr,str)`.

- `QUOTE(str)`

Quotes a string to produce a result that can be used as a properly escaped data value in an SQL statement. The string is returned enclosed by single quotation marks and with each instance of backslash ("`\`"), single quote ("`'`"), ASCII `NUL`, and Control+Z preceded by a backslash. If the argument is `NULL`, the return value is the word "NULL" without enclosing single quotation marks.

```
mysql> SELECT QUOTE('Don\'t!');
        -> 'Don\'t!'
mysql> SELECT QUOTE(NULL);
        -> NULL
```

For comparison, see the quoting rules for literal strings and within the C API in Section 9.1.1, "String Literals", and Section 21.8.7.55, "`mysql_real_escape_string()`".

- `REPEAT(`*`str`*`,`*`count`*`)`

  Returns a string consisting of the string *`str`* repeated *`count`* times. If *`count`* is less than 1, returns an empty string. Returns `NULL` if *`str`* or *`count`* are `NULL`.

  ```
  mysql> SELECT REPEAT('MySQL', 3);
          -> 'MySQLMySQLMySQL'
  ```

- `REPLACE(`*`str`*`,`*`from_str`*`,`*`to_str`*`)`

  Returns the string *`str`* with all occurrences of the string *`from_str`* replaced by the string *`to_str`*. `REPLACE()` performs a case-sensitive match when searching for *`from_str`*.

  ```
  mysql> SELECT REPLACE('www.mysql.com', 'w', 'Ww');
          -> 'WwWwWw.mysql.com'
  ```

  This function is multi-byte safe.

- `REVERSE(`*`str`*`)`

  Returns the string *`str`* with the order of the characters reversed.

  ```
  mysql> SELECT REVERSE('abc');
          -> 'cba'
  ```

  This function is multi-byte safe.

- `RIGHT(`*`str`*`,`*`len`*`)`

  Returns the rightmost *`len`* characters from the string *`str`*, or `NULL` if any argument is `NULL`.

  ```
  mysql> SELECT RIGHT('foobarbar', 4);
          -> 'rbar'
  ```

  This function is multi-byte safe.

- `RPAD(`*`str`*`,`*`len`*`,`*`padstr`*`)`

  Returns the string *`str`*, right-padded with the string *`padstr`* to a length of *`len`* characters. If *`str`* is longer than *`len`*, the return value is shortened to *`len`* characters.

```
mysql> SELECT RPAD('hi',5,'?');
        -> 'hi???'
mysql> SELECT RPAD('hi',1,'?');
        -> 'h'
```

This function is multi-byte safe.

- `RTRIM(str)`

  Returns the string *`str`* with trailing space characters removed.

  ```
  mysql> SELECT RTRIM('barbar   ');
          -> 'barbar'
  ```

  This function is multi-byte safe.

- `SOUNDEX(str)`

  Returns a soundex string from *`str`*. Two strings that sound almost the same should have identical soundex strings. A standard soundex string is four characters long, but the `SOUNDEX()` function returns an arbitrarily long string. You can use `SUBSTRING()` on the result to get a standard soundex string. All nonalphabetic characters in *`str`* are ignored. All international alphabetic characters outside the A-Z range are treated as vowels.

  Important

  When using `SOUNDEX()`, you should be aware of the following limitations:

  - This function, as currently implemented, is intended to work well with strings that are in the English language only. Strings in other languages may not produce reliable results.

  - This function is not guaranteed to provide consistent results with strings that use multi-byte character sets, including `utf-8`.

    We hope to remove these limitations in a future release. See Bug #22638 for more information.

  ```
  mysql> SELECT SOUNDEX('Hello');
          -> 'H400'
  mysql> SELECT SOUNDEX('Quadratically');
          -> 'Q36324'
  ```

  Note

  This function implements the original Soundex algorithm, not the more popular enhanced version (also described by D. Knuth). The difference is that original version discards vowels first and duplicates second, whereas the enhanced version discards duplicates first and vowels second.

- `expr1 SOUNDS LIKE expr2`

  This is the same as `SOUNDEX(expr1) = SOUNDEX(expr2)`.

- `SPACE(N)`

  Returns a string consisting of *`N`* space characters.

  ```
  mysql> SELECT SPACE(6);
  ```

```
        -> '      '
```

- `SUBSTR(str,pos)`, `SUBSTR(str FROM pos)`, `SUBSTR(str,pos,len)`, `SUBSTR(str FROM pos FOR len)`

  `SUBSTR()` is a synonym for `SUBSTRING()`.

- `SUBSTRING(str,pos)`, `SUBSTRING(str FROM pos)`, `SUBSTRING(str,pos,len)`, `SUBSTRING(str FROM pos FOR len)`

  The forms without a *`len`* argument return a substring from string *`str`* starting at position *`pos`*. The forms with a *`len`* argument return a substring *`len`* characters long from string *`str`*, starting at position *`pos`*. The forms that use `FROM` are standard SQL syntax. It is also possible to use a negative value for *`pos`*. In this case, the beginning of the substring is *`pos`* characters from the end of the string, rather than the beginning. A negative value may be used for *`pos`* in any of the forms of this function.

  For all forms of `SUBSTRING()`, the position of the first character in the string from which the substring is to be extracted is reckoned as `1`.

  ```
  mysql> SELECT SUBSTRING('Quadratically',5);
          -> 'ratically'
  mysql> SELECT SUBSTRING('foobarbar' FROM 4);
          -> 'barbar'
  mysql> SELECT SUBSTRING('Quadratically',5,6);
          -> 'ratica'
  mysql> SELECT SUBSTRING('Sakila', -3);
          -> 'ila'
  mysql> SELECT SUBSTRING('Sakila', -5, 3);
          -> 'aki'
  mysql> SELECT SUBSTRING('Sakila' FROM -4 FOR 2);
          -> 'ki'
  ```

  This function is multi-byte safe.

  If *`len`* is less than 1, the result is the empty string.

- `SUBSTRING_INDEX(str,delim,count)`

  Returns the substring from string *`str`* before *`count`* occurrences of the delimiter *`delim`*. If *`count`* is positive, everything to the left of the final delimiter (counting from the left) is returned. If *`count`* is negative, everything to the right of the final delimiter (counting from the right) is returned. `SUBSTRING_INDEX()` performs a case-sensitive match when searching for *`delim`*.

  ```
  mysql> SELECT SUBSTRING_INDEX('www.mysql.com', '.', 2);
          -> 'www.mysql'
  mysql> SELECT SUBSTRING_INDEX('www.mysql.com', '.', -2);
          -> 'mysql.com'
  ```

  This function is multi-byte safe.

- `TO_BASE64(str)`

  Converts the string argument to base-64 encoded form and returns the result as a character string with the connection character set and collation. If the argument is not a string, it is converted to a string before conversion takes place. The result is `NULL` if the argument is `NULL`. Base-64 encoded strings can be decoded using the the `FROM_BASE64()` function.

  ```
  mysql> SELECT TO_BASE64('abc'), FROM_BASE64(TO_BASE64('abc'));
  ```

```
        -> 'JWJj', 'abc'
```

Different base-64 encoding schemes exist. These are the encoding and decoding rules used by `TO_BASE64()` and `FROM_BASE64()`:

- The encoding for alphabet value 62 is `'+'`.
- The encoding for alphabet value 63 is `'/'`.
- Encoded output consists of groups of 4 printable characters. Each 3 bytes of the input data are encoded using 4 characters. If the last group is incomplete, it is padded with `'='` characters to a length of 4.
- A newline is added after each 76 characters of encoded output to divide long output into multiple lines.
- Decoding recognizes and ignores newline, carriage return, tab, and space.

- `TRIM([{BOTH | LEADING | TRAILING} [`*`remstr`*`] FROM] `*`str`*`)`, `TRIM([`*`remstr`*` FROM] `*`str`*`)`

  Returns the string *`str`* with all *`remstr`* prefixes or suffixes removed. If none of the specifiers `BOTH`, `LEADING`, or `TRAILING` is given, `BOTH` is assumed. *`remstr`* is optional and, if not specified, spaces are removed.

```
mysql> SELECT TRIM('  bar   ');
        -> 'bar'
mysql> SELECT TRIM(LEADING 'x' FROM 'xxxbarxxx');
        -> 'barxxx'
mysql> SELECT TRIM(BOTH 'x' FROM 'xxxbarxxx');
        -> 'bar'
mysql> SELECT TRIM(TRAILING 'xyz' FROM 'barxxyz');
        -> 'barx'
```

  This function is multi-byte safe.

- `UCASE(`*`str`*`)`

  `UCASE()` is a synonym for `UPPER()`.

  In previous versions of MySQL, `UPPER()` used within a view was rewritten as `UCASE()` when storing the view's definition. In MySQL 5.7, `UPPER()` is never rewritten in such cases, but `UCASE()` used within views is instead rewritten as `UPPER()`. (Bug #12844279)

  In MySQL 5.7, `UCASE()` used in a view is rewritten as `UPPER()` when storing the view's definition. (Bug #12844279)

- `UNHEX(`*`str`*`)`

  For a string argument *`str`*, `UNHEX(`*`str`*`)` interprets each pair of characters in the argument as a hexadecimal number and converts it to the byte represented by the number. The return value is a binary string.

```
mysql> SELECT UNHEX('4D7953514C');
        -> 'MySQL'
mysql> SELECT 0x4D7953514C;
        -> 'MySQL'
mysql> SELECT UNHEX(HEX('string'));
        -> 'string'
```

```
mysql> SELECT HEX(UNHEX('1267'));
        -> '1267'
```

The characters in the argument string must be legal hexadecimal digits: `'0'` .. `'9'`, `'A'` .. `'F'`, `'a'` .. `'f'`. If the argument contains any nonhexadecimal digits, the result is `NULL`:

```
mysql> SELECT UNHEX('GG');
+-------------+
| UNHEX('GG') |
+-------------+
| NULL        |
+-------------+
```

A `NULL` result can occur if the argument to `UNHEX()` is a `BINARY` column, because values are padded with 0x00 bytes when stored but those bytes are not stripped on retrieval. For example, `'41'` is stored into a `CHAR(3)` column as `'41 '` and retrieved as `'41'` (with the trailing pad space stripped), so `UNHEX()` for the column value returns `'A'`. By contrast `'41'` is stored into a `BINARY(3)` column as `'41\0'` and retrieved as `'41\0'` (with the trailing pad `0x00` byte not stripped). `'\0'` is not a legal hexadecimal digit, so `UNHEX()` for the column value returns `NULL`.

For a numeric argument *`N`*, the inverse of `HEX(`*`N`*`)` is not performed by `UNHEX()`. Use `CONV(HEX(`*`N`*`),16,10)` instead. See the description of `HEX()`.

- `UPPER(`*`str`*`)`

  Returns the string *`str`* with all characters changed to uppercase according to the current character set mapping. The default is `latin1` (cp1252 West European).

  ```
  mysql> SELECT UPPER('Hej');
          -> 'HEJ'
  ```

  See the description of `LOWER()` for information that also applies to `UPPER()`. This included information about how to perform lettercase conversion of binary strings (`BINARY`, `VARBINARY`, `BLOB`) for which these functions are ineffective, and information about case folding for Unicode character sets.

  This function is multi-byte safe.

  In previous versions of MySQL, `UPPER()` used within a view was rewritten as `UCASE()` when storing the view's definition. In MySQL 5.7, `UPPER()` is never rewritten in such cases, but `UCASE()` used within views is instead rewritten as `UPPER()`. (Bug #12844279)

- `WEIGHT_STRING(`*`str`*` [AS {CHAR|BINARY}(`*`N`*`)] [LEVEL `*`levels`*`] [`*`flags`*`])`

  `levels``: `*`N`*` [ASC|DESC|REVERSE] [, `*`N`*` [ASC|DESC|REVERSE]] ...`

  This function returns the weight string for the input string. The return value is a binary string that represents the sorting and comparison value of the string. It has these properties:

  - If `WEIGHT_STRING(`*`str1`*`)` = `WEIGHT_STRING(`*`str2`*`)`, then *`str1`*` = `*`str2`* (*`str1`* and *`str2`* are considered equal)

  - If `WEIGHT_STRING(`*`str1`*`)` < `WEIGHT_STRING(`*`str2`*`)`, then *`str1`*` < `*`str2`* (*`str1`* sorts before *`str2`*)

  `WEIGHT_STRING()` can be used for testing and debugging of collations, especially if you are adding a new collation. See Section 10.4, “Adding a Collation to a Character Set”.

The input string, *`str`*, is a string expression. If the input is a nonbinary (character) string such as a `CHAR`, `VARCHAR`, or `TEXT` value, the return value contains the collation weights for the string. If the input is a binary (byte) string such as a `BINARY`, `VARBINARY`, or `BLOB` value, the return value is the same as the input (the weight for each byte in a binary string is the byte value). If the input is `NULL`, `WEIGHT_STRING()` returns `NULL`.

Examples:

```
mysql> SET @s = _latin1 'AB' COLLATE latin1_swedish_ci;
mysql> SELECT @s, HEX(@s), HEX(WEIGHT_STRING(@s));
+------+---------+------------------------+
| @s   | HEX(@s) | HEX(WEIGHT_STRING(@s)) |
+------+---------+------------------------+
| AB   | 4142    | 4142                   |
+------+---------+------------------------+
```

```
mysql> SET @s = _latin1 'ab' COLLATE latin1_swedish_ci;
mysql> SELECT @s, HEX(@s), HEX(WEIGHT_STRING(@s));
+------+---------+------------------------+
| @s   | HEX(@s) | HEX(WEIGHT_STRING(@s)) |
+------+---------+------------------------+
| ab   | 6162    | 4142                   |
+------+---------+------------------------+
```

```
mysql> SET @s = CAST('AB' AS BINARY);
mysql> SELECT @s, HEX(@s), HEX(WEIGHT_STRING(@s));
+------+---------+------------------------+
| @s   | HEX(@s) | HEX(WEIGHT_STRING(@s)) |
+------+---------+------------------------+
| AB   | 4142    | 4142                   |
+------+---------+------------------------+
```

```
mysql> SET @s = CAST('ab' AS BINARY);
mysql> SELECT @s, HEX(@s), HEX(WEIGHT_STRING(@s));
+------+---------+------------------------+
| @s   | HEX(@s) | HEX(WEIGHT_STRING(@s)) |
+------+---------+------------------------+
| ab   | 6162    | 6162                   |
+------+---------+------------------------+
```

The preceding examples use `HEX()` to display the `WEIGHT_STRING()` result. Because the result is a binary value, `HEX()` can be especially useful when the result contains nonprinting values, to display it in printable form:

```
mysql> SET @s = CONVERT(0xC39F USING utf8) COLLATE utf8_czech_ci;
mysql> SELECT HEX(WEIGHT_STRING(@s));
+------------------------+
| HEX(WEIGHT_STRING(@s)) |
+------------------------+
| 0FEA0FEA               |
+------------------------+
```

For non-`NULL` return values, the data type of the value is `VARBINARY` if its length is within the maximum length for `VARBINARY`, otherwise the data type is `BLOB`.

The `AS` clause may be given to cast the input string to a nonbinary or binary string and to force it to a given length:

- `AS CHAR(N)` casts the string to a nonbinary string and pads it on the right with spaces to a length of *N* characters. *N* must be at least 1. If *N* is less than the length of the input string, the string is truncated to *N* characters. No warning occurs for truncation.

- `AS BINARY(N)` is similar but casts the string to a binary string, *N* is measured in bytes (not characters), and padding uses `0x00` bytes (not spaces).

```
mysql> SELECT HEX(WEIGHT_STRING('ab' AS CHAR(4)));
+-------------------------------------+
| HEX(WEIGHT_STRING('ab' AS CHAR(4))) |
+-------------------------------------+
| 41422020                            |
+-------------------------------------+
```

```
mysql> SELECT HEX(WEIGHT_STRING('ab' AS BINARY(4)));
+---------------------------------------+
| HEX(WEIGHT_STRING('ab' AS BINARY(4))) |
+---------------------------------------+
| 61620000                              |
+---------------------------------------+
```

The `LEVEL` clause may be given to specify that the return value should contain weights for specific collation levels.

The *`levels`* specifier following the `LEVEL` keyword may be given either as a list of one or more integers separated by commas, or as a range of two integers separated by a dash. Whitespace around the punctuation characters does not matter.

Examples:

```
LEVEL 1
LEVEL 2, 3, 5
LEVEL 1-3
```

Any level less than 1 is treated as 1. Any level greater than the maximum for the input string collation is treated as maximum for the collation. The maximum varies per collation, but is never greater than 6.

In a list of levels, levels must be given in increasing order. In a range of levels, if the second number is less than the first, it is treated as the first number (for example, 4-2 is the same as 4-4).

If the `LEVEL` clause is omitted, MySQL assumes `LEVEL 1 - max`, where *`max`* is the maximum level for the collation.

If `LEVEL` is specified using list syntax (not range syntax), any level number can be followed by these modifiers:

- `ASC`: Return the weights without modification. This is the default.

- `DESC`: Return bitwise-inverted weights (for example, `0x78f0 DESC` = `0x870f`).

- `REVERSE`: Return the weights in reverse order (that is,the weights for the reversed string, with the first character last and the last first).

Examples:

```
mysql> SELECT HEX(WEIGHT_STRING(0x007fff LEVEL 1));
+--------------------------------------+
| HEX(WEIGHT_STRING(0x007fff LEVEL 1)) |
+--------------------------------------+
| 007FFF                               |
+--------------------------------------+
```

```
mysql> SELECT HEX(WEIGHT_STRING(0x007fff LEVEL 1 DESC));
+-------------------------------------------+
| HEX(WEIGHT_STRING(0x007fff LEVEL 1 DESC)) |
+-------------------------------------------+
| FF8000                                    |
+-------------------------------------------+
```

```
mysql> SELECT HEX(WEIGHT_STRING(0x007fff LEVEL 1 REVERSE));
+----------------------------------------------+
| HEX(WEIGHT_STRING(0x007fff LEVEL 1 REVERSE)) |
+----------------------------------------------+
| FF7F00                                       |
+----------------------------------------------+
```

```
mysql> SELECT HEX(WEIGHT_STRING(0x007fff LEVEL 1 DESC REVERSE));
+---------------------------------------------------+
| HEX(WEIGHT_STRING(0x007fff LEVEL 1 DESC REVERSE)) |
+---------------------------------------------------+
| 0080FF                                            |
+---------------------------------------------------+
```

The *flags* clause currently is unused.

12.5.1 String Comparison Functions

Table 12.8 String Comparison Operators

Name	Description
LIKE	Simple pattern matching
NOT LIKE	Negation of simple pattern matching
STRCMP()	Compare two strings

If a string function is given a binary string as an argument, the resulting string is also a binary string. A number converted to a string is treated as a binary string. This affects only comparisons.

Normally, if any expression in a string comparison is case sensitive, the comparison is performed in case-sensitive fashion.

- *expr* LIKE *pat* [ESCAPE '*escape_char*']

  Pattern matching using SQL simple regular expression comparison. Returns 1 (TRUE) or 0 (FALSE). If either *expr* or *pat* is NULL, the result is NULL.

  The pattern need not be a literal string. For example, it can be specified as a string expression or table column.

  Per the SQL standard, LIKE performs matching on a per-character basis, thus it can produce results different from the = comparison operator:

```
mysql> SELECT 'ä' LIKE 'ae' COLLATE latin1_german2_ci;
+-----------------------------------------+
| 'ä' LIKE 'ae' COLLATE latin1_german2_ci |
+-----------------------------------------+
|                                       0 |
+-----------------------------------------+
mysql> SELECT 'ä' = 'ae' COLLATE latin1_german2_ci;
+--------------------------------------+
| 'ä' = 'ae' COLLATE latin1_german2_ci |
+--------------------------------------+
|                                    1 |
+--------------------------------------+
```

In particular, trailing spaces are significant, which is not true for CHAR or VARCHAR comparisons performed with the = operator:

```
mysql> SELECT 'a' = 'a ', 'a' LIKE 'a ';
+------------+---------------+
| 'a' = 'a ' | 'a' LIKE 'a ' |
+------------+---------------+
|          1 |             0 |
+------------+---------------+
1 row in set (0.00 sec)
```

With LIKE you can use the following two wildcard characters in the pattern.

Character	Description
%	Matches any number of characters, even zero characters
_	Matches exactly one character

```
mysql> SELECT 'David!' LIKE 'David_';
        -> 1
mysql> SELECT 'David!' LIKE '%D%v%';
        -> 1
```

To test for literal instances of a wildcard character, precede it by the escape character. If you do not specify the ESCAPE character, “\” is assumed.

String	Description
\%	Matches one “%” character
_	Matches one “_” character

```
mysql> SELECT 'David!' LIKE 'David\_';
        -> 0
mysql> SELECT 'David_' LIKE 'David\_';
        -> 1
```

To specify a different escape character, use the ESCAPE clause:

```
mysql> SELECT 'David_' LIKE 'David|_' ESCAPE '|';
        -> 1
```

The escape sequence should be empty or one character long. The expression must evaluate as a constant at execution time. If the NO_BACKSLASH_ESCAPES SQL mode is enabled, the sequence cannot be empty.

The following two statements illustrate that string comparisons are not case sensitive unless one of the operands is a binary string:

```
mysql> SELECT 'abc' LIKE 'ABC';
        -> 1
mysql> SELECT 'abc' LIKE BINARY 'ABC';
        -> 0
```

In MySQL, `LIKE` is permitted on numeric expressions. (This is an extension to the standard SQL `LIKE`.)

```
mysql> SELECT 10 LIKE '1%';
        -> 1
```

Note

Because MySQL uses C escape syntax in strings (for example, “`\n`” to represent a newline character), you must double any “`\`” that you use in `LIKE` strings. For example, to search for “`\n`”, specify it as “`\\n`”. To search for “`\`”, specify it as “`\\\\`”; this is because the backslashes are stripped once by the parser and again when the pattern match is made, leaving a single backslash to be matched against.

Exception: At the end of the pattern string, backslash can be specified as “`\\`”. At the end of the string, backslash stands for itself because there is nothing following to escape. Suppose that a table contains the following values:

```
mysql> SELECT filename FROM t1;
+--------------+
| filename     |
+--------------+
| C:           |
| C:\          |
| C:\Programs  |
| C:\Programs\ |
+--------------+
```

To test for values that end with backslash, you can match the values using either of the following patterns:

```
mysql> SELECT filename, filename LIKE '%\\' FROM t1;
+--------------+---------------------+
| filename     | filename LIKE '%\\' |
+--------------+---------------------+
| C:           |                   0 |
| C:\          |                   1 |
| C:\Programs  |                   0 |
| C:\Programs\ |                   1 |
+--------------+---------------------+

mysql> SELECT filename, filename LIKE '%\\\\' FROM t1;
+--------------+-----------------------+
| filename     | filename LIKE '%\\\\' |
+--------------+-----------------------+
| C:           |                     0 |
| C:\          |                     1 |
| C:\Programs  |                     0 |
| C:\Programs\ |                     1 |
+--------------+-----------------------+
```

- *expr* NOT LIKE *pat* [ESCAPE '*escape_char*']

  This is the same as NOT (*expr* LIKE *pat* [ESCAPE '*escape_char*']).

  Note

  Aggregate queries involving NOT LIKE comparisons with columns containing NULL may yield unexpected results. For example, consider the following table and data:

  ```
  CREATE TABLE foo (bar VARCHAR(10));

  INSERT INTO foo VALUES (NULL), (NULL);
  ```

  The query SELECT COUNT(*) FROM foo WHERE bar LIKE '%baz%'; returns 0. You might assume that SELECT COUNT(*) FROM foo WHERE bar NOT LIKE '%baz%'; would return 2. However, this is not the case: The second query returns 0. This is because NULL NOT LIKE *expr* always returns NULL, regardless of the value of *expr*. The same is true for aggregate queries involving NULL and comparisons using NOT RLIKE or NOT REGEXP. In such cases, you must test explicitly for NOT NULL using OR (and not AND), as shown here:

  ```
  SELECT COUNT(*) FROM foo WHERE bar NOT LIKE '%baz%' OR bar IS NULL;
  ```

- STRCMP(*expr1*,*expr2*)

  STRCMP() returns 0 if the strings are the same, -1 if the first argument is smaller than the second according to the current sort order, and 1 otherwise.

  ```
  mysql> SELECT STRCMP('text', 'text2');
          -> -1
  mysql> SELECT STRCMP('text2', 'text');
          -> 1
  mysql> SELECT STRCMP('text', 'text');
          -> 0
  ```

  STRCMP() performs the comparison using the collation of the arguments.

  ```
  mysql> SET @s1 = _latin1 'x' COLLATE latin1_general_ci;
  mysql> SET @s2 = _latin1 'X' COLLATE latin1_general_ci;
  mysql> SET @s3 = _latin1 'x' COLLATE latin1_general_cs;
  mysql> SET @s4 = _latin1 'X' COLLATE latin1_general_cs;
  mysql> SELECT STRCMP(@s1, @s2), STRCMP(@s3, @s4);
  +------------------+------------------+
  | STRCMP(@s1, @s2) | STRCMP(@s3, @s4) |
  +------------------+------------------+
  |                0 |                1 |
  +------------------+------------------+
  ```

  If the collations are incompatible, one of the arguments must be converted to be compatible with the other. See Section 10.1.7.5, “Collation of Expressions”.

  ```
  mysql> SELECT STRCMP(@s1, @s3);
  ERROR 1267 (HY000): Illegal mix of collations (latin1_general_ci,IMPLICIT) and (latin1_general_cs,IMPLIC
  mysql> SELECT STRCMP(@s1, @s3 COLLATE latin1_general_ci);
  +--------------------------------------------+
  ```

```
| STRCMP(@s1, @s3 COLLATE latin1_general_ci) |
+--------------------------------------------+
|                                          0 |
+--------------------------------------------+
```

12.5.2 Regular Expressions

Table 12.9 String Regular Expression Operators

Name	Description
NOT REGEXP	Negation of REGEXP
REGEXP	Pattern matching using regular expressions
RLIKE	Synonym for REGEXP

A regular expression is a powerful way of specifying a pattern for a complex search.

MySQL uses Henry Spencer's implementation of regular expressions, which is aimed at conformance with POSIX 1003.2. MySQL uses the extended version to support pattern-matching operations performed with the REGEXP operator in SQL statements.

This section summarizes, with examples, the special characters and constructs that can be used in MySQL for REGEXP operations. It does not contain all the details that can be found in Henry Spencer's regex(7) manual page. That manual page is included in MySQL source distributions, in the regex.7 file under the regex directory. See also Section 3.3.4.7, “Pattern Matching”.

Regular Expression Operators

- *expr* NOT REGEXP *pat*, *expr* NOT RLIKE *pat*

  This is the same as NOT (*expr* REGEXP *pat*).

- *expr* REGEXP *pat*, *expr* RLIKE *pat*

  Performs a pattern match of a string expression *expr* against a pattern *pat*. The pattern can be an extended regular expression, the syntax for which is discussed later in this section. Returns 1 if *expr* matches *pat*; otherwise it returns 0. If either *expr* or *pat* is NULL, the result is NULL. RLIKE is a synonym for REGEXP, provided for mSQL compatibility.

  The pattern need not be a literal string. For example, it can be specified as a string expression or table column.

  Note

  Because MySQL uses the C escape syntax in strings (for example, “\n” to represent the newline character), you must double any “\” that you use in your REGEXP strings.

  REGEXP is not case sensitive, except when used with binary strings.

```
mysql> SELECT 'Monty!' REGEXP '.*';
        -> 1
mysql> SELECT 'new*\n*line' REGEXP 'new\\*.\\*line';
        -> 1
mysql> SELECT 'a' REGEXP 'A', 'a' REGEXP BINARY 'A';
        -> 1  0
mysql> SELECT 'a' REGEXP '^[a-d]';
        -> 1
```

REGEXP and RLIKE use the character set and collations of the arguments when deciding the type of a character and performing the comparison. If the arguments have different character sets or collations, coercibility rules apply as described in Section 10.1.7.5, “Collation of Expressions”.

Warning

The REGEXP and RLIKE operators work in byte-wise fashion, so they are not multi-byte safe and may produce unexpected results with multi-byte character sets. In addition, these operators compare characters by their byte values and accented characters may not compare as equal even if a given collation treats them as equal.

Syntax of Regular Expressions

A regular expression describes a set of strings. The simplest regular expression is one that has no special characters in it. For example, the regular expression hello matches hello and nothing else.

Nontrivial regular expressions use certain special constructs so that they can match more than one string. For example, the regular expression hello|word matches either the string hello or the string word.

As a more complex example, the regular expression B[an]*s matches any of the strings Bananas, Baaaaas, Bs, and any other string starting with a B, ending with an s, and containing any number of a or n characters in between.

A regular expression for the REGEXP operator may use any of the following special characters and constructs:

- ^

  Match the beginning of a string.

  ```
  mysql> SELECT 'fo\nfo' REGEXP '^fo$';                   -> 0
  mysql> SELECT 'fofo' REGEXP '^fo';                      -> 1
  ```

- $

  Match the end of a string.

  ```
  mysql> SELECT 'fo\no' REGEXP '^fo\no$';                 -> 1
  mysql> SELECT 'fo\no' REGEXP '^fo$';                    -> 0
  ```

- .

  Match any character (including carriage return and newline).

  ```
  mysql> SELECT 'fofo' REGEXP '^f.*$';                    -> 1
  mysql> SELECT 'fo\r\nfo' REGEXP '^f.*$';                -> 1
  ```

- a*

  Match any sequence of zero or more a characters.

  ```
  mysql> SELECT 'Ban' REGEXP '^Ba*n';                     -> 1
  mysql> SELECT 'Baaan' REGEXP '^Ba*n';                   -> 1
  mysql> SELECT 'Bn' REGEXP '^Ba*n';                      -> 1
  ```

- a+

  Match any sequence of one or more a characters.

  ```
  mysql> SELECT 'Ban' REGEXP '^Ba+n';                     -> 1
  mysql> SELECT 'Bn' REGEXP '^Ba+n';                      -> 0
  ```

- a?

  Match either zero or one a character.

  ```
  mysql> SELECT 'Bn' REGEXP '^Ba?n';                      -> 1
  mysql> SELECT 'Ban' REGEXP '^Ba?n';                     -> 1
  mysql> SELECT 'Baan' REGEXP '^Ba?n';                    -> 0
  ```

- de|abc

  Match either of the sequences de or abc.

  ```
  mysql> SELECT 'pi' REGEXP 'pi|apa';                     -> 1
  mysql> SELECT 'axe' REGEXP 'pi|apa';                    -> 0
  mysql> SELECT 'apa' REGEXP 'pi|apa';                    -> 1
  mysql> SELECT 'apa' REGEXP '^(pi|apa)$';                -> 1
  mysql> SELECT 'pi' REGEXP '^(pi|apa)$';                 -> 1
  mysql> SELECT 'pix' REGEXP '^(pi|apa)$';                -> 0
  ```

- (abc)*

  Match zero or more instances of the sequence abc.

  ```
  mysql> SELECT 'pi' REGEXP '^(pi)*$';                    -> 1
  mysql> SELECT 'pip' REGEXP '^(pi)*$';                   -> 0
  mysql> SELECT 'pipi' REGEXP '^(pi)*$';                  -> 1
  ```

- {1}, {2,3}

  {n} or {m,n} notation provides a more general way of writing regular expressions that match many occurrences of the previous atom (or “piece”) of the pattern. m and n are integers.

  - a*

    Can be written as a{0,}.

  - a+

    Can be written as a{1,}.

  - a?

    Can be written as a{0,1}.

  To be more precise, a{n} matches exactly n instances of a. a{n,} matches n or more instances of a. a{m,n} matches m through n instances of a, inclusive.

  m and n must be in the range from 0 to RE_DUP_MAX (default 255), inclusive. If both m and n are given, m must be less than or equal to n.

```
mysql> SELECT 'abcde' REGEXP 'a[bcd]{2}e';              -> 0
mysql> SELECT 'abcde' REGEXP 'a[bcd]{3}e';              -> 1
mysql> SELECT 'abcde' REGEXP 'a[bcd]{1,10}e';           -> 1
```

- `[a-dX]`, `[^a-dX]`

  Matches any character that is (or is not, if ^ is used) either `a`, `b`, `c`, `d` or `X`. A `-` character between two other characters forms a range that matches all characters from the first character to the second. For example, `[0-9]` matches any decimal digit. To include a literal `]` character, it must immediately follow the opening bracket `[`. To include a literal `-` character, it must be written first or last. Any character that does not have a defined special meaning inside a `[]` pair matches only itself.

```
mysql> SELECT 'aXbc' REGEXP '[a-dXYZ]';                 -> 1
mysql> SELECT 'aXbc' REGEXP '^[a-dXYZ]$';               -> 0
mysql> SELECT 'aXbc' REGEXP '^[a-dXYZ]+$';              -> 1
mysql> SELECT 'aXbc' REGEXP '^[^a-dXYZ]+$';             -> 0
mysql> SELECT 'gheis' REGEXP '^[^a-dXYZ]+$';            -> 1
mysql> SELECT 'gheisa' REGEXP '^[^a-dXYZ]+$';           -> 0
```

- `[.characters.]`

  Within a bracket expression (written using `[` and `]`), matches the sequence of characters of that collating element. `characters` is either a single character or a character name like `newline`. The following table lists the permissible character names.

  The following table shows the permissible character names and the characters that they match. For characters given as numeric values, the values are represented in octal.

Name	Character	Name	Character
`NUL`	`0`	`SOH`	`001`
`STX`	`002`	`ETX`	`003`
`EOT`	`004`	`ENQ`	`005`
`ACK`	`006`	`BEL`	`007`
`alert`	`007`	`BS`	`010`
`backspace`	`'\b'`	`HT`	`011`
`tab`	`'\t'`	`LF`	`012`
`newline`	`'\n'`	`VT`	`013`
`vertical-tab`	`'\v'`	`FF`	`014`
`form-feed`	`'\f'`	`CR`	`015`
`carriage-return`	`'\r'`	`SO`	`016`
`SI`	`017`	`DLE`	`020`
`DC1`	`021`	`DC2`	`022`
`DC3`	`023`	`DC4`	`024`
`NAK`	`025`	`SYN`	`026`
`ETB`	`027`	`CAN`	`030`
`EM`	`031`	`SUB`	`032`
`ESC`	`033`	`IS4`	`034`
`FS`	`034`	`IS3`	`035`

Name	Character	Name	Character
GS	035	IS2	036
RS	036	IS1	037
US	037	space	' '
exclamation-mark	'!'	quotation-mark	'"'
number-sign	'#'	dollar-sign	'$'
percent-sign	'%'	ampersand	'&'
apostrophe	'\''	left-parenthesis	'('
right-parenthesis	')'	asterisk	'*'
plus-sign	'+'	comma	','
hyphen	'-'	hyphen-minus	'-'
period	'.'	full-stop	'.'
slash	'/'	solidus	'/'
zero	'0'	one	'1'
two	'2'	three	'3'
four	'4'	five	'5'
six	'6'	seven	'7'
eight	'8'	nine	'9'
colon	':'	semicolon	';'
less-than-sign	'<'	equals-sign	'='
greater-than-sign	'>'	question-mark	'?'
commercial-at	'@'	left-square-bracket	'['
backslash	'\\'	reverse-solidus	'\\'
right-square-bracket	']'	circumflex	'^'
circumflex-accent	'^'	underscore	'_'
low-line	'_'	grave-accent	'`'
left-brace	'{'	left-curly-bracket	'{'
vertical-line	'|'	right-brace	'}'
right-curly-bracket	'}'	tilde	'~'
DEL	177		

```
mysql> SELECT '~' REGEXP '[[.~.]]';                     -> 1
mysql> SELECT '~' REGEXP '[[.tilde.]]';                 -> 1
```

- [=character_class=]

  Within a bracket expression (written using [and]), [=character_class=] represents an equivalence class. It matches all characters with the same collation value, including itself. For example, if `o` and `(+)` are the members of an equivalence class, `[[=o=]]`, `[[=(+)=]]`, and `[o(+)]` are all synonymous. An equivalence class may not be used as an endpoint of a range.

- `[:character_class:]`

  Within a bracket expression (written using `[` and `]`), `[:character_class:]` represents a character class that matches all characters belonging to that class. The following table lists the standard class names. These names stand for the character classes defined in the `ctype(3)` manual page. A particular locale may provide other class names. A character class may not be used as an endpoint of a range.

Character Class Name	Meaning
`alnum`	Alphanumeric characters
`alpha`	Alphabetic characters
`blank`	Whitespace characters
`cntrl`	Control characters
`digit`	Digit characters
`graph`	Graphic characters
`lower`	Lowercase alphabetic characters
`print`	Graphic or space characters
`punct`	Punctuation characters
`space`	Space, tab, newline, and carriage return
`upper`	Uppercase alphabetic characters
`xdigit`	Hexadecimal digit characters

```
mysql> SELECT 'justalnums' REGEXP '[[:alnum:]]+';       -> 1
mysql> SELECT '!!' REGEXP '[[:alnum:]]+';               -> 0
```

- `[[:<:]]`, `[[:>:]]`

  These markers stand for word boundaries. They match the beginning and end of words, respectively. A word is a sequence of word characters that is not preceded by or followed by word characters. A word character is an alphanumeric character in the `alnum` class or an underscore (`_`).

```
mysql> SELECT 'a word a' REGEXP '[[:<:]]word[[:>:]]';   -> 1
mysql> SELECT 'a xword a' REGEXP '[[:<:]]word[[:>:]]';  -> 0
```

To use a literal instance of a special character in a regular expression, precede it by two backslash (\) characters. The MySQL parser interprets one of the backslashes, and the regular expression library interprets the other. For example, to match the string `1+2` that contains the special `+` character, only the last of the following regular expressions is the correct one:

```
mysql> SELECT '1+2' REGEXP '1+2';                       -> 0
mysql> SELECT '1+2' REGEXP '1\+2';                      -> 0
mysql> SELECT '1+2' REGEXP '1\\+2';                     -> 1
```

12.6 Numeric Functions and Operators

Table 12.10 Numeric Functions and Operators

Name	Description
ABS()	Return the absolute value
ACOS()	Return the arc cosine
ASIN()	Return the arc sine
ATAN2(), ATAN()	Return the arc tangent of the two arguments
ATAN()	Return the arc tangent
CEIL()	Return the smallest integer value not less than the argument
CEILING()	Return the smallest integer value not less than the argument
CONV()	Convert numbers between different number bases
COS()	Return the cosine
COT()	Return the cotangent
CRC32()	Compute a cyclic redundancy check value
DEGREES()	Convert radians to degrees
DIV	Integer division
/	Division operator
EXP()	Raise to the power of
FLOOR()	Return the largest integer value not greater than the argument
LN()	Return the natural logarithm of the argument
LOG10()	Return the base-10 logarithm of the argument
LOG2()	Return the base-2 logarithm of the argument
LOG()	Return the natural logarithm of the first argument
-	Minus operator
MOD()	Return the remainder
% or MOD	Modulo operator
PI()	Return the value of pi
+	Addition operator
POW()	Return the argument raised to the specified power
POWER()	Return the argument raised to the specified power
RADIANS()	Return argument converted to radians
RAND()	Return a random floating-point value
ROUND()	Round the argument
SIGN()	Return the sign of the argument
SIN()	Return the sine of the argument
SQRT()	Return the square root of the argument
TAN()	Return the tangent of the argument
*	Multiplication operator

Name	Description
TRUNCATE()	Truncate to specified number of decimal places
-	Change the sign of the argument

12.6.1 Arithmetic Operators

Table 12.11 Arithmetic Operators

Name	Description
DIV	Integer division
/	Division operator
-	Minus operator
% or MOD	Modulo operator
+	Addition operator
*	Multiplication operator
-	Change the sign of the argument

The usual arithmetic operators are available. The result is determined according to the following rules:

- In the case of `-`, `+`, and `*`, the result is calculated with `BIGINT` (64-bit) precision if both operands are integers.
- If both operands are integers and any of them are unsigned, the result is an unsigned integer. For subtraction, if the `NO_UNSIGNED_SUBTRACTION` SQL mode is enabled, the result is signed even if any operand is unsigned.
- If any of the operands of a `+`, `-`, `/`, `*`, `%` is a real or string value, the precision of the result is the precision of the operand with the maximum precision.
- In division performed with `/`, the scale of the result when using two exact-value operands is the scale of the first operand plus the value of the `div_precision_increment` system variable (which is 4 by default). For example, the result of the expression `5.05 / 0.014` has a scale of six decimal places (`360.714286`).

These rules are applied for each operation, such that nested calculations imply the precision of each component. Hence, `(14620 / 9432456) / (24250 / 9432456)`, resolves first to `(0.0014) / (0.0026)`, with the final result having 8 decimal places (`0.60288653`).

Because of these rules and the way they are applied, care should be taken to ensure that components and subcomponents of a calculation use the appropriate level of precision. See Section 12.10, “Cast Functions and Operators”.

For information about handling of overflow in numeric expression evaluation, see Section 11.2.6, “Out-of-Range and Overflow Handling”.

Arithmetic operators apply to numbers. For other types of values, alternative operations may be available. For example, to add date values, use `DATE_ADD()`; see Section 12.7, “Date and Time Functions”.

- `+`

  Addition:

```
mysql> SELECT 3+5;
        -> 8
```

- `-`

  Subtraction:

  ```
  mysql> SELECT 3-5;
          -> -2
  ```

- `-`

  Unary minus. This operator changes the sign of the operand.

  ```
  mysql> SELECT - 2;
          -> -2
  ```

  Note

  If this operator is used with a `BIGINT`, the return value is also a `BIGINT`. This means that you should avoid using `-` on integers that may have the value of –2^{63}.

- `*`

  Multiplication:

  ```
  mysql> SELECT 3*5;
          -> 15
  mysql> SELECT 18014398509481984*18014398509481984.0;
          -> 324518553658426726783156020576256.0
  mysql> SELECT 18014398509481984*18014398509481984;
          -> out-of-range error
  ```

  The last expression produces an error because the result of the integer multiplication exceeds the 64-bit range of `BIGINT` calculations. (See Section 11.2, “Numeric Types”.)

- `/`

  Division:

  ```
  mysql> SELECT 3/5;
          -> 0.60
  ```

  Division by zero produces a `NULL` result:

  ```
  mysql> SELECT 102/(1-1);
          -> NULL
  ```

  A division is calculated with `BIGINT` arithmetic only if performed in a context where its result is converted to an integer.

- `DIV`

  Integer division. Similar to `FLOOR()`, but is safe with `BIGINT` values.

  In MySQL 5.7, if either operand has a noninteger type, the operands are converted to `DECIMAL` and divided using `DECIMAL` arithmetic before converting the result to `BIGINT`. If the result exceeds `BIGINT` range, an error occurs.

```
mysql> SELECT 5 DIV 2;
        -> 2
```

- *`N`* `%` *`M`*, *`N`* `MOD` *`M`*

  Modulo operation. Returns the remainder of *`N`* divided by *`M`*. For more information, see the description for the `MOD()` function in Section 12.6.2, “Mathematical Functions”.

12.6.2 Mathematical Functions

Table 12.12 Mathematical Functions

Name	Description
`ABS()`	Return the absolute value
`ACOS()`	Return the arc cosine
`ASIN()`	Return the arc sine
`ATAN2()`, `ATAN()`	Return the arc tangent of the two arguments
`ATAN()`	Return the arc tangent
`CEIL()`	Return the smallest integer value not less than the argument
`CEILING()`	Return the smallest integer value not less than the argument
`CONV()`	Convert numbers between different number bases
`COS()`	Return the cosine
`COT()`	Return the cotangent
`CRC32()`	Compute a cyclic redundancy check value
`DEGREES()`	Convert radians to degrees
`EXP()`	Raise to the power of
`FLOOR()`	Return the largest integer value not greater than the argument
`LN()`	Return the natural logarithm of the argument
`LOG10()`	Return the base-10 logarithm of the argument
`LOG2()`	Return the base-2 logarithm of the argument
`LOG()`	Return the natural logarithm of the first argument
`MOD()`	Return the remainder
`PI()`	Return the value of pi
`POW()`	Return the argument raised to the specified power
`POWER()`	Return the argument raised to the specified power
`RADIANS()`	Return argument converted to radians
`RAND()`	Return a random floating-point value
`ROUND()`	Round the argument
`SIGN()`	Return the sign of the argument
`SIN()`	Return the sine of the argument
`SQRT()`	Return the square root of the argument
`TAN()`	Return the tangent of the argument
`TRUNCATE()`	Truncate to specified number of decimal places

All mathematical functions return `NULL` in the event of an error.

- `ABS(X)`

  Returns the absolute value of *X*.

  ```
  mysql> SELECT ABS(2);
          -> 2
  mysql> SELECT ABS(-32);
          -> 32
  ```

  This function is safe to use with `BIGINT` values.

- `ACOS(X)`

  Returns the arc cosine of *X*, that is, the value whose cosine is *X*. Returns `NULL` if *X* is not in the range `-1` to `1`.

  ```
  mysql> SELECT ACOS(1);
          -> 0
  mysql> SELECT ACOS(1.0001);
          -> NULL
  mysql> SELECT ACOS(0);
          -> 1.5707963267949
  ```

- `ASIN(X)`

  Returns the arc sine of *X*, that is, the value whose sine is *X*. Returns `NULL` if *X* is not in the range `-1` to `1`.

  ```
  mysql> SELECT ASIN(0.2);
          -> 0.20135792079033
  mysql> SELECT ASIN('foo');

  +-------------+
  | ASIN('foo') |
  +-------------+
  |           0 |
  +-------------+
  1 row in set, 1 warning (0.00 sec)

  mysql> SHOW WARNINGS;
  +---------+------+-----------------------------------------+
  | Level   | Code | Message                                 |
  +---------+------+-----------------------------------------+
  | Warning | 1292 | Truncated incorrect DOUBLE value: 'foo' |
  +---------+------+-----------------------------------------+
  ```

- `ATAN(X)`

  Returns the arc tangent of *X*, that is, the value whose tangent is *X*.

  ```
  mysql> SELECT ATAN(2);
          -> 1.1071487177941
  mysql> SELECT ATAN(-2);
          -> -1.1071487177941
  ```

- `ATAN(Y,X)`, `ATAN2(Y,X)`

  Returns the arc tangent of the two variables *X* and *Y*. It is similar to calculating the arc tangent of `Y / X`, except that the signs of both arguments are used to determine the quadrant of the result.

```
mysql> SELECT ATAN(-2,2);
        -> -0.78539816339745
mysql> SELECT ATAN2(PI(),0);
        -> 1.5707963267949
```

- `CEIL(X)`

  `CEIL()` is a synonym for `CEILING()`.

- `CEILING(X)`

  Returns the smallest integer value not less than *`X`*.

  ```
  mysql> SELECT CEILING(1.23);
          -> 2
  mysql> SELECT CEILING(-1.23);
          -> -1
  ```

  For exact-value numeric arguments, the return value has an exact-value numeric type. For string or floating-point arguments, the return value has a floating-point type.

- `CONV(N,from_base,to_base)`

  Converts numbers between different number bases. Returns a string representation of the number *`N`*, converted from base *`from_base`* to base *`to_base`*. Returns `NULL` if any argument is `NULL`. The argument *`N`* is interpreted as an integer, but may be specified as an integer or a string. The minimum base is `2` and the maximum base is `36`. If *`to_base`* is a negative number, *`N`* is regarded as a signed number. Otherwise, *`N`* is treated as unsigned. `CONV()` works with 64-bit precision.

  ```
  mysql> SELECT CONV('a',16,2);
          -> '1010'
  mysql> SELECT CONV('6E',18,8);
          -> '172'
  mysql> SELECT CONV(-17,10,-18);
          -> '-H'
  mysql> SELECT CONV(10+'10'+'10'+0xa,10,10);
          -> '40'
  ```

- `COS(X)`

  Returns the cosine of *`X`*, where *`X`* is given in radians.

  ```
  mysql> SELECT COS(PI());
          -> -1
  ```

- `COT(X)`

  Returns the cotangent of *`X`*.

  ```
  mysql> SELECT COT(12);
          -> -1.5726734063977
  mysql> SELECT COT(0);
          -> NULL
  ```

- `CRC32(expr)`

Computes a cyclic redundancy check value and returns a 32-bit unsigned value. The result is `NULL` if the argument is `NULL`. The argument is expected to be a string and (if possible) is treated as one if it is not.

```
mysql> SELECT CRC32('MySQL');
        -> 3259397556
mysql> SELECT CRC32('mysql');
        -> 2501908538
```

- `DEGREES(X)`

  Returns the argument *X*, converted from radians to degrees.

  ```
  mysql> SELECT DEGREES(PI());
          -> 180
  mysql> SELECT DEGREES(PI() / 2);
          -> 90
  ```

- `EXP(X)`

  Returns the value of *e* (the base of natural logarithms) raised to the power of *X*. The inverse of this function is `LOG()` (using a single argument only) or `LN()`.

  ```
  mysql> SELECT EXP(2);
          -> 7.3890560989307
  mysql> SELECT EXP(-2);
          -> 0.13533528323661
  mysql> SELECT EXP(0);
          -> 1
  ```

- `FLOOR(X)`

  Returns the largest integer value not greater than *X*.

  ```
  mysql> SELECT FLOOR(1.23);
          -> 1
  mysql> SELECT FLOOR(-1.23);
          -> -2
  ```

  For exact-value numeric arguments, the return value has an exact-value numeric type. For string or floating-point arguments, the return value has a floating-point type.

- `FORMAT(X,D)`

  Formats the number *X* to a format like `'#,###,###.##'`, rounded to *D* decimal places, and returns the result as a string. For details, see Section 12.5, “String Functions”.

- `HEX(N_or_S)`

  This function can be used to obtain a hexadecimal representation of a decimal number or a string; the manner in which it does so varies according to the argument's type. See this function's description in Section 12.5, “String Functions”, for details.

- `LN(X)`

  Returns the natural logarithm of *X*; that is, the base-*e* logarithm of *X*. As of MySQL 5.7.4, if *X* is less than or equal to 0.0E0, the error “Invalid argument for logarithm” is reported in strict SQL mode, and `NULL` is returned in non-strict mode. Before MySQL 5.7.4, if *X* is less than or equal to 0.0E0, `NULL` is returned.

```
mysql> SELECT LN(2);
        -> 0.69314718055995
mysql> SELECT LN(-2);
        -> NULL
```

This function is synonymous with `LOG(X)`. The inverse of this function is the `EXP()` function.

- `LOG(X)`, `LOG(B,X)`

  If called with one parameter, this function returns the natural logarithm of *X*. As of MySQL 5.7.4, if *X* is less than or equal to 0.0E0, the error “Invalid argument for logarithm” is reported in strict SQL mode, and `NULL` is returned in non-strict mode. Before MySQL 5.7.4, if *X* is less than or equal to 0.0E0, `NULL` is returned.

  The inverse of this function (when called with a single argument) is the `EXP()` function.

  ```
  mysql> SELECT LOG(2);
          -> 0.69314718055995
  mysql> SELECT LOG(-2);
          -> NULL
  ```

  If called with two parameters, this function returns the logarithm of *X* to the base *B*. If *X* is less than or equal to 0, or if *B* is less than or equal to 1, then `NULL` is returned.

  ```
  mysql> SELECT LOG(2,65536);
          -> 16
  mysql> SELECT LOG(10,100);
          -> 2
  mysql> SELECT LOG(1,100);
          -> NULL
  ```

  `LOG(B,X)` is equivalent to `LOG(X) / LOG(B)`.

- `LOG2(X)`

  Returns the base-2 logarithm of *X*. As of MySQL 5.7.4, if *X* is less than or equal to 0.0E0, the error “Invalid argument for logarithm” is reported in strict SQL mode, and `NULL` is returned in non-strict mode. Before MySQL 5.7.4, if *X* is less than or equal to 0.0E0, `NULL` is returned.

  ```
  mysql> SELECT LOG2(65536);
          -> 16
  mysql> SELECT LOG2(-100);
          -> NULL
  ```

  `LOG2()` is useful for finding out how many bits a number requires for storage. This function is equivalent to the expression `LOG(X) / LOG(2)`.

- `LOG10(X)`

  Returns the base-10 logarithm of *X*. As of MySQL 5.7.4, if *X* is less than or equal to 0.0E0, the error “Invalid argument for logarithm” is reported in strict SQL mode, and `NULL` is returned in non-strict mode. Before MySQL 5.7.4, if *X* is less than or equal to 0.0E0, `NULL` is returned.

  ```
  mysql> SELECT LOG10(2);
          -> 0.30102999566398
  mysql> SELECT LOG10(100);
  ```

```
        -> 2
mysql> SELECT LOG10(-100);
        -> NULL
```

`LOG10(X)` is equivalent to `LOG(10,X)`.

- `MOD(N,M)`, `N % M`, `N MOD M`

  Modulo operation. Returns the remainder of *N* divided by *M*.

  ```
  mysql> SELECT MOD(234, 10);
          -> 4
  mysql> SELECT 253 % 7;
          -> 1
  mysql> SELECT MOD(29,9);
          -> 2
  mysql> SELECT 29 MOD 9;
          -> 2
  ```

  This function is safe to use with `BIGINT` values.

  `MOD()` also works on values that have a fractional part and returns the exact remainder after division:

  ```
  mysql> SELECT MOD(34.5,3);
          -> 1.5
  ```

  `MOD(N,0)` returns `NULL`.

- `PI()`

  Returns the value of π (pi). The default number of decimal places displayed is seven, but MySQL uses the full double-precision value internally.

  ```
  mysql> SELECT PI();
          -> 3.141593
  mysql> SELECT PI()+0.000000000000000000;
          -> 3.141592653589793116
  ```

- `POW(X,Y)`

  Returns the value of *X* raised to the power of *Y*.

  ```
  mysql> SELECT POW(2,2);
          -> 4
  mysql> SELECT POW(2,-2);
          -> 0.25
  ```

- `POWER(X,Y)`

  This is a synonym for `POW()`.

- `RADIANS(X)`

  Returns the argument *X*, converted from degrees to radians. (Note that π radians equals 180 degrees.)

  ```
  mysql> SELECT RADIANS(90);
          -> 1.5707963267949
  ```

- `RAND()`, `RAND(N)`

  Returns a random floating-point value *v* in the range `0` <= *v* < `1.0`. If a constant integer argument *N* is specified, it is used as the seed value, which produces a repeatable sequence of column values. In the following example, note that the sequences of values produced by `RAND(3)` is the same both places where it occurs.

```
mysql> CREATE TABLE t (i INT);
Query OK, 0 rows affected (0.42 sec)

mysql> INSERT INTO t VALUES(1),(2),(3);
Query OK, 3 rows affected (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> SELECT i, RAND() FROM t;
+------+------------------+
| i    | RAND()           |
+------+------------------+
|    1 | 0.61914388706828 |
|    2 | 0.93845168309142 |
|    3 | 0.83482678498591 |
+------+------------------+
3 rows in set (0.00 sec)

mysql> SELECT i, RAND(3) FROM t;
+------+------------------+
| i    | RAND(3)          |
+------+------------------+
|    1 | 0.90576975597606 |
|    2 | 0.37307905813035 |
|    3 | 0.14808605345719 |
+------+------------------+
3 rows in set (0.00 sec)

mysql> SELECT i, RAND() FROM t;
+------+------------------+
| i    | RAND()           |
+------+------------------+
|    1 | 0.35877890638893 |
|    2 | 0.28941420772058 |
|    3 | 0.37073435016976 |
+------+------------------+
3 rows in set (0.00 sec)

mysql> SELECT i, RAND(3) FROM t;
+------+------------------+
| i    | RAND(3)          |
+------+------------------+
|    1 | 0.90576975597606 |
|    2 | 0.37307905813035 |
|    3 | 0.14808605345719 |
+------+------------------+
3 rows in set (0.01 sec)
```

  With a constant initializer, the seed is initialized once when the statement is compiled, prior to execution. If a nonconstant initializer (such as a column name) is used as the argument, the seed is initialized with the value for each invocation of `RAND()`. (One implication of this is that for equal argument values, `RAND()` will return the same value each time.)

  To obtain a random integer *R* in the range *i* <= *R* < *j*, use the expression `FLOOR(i + RAND() * (j – i))`. For example, to obtain a random integer in the range the range `7` <= *R* < `12`, you could use the following statement:

```
SELECT FLOOR(7 + (RAND() * 5));
```

`RAND()` in a `WHERE` clause is re-evaluated every time the `WHERE` is executed.

You cannot use a column with `RAND()` values in an `ORDER BY` clause, because `ORDER BY` would evaluate the column multiple times. However, you can retrieve rows in random order like this:

```
mysql> SELECT * FROM tbl_name ORDER BY RAND();
```

`ORDER BY RAND()` combined with `LIMIT` is useful for selecting a random sample from a set of rows:

```
mysql> SELECT * FROM table1, table2 WHERE a=b AND c<d -> ORDER BY RAND() LIMIT 1000;
```

`RAND()` is not meant to be a perfect random generator. It is a fast way to generate random numbers on demand that is portable between platforms for the same MySQL version.

This function is unsafe for statement-based replication. A warning is logged if you use this function when `binlog_format` is set to `STATEMENT`. (Bug #49222)

- `ROUND(X)`, `ROUND(X,D)`

  Rounds the argument *X* to *D* decimal places. The rounding algorithm depends on the data type of *X*. *D* defaults to 0 if not specified. *D* can be negative to cause *D* digits left of the decimal point of the value *X* to become zero.

  ```
  mysql> SELECT ROUND(-1.23);
          -> -1
  mysql> SELECT ROUND(-1.58);
          -> -2
  mysql> SELECT ROUND(1.58);
          -> 2
  mysql> SELECT ROUND(1.298, 1);
          -> 1.3
  mysql> SELECT ROUND(1.298, 0);
          -> 1
  mysql> SELECT ROUND(23.298, -1);
          -> 20
  ```

  The return type is the same type as that of the first argument (assuming that it is integer, double, or decimal). This means that for an integer argument, the result is an integer (no decimal places):

  ```
  mysql> SELECT ROUND(150.000,2), ROUND(150,2);
  +------------------+--------------+
  | ROUND(150.000,2) | ROUND(150,2) |
  +------------------+--------------+
  |           150.00 |          150 |
  +------------------+--------------+
  ```

  `ROUND()` uses the following rules depending on the type of the first argument:

  - For exact-value numbers, `ROUND()` uses the “round half away from zero” or “round toward nearest” rule: A value with a fractional part of .5 or greater is rounded up to the next integer if positive or down to the next integer if negative. (In other words, it is rounded away from zero.) A value with a fractional part less than .5 is rounded down to the next integer if positive or up to the next integer if negative.

  - For approximate-value numbers, the result depends on the C library. On many systems, this means that `ROUND()` uses the "round to nearest even" rule: A value with any fractional part is rounded to the nearest even integer.

  The following example shows how rounding differs for exact and approximate values:

  ```
  mysql> SELECT ROUND(2.5), ROUND(25E-1);
  +------------+--------------+
  | ROUND(2.5) | ROUND(25E-1) |
  +------------+--------------+
  | 3          |            2 |
  +------------+--------------+
  ```

  For more information, see Section 12.19, “Precision Math”.

- `SIGN(X)`

  Returns the sign of the argument as `-1`, `0`, or `1`, depending on whether *X* is negative, zero, or positive.

  ```
  mysql> SELECT SIGN(-32);
          -> -1
  mysql> SELECT SIGN(0);
          -> 0
  mysql> SELECT SIGN(234);
          -> 1
  ```

- `SIN(X)`

  Returns the sine of *X*, where *X* is given in radians.

  ```
  mysql> SELECT SIN(PI());
          -> 1.2246063538224e-16
  mysql> SELECT ROUND(SIN(PI()));
          -> 0
  ```

- `SQRT(X)`

  Returns the square root of a nonnegative number *X*.

  ```
  mysql> SELECT SQRT(4);
          -> 2
  mysql> SELECT SQRT(20);
          -> 4.4721359549996
  mysql> SELECT SQRT(-16);
          -> NULL
  ```

- `TAN(X)`

  Returns the tangent of *X*, where *X* is given in radians.

  ```
  mysql> SELECT TAN(PI());
          -> -1.2246063538224e-16
  mysql> SELECT TAN(PI()+1);
          -> 1.5574077246549
  ```

- `TRUNCATE(X,D)`

Returns the number *X*, truncated to *D* decimal places. If *D* is 0, the result has no decimal point or fractional part. *D* can be negative to cause *D* digits left of the decimal point of the value *X* to become zero.

```
mysql> SELECT TRUNCATE(1.223,1);
        -> 1.2
mysql> SELECT TRUNCATE(1.999,1);
        -> 1.9
mysql> SELECT TRUNCATE(1.999,0);
        -> 1
mysql> SELECT TRUNCATE(-1.999,1);
        -> -1.9
mysql> SELECT TRUNCATE(122,-2);
       -> 100
mysql> SELECT TRUNCATE(10.28*100,0);
       -> 1028
```

All numbers are rounded toward zero.

12.7 Date and Time Functions

This section describes the functions that can be used to manipulate temporal values. See Section 11.3, "Date and Time Types", for a description of the range of values each date and time type has and the valid formats in which values may be specified.

Table 12.13 Date/Time Functions

Name	Description
ADDDATE()	Add time values (intervals) to a date value
ADDTIME()	Add time
CONVERT_TZ()	Convert from one timezone to another
CURDATE()	Return the current date
CURRENT_DATE(), CURRENT_DATE	Synonyms for CURDATE()
CURRENT_TIME(), CURRENT_TIME	Synonyms for CURTIME()
CURRENT_TIMESTAMP(), CURRENT_TIMESTAMP	Synonyms for NOW()
CURTIME()	Return the current time
DATE_ADD()	Add time values (intervals) to a date value
DATE_FORMAT()	Format date as specified
DATE_SUB()	Subtract a time value (interval) from a date
DATE()	Extract the date part of a date or datetime expression
DATEDIFF()	Subtract two dates
DAY()	Synonym for DAYOFMONTH()
DAYNAME()	Return the name of the weekday
DAYOFMONTH()	Return the day of the month (0-31)
DAYOFWEEK()	Return the weekday index of the argument
DAYOFYEAR()	Return the day of the year (1-366)
EXTRACT()	Extract part of a date

Name	Description
`FROM_DAYS()`	Convert a day number to a date
`FROM_UNIXTIME()`	Format UNIX timestamp as a date
`GET_FORMAT()`	Return a date format string
`HOUR()`	Extract the hour
`LAST_DAY`	Return the last day of the month for the argument
`LOCALTIME()`, `LOCALTIME`	Synonym for NOW()
`LOCALTIMESTAMP`, `LOCALTIMESTAMP()`	Synonym for NOW()
`MAKEDATE()`	Create a date from the year and day of year
`MAKETIME()`	Create time from hour, minute, second
`MICROSECOND()`	Return the microseconds from argument
`MINUTE()`	Return the minute from the argument
`MONTH()`	Return the month from the date passed
`MONTHNAME()`	Return the name of the month
`NOW()`	Return the current date and time
`PERIOD_ADD()`	Add a period to a year-month
`PERIOD_DIFF()`	Return the number of months between periods
`QUARTER()`	Return the quarter from a date argument
`SEC_TO_TIME()`	Converts seconds to 'HH:MM:SS' format
`SECOND()`	Return the second (0-59)
`STR_TO_DATE()`	Convert a string to a date
`SUBDATE()`	Synonym for DATE_SUB() when invoked with three arguments
`SUBTIME()`	Subtract times
`SYSDATE()`	Return the time at which the function executes
`TIME_FORMAT()`	Format as time
`TIME_TO_SEC()`	Return the argument converted to seconds
`TIME()`	Extract the time portion of the expression passed
`TIMEDIFF()`	Subtract time
`TIMESTAMP()`	With a single argument, this function returns the date or datetime expression; with two arguments, the sum of the arguments
`TIMESTAMPADD()`	Add an interval to a datetime expression
`TIMESTAMPDIFF()`	Subtract an interval from a datetime expression
`TO_DAYS()`	Return the date argument converted to days
`TO_SECONDS()`	Return the date or datetime argument converted to seconds since Year 0
`UNIX_TIMESTAMP()`	Return a UNIX timestamp
`UTC_DATE()`	Return the current UTC date
`UTC_TIME()`	Return the current UTC time

Name	Description
UTC_TIMESTAMP()	Return the current UTC date and time
WEEK()	Return the week number
WEEKDAY()	Return the weekday index
WEEKOFYEAR()	Return the calendar week of the date (0-53)
YEAR()	Return the year
YEARWEEK()	Return the year and week

Here is an example that uses date functions. The following query selects all rows with a *date_col* value from within the last 30 days:

```
mysql> SELECT something FROM tbl_name
    -> WHERE DATE_SUB(CURDATE(),INTERVAL 30 DAY) <= date_col;
```

The query also selects rows with dates that lie in the future.

Functions that expect date values usually accept datetime values and ignore the time part. Functions that expect time values usually accept datetime values and ignore the date part.

Functions that return the current date or time each are evaluated only once per query at the start of query execution. This means that multiple references to a function such as NOW() within a single query always produce the same result. (For our purposes, a single query also includes a call to a stored program (stored routine, trigger, or event) and all subprograms called by that program.) This principle also applies to CURDATE(), CURTIME(), UTC_DATE(), UTC_TIME(), UTC_TIMESTAMP(), and to any of their synonyms.

The CURRENT_TIMESTAMP(), CURRENT_TIME(), CURRENT_DATE(), and FROM_UNIXTIME() functions return values in the connection's current time zone, which is available as the value of the time_zone system variable. In addition, UNIX_TIMESTAMP() assumes that its argument is a datetime value in the current time zone. See Section 10.6, “MySQL Server Time Zone Support”.

Some date functions can be used with “zero” dates or incomplete dates such as '2001-11-00', whereas others cannot. Functions that extract parts of dates typically work with incomplete dates and thus can return 0 when you might otherwise expect a nonzero value. For example:

```
mysql> SELECT DAYOFMONTH('2001-11-00'), MONTH('2005-00-00');
        -> 0, 0
```

Other functions expect complete dates and return NULL for incomplete dates. These include functions that perform date arithmetic or that map parts of dates to names. For example:

```
mysql> SELECT DATE_ADD('2006-05-00',INTERVAL 1 DAY);
        -> NULL
mysql> SELECT DAYNAME('2006-05-00');
        -> NULL
```

Several functions are more strict when passed a DATE() function value as their argument and reject incomplete dates with a day part of zero. These functions are affected: CONVERT_TZ(), DATE_ADD(), DATE_SUB(), DAYOFYEAR(), LAST_DAY() (permits a day part of zero), TIMESTAMPDIFF(), TO_DAYS(), TO_SECONDS(), WEEK(), WEEKDAY(), WEEKOFYEAR(), YEARWEEK().

Fractional seconds for TIME, DATETIME, and TIMESTAMP values are supported, with up to microsecond precision. Functions that take temporal arguments accept values with fractional seconds. Return values from temporal functions include fractional seconds as appropriate.

- ADDDATE(*date*,INTERVAL *expr* *unit*), ADDDATE(*expr*,*days*)

  When invoked with the INTERVAL form of the second argument, ADDDATE() is a synonym for DATE_ADD(). The related function SUBDATE() is a synonym for DATE_SUB(). For information on the INTERVAL *unit* argument, see the discussion for DATE_ADD().

  ```
  mysql> SELECT DATE_ADD('2008-01-02', INTERVAL 31 DAY);
          -> '2008-02-02'
  mysql> SELECT ADDDATE('2008-01-02', INTERVAL 31 DAY);
          -> '2008-02-02'
  ```

  When invoked with the *days* form of the second argument, MySQL treats it as an integer number of days to be added to *expr*.

  ```
  mysql> SELECT ADDDATE('2008-01-02', 31);
          -> '2008-02-02'
  ```

- ADDTIME(*expr1*,*expr2*)

  ADDTIME() adds *expr2* to *expr1* and returns the result. *expr1* is a time or datetime expression, and *expr2* is a time expression.

  ```
  mysql> SELECT ADDTIME('2007-12-31 23:59:59.999999', '1 1:1:1.000002');
          -> '2008-01-02 01:01:01.000001'
  mysql> SELECT ADDTIME('01:00:00.999999', '02:00:00.999998');
          -> '03:00:01.999997'
  ```

- CONVERT_TZ(*dt*,*from_tz*,*to_tz*)

  CONVERT_TZ() converts a datetime value *dt* from the time zone given by *from_tz* to the time zone given by *to_tz* and returns the resulting value. Time zones are specified as described in Section 10.6, “MySQL Server Time Zone Support”. This function returns NULL if the arguments are invalid.

  If the value falls out of the supported range of the TIMESTAMP type when converted from *from_tz* to UTC, no conversion occurs. The TIMESTAMP range is described in Section 11.1.2, “Date and Time Type Overview”.

  ```
  mysql> SELECT CONVERT_TZ('2004-01-01 12:00:00','GMT','MET');
          -> '2004-01-01 13:00:00'
  mysql> SELECT CONVERT_TZ('2004-01-01 12:00:00','+00:00','+10:00');
          -> '2004-01-01 22:00:00'
  ```

  Note

  To use named time zones such as 'MET' or 'Europe/Moscow', the time zone tables must be properly set up. See Section 10.6, “MySQL Server Time Zone Support”, for instructions.

- CURDATE()

  Returns the current date as a value in 'YYYY-MM-DD' or YYYYMMDD format, depending on whether the function is used in a string or numeric context.

  ```
  mysql> SELECT CURDATE();
          -> '2008-06-13'
  mysql> SELECT CURDATE() + 0;
  ```

```
        -> 20080613
```

- `CURRENT_DATE`, `CURRENT_DATE()`

  `CURRENT_DATE` and `CURRENT_DATE()` are synonyms for `CURDATE()`.

- `CURRENT_TIME`, `CURRENT_TIME([fsp])`

  `CURRENT_TIME` and `CURRENT_TIME()` are synonyms for `CURTIME()`.

- `CURRENT_TIMESTAMP`, `CURRENT_TIMESTAMP([fsp])`

  `CURRENT_TIMESTAMP` and `CURRENT_TIMESTAMP()` are synonyms for `NOW()`.

- `CURTIME([fsp])`

  Returns the current time as a value in `'HH:MM:SS'` or `HHMMSS` format, depending on whether the function is used in a string or numeric context. The value is expressed in the current time zone.

  If the *`fsp`* argument is given to specify a fractional seconds precision from 0 to 6, the return value includes a fractional seconds part of that many digits.

```
mysql> SELECT CURTIME();
        -> '23:50:26'
mysql> SELECT CURTIME() + 0;
        -> 235026.000000
```

- `DATE(expr)`

  Extracts the date part of the date or datetime expression *`expr`*.

```
mysql> SELECT DATE('2003-12-31 01:02:03');
        -> '2003-12-31'
```

- `DATEDIFF(expr1,expr2)`

  `DATEDIFF()` returns *`expr1`* – *`expr2`* expressed as a value in days from one date to the other. *`expr1`* and *`expr2`* are date or date-and-time expressions. Only the date parts of the values are used in the calculation.

```
mysql> SELECT DATEDIFF('2007-12-31 23:59:59','2007-12-30');
        -> 1
mysql> SELECT DATEDIFF('2010-11-30 23:59:59','2010-12-31');
        -> -31
```

- `DATE_ADD(date,INTERVAL expr unit)`, `DATE_SUB(date,INTERVAL expr unit)`

  These functions perform date arithmetic. The *`date`* argument specifies the starting date or datetime value. *`expr`* is an expression specifying the interval value to be added or subtracted from the starting date. *`expr`* is a string; it may start with a “`-`” for negative intervals. *`unit`* is a keyword indicating the units in which the expression should be interpreted.

  The `INTERVAL` keyword and the *`unit`* specifier are not case sensitive.

  The following table shows the expected form of the *`expr`* argument for each *`unit`* value.

unit Value	Expected *expr* Format
MICROSECOND	MICROSECONDS
SECOND	SECONDS
MINUTE	MINUTES
HOUR	HOURS
DAY	DAYS
WEEK	WEEKS
MONTH	MONTHS
QUARTER	QUARTERS
YEAR	YEARS
SECOND_MICROSECOND	'SECONDS.MICROSECONDS'
MINUTE_MICROSECOND	'MINUTES:SECONDS.MICROSECONDS'
MINUTE_SECOND	'MINUTES:SECONDS'
HOUR_MICROSECOND	'HOURS:MINUTES:SECONDS.MICROSECONDS'
HOUR_SECOND	'HOURS:MINUTES:SECONDS'
HOUR_MINUTE	'HOURS:MINUTES'
DAY_MICROSECOND	'DAYS HOURS:MINUTES:SECONDS.MICROSECONDS'
DAY_SECOND	'DAYS HOURS:MINUTES:SECONDS'
DAY_MINUTE	'DAYS HOURS:MINUTES'
DAY_HOUR	'DAYS HOURS'
YEAR_MONTH	'YEARS-MONTHS'

The return value depends on the arguments:

- DATETIME if the first argument is a DATETIME (or TIMESTAMP) value, or if the first argument is a DATE and the *unit* value uses HOURS, MINUTES, or SECONDS.
- String otherwise.

To ensure that the result is DATETIME, you can use CAST() to convert the first argument to DATETIME.

MySQL permits any punctuation delimiter in the *expr* format. Those shown in the table are the suggested delimiters. If the *date* argument is a DATE value and your calculations involve only YEAR, MONTH, and DAY parts (that is, no time parts), the result is a DATE value. Otherwise, the result is a DATETIME value.

Date arithmetic also can be performed using INTERVAL together with the + or - operator:

```
date + INTERVAL expr unit
date - INTERVAL expr unit
```

INTERVAL *expr* *unit* is permitted on either side of the + operator if the expression on the other side is a date or datetime value. For the - operator, INTERVAL *expr* *unit* is permitted only on the right side, because it makes no sense to subtract a date or datetime value from an interval.

```
mysql> SELECT '2008-12-31 23:59:59' + INTERVAL 1 SECOND;
        -> '2009-01-01 00:00:00'
mysql> SELECT INTERVAL 1 DAY + '2008-12-31';
        -> '2009-01-01'
mysql> SELECT '2005-01-01' - INTERVAL 1 SECOND;
        -> '2004-12-31 23:59:59'
mysql> SELECT DATE_ADD('2000-12-31 23:59:59',
    ->                 INTERVAL 1 SECOND);
        -> '2001-01-01 00:00:00'
mysql> SELECT DATE_ADD('2010-12-31 23:59:59',
    ->                 INTERVAL 1 DAY);
        -> '2011-01-01 23:59:59'
mysql> SELECT DATE_ADD('2100-12-31 23:59:59',
    ->                 INTERVAL '1:1' MINUTE_SECOND);
        -> '2101-01-01 00:01:00'
mysql> SELECT DATE_SUB('2005-01-01 00:00:00',
    ->                 INTERVAL '1 1:1:1' DAY_SECOND);
        -> '2004-12-30 22:58:59'
mysql> SELECT DATE_ADD('1900-01-01 00:00:00',
    ->                 INTERVAL '-1 10' DAY_HOUR);
        -> '1899-12-30 14:00:00'
mysql> SELECT DATE_SUB('1998-01-02', INTERVAL 31 DAY);
        -> '1997-12-02'
mysql> SELECT DATE_ADD('1992-12-31 23:59:59.000002',
    ->            INTERVAL '1.999999' SECOND_MICROSECOND);
        -> '1993-01-01 00:00:01.000001'
```

If you specify an interval value that is too short (does not include all the interval parts that would be expected from the *`unit`* keyword), MySQL assumes that you have left out the leftmost parts of the interval value. For example, if you specify a *`unit`* of `DAY_SECOND`, the value of *`expr`* is expected to have days, hours, minutes, and seconds parts. If you specify a value like `'1:10'`, MySQL assumes that the days and hours parts are missing and the value represents minutes and seconds. In other words, `'1:10' DAY_SECOND` is interpreted in such a way that it is equivalent to `'1:10' MINUTE_SECOND`. This is analogous to the way that MySQL interprets `TIME` values as representing elapsed time rather than as a time of day.

Because *`expr`* is treated as a string, be careful if you specify a nonstring value with `INTERVAL`. For example, with an interval specifier of `HOUR_MINUTE`, `6/4` evaluates to `1.5000` and is treated as 1 hour, 5000 minutes:

```
mysql> SELECT 6/4;
        -> 1.5000
mysql> SELECT DATE_ADD('2009-01-01', INTERVAL 6/4 HOUR_MINUTE);
        -> '2009-01-04 12:20:00'
```

To ensure interpretation of the interval value as you expect, a `CAST()` operation may be used. To treat `6/4` as 1 hour, 5 minutes, cast it to a `DECIMAL` value with a single fractional digit:

```
mysql> SELECT CAST(6/4 AS DECIMAL(3,1));
        -> 1.5
mysql> SELECT DATE_ADD('1970-01-01 12:00:00',
    ->                 INTERVAL CAST(6/4 AS DECIMAL(3,1)) HOUR_MINUTE);
        -> '1970-01-01 13:05:00'
```

If you add to or subtract from a date value something that contains a time part, the result is automatically converted to a datetime value:

```
mysql> SELECT DATE_ADD('2013-01-01', INTERVAL 1 DAY);
```

```
        -> '2013-01-02'
mysql> SELECT DATE_ADD('2013-01-01', INTERVAL 1 HOUR);
        -> '2013-01-01 01:00:00'
```

If you add `MONTH`, `YEAR_MONTH`, or `YEAR` and the resulting date has a day that is larger than the maximum day for the new month, the day is adjusted to the maximum days in the new month:

```
mysql> SELECT DATE_ADD('2009-01-30', INTERVAL 1 MONTH);
        -> '2009-02-28'
```

Date arithmetic operations require complete dates and do not work with incomplete dates such as `'2006-07-00'` or badly malformed dates:

```
mysql> SELECT DATE_ADD('2006-07-00', INTERVAL 1 DAY);
        -> NULL
mysql> SELECT '2005-03-32' + INTERVAL 1 MONTH;
        -> NULL
```

- `DATE_FORMAT(`*`date`*`,`*`format`*`)`

  Formats the *`date`* value according to the *`format`* string.

  The following specifiers may be used in the *`format`* string. The "`%`" character is required before format specifier characters.

Specifier	Description
`%a`	Abbreviated weekday name (`Sun`..`Sat`)
`%b`	Abbreviated month name (`Jan`..`Dec`)
`%c`	Month, numeric (`0`..`12`)
`%D`	Day of the month with English suffix (`0th`, `1st`, `2nd`, `3rd`, …)
`%d`	Day of the month, numeric (`00`..`31`)
`%e`	Day of the month, numeric (`0`..`31`)
`%f`	Microseconds (`000000`..`999999`)
`%H`	Hour (`00`..`23`)
`%h`	Hour (`01`..`12`)
`%I`	Hour (`01`..`12`)
`%i`	Minutes, numeric (`00`..`59`)
`%j`	Day of year (`001`..`366`)
`%k`	Hour (`0`..`23`)
`%l`	Hour (`1`..`12`)
`%M`	Month name (`January`..`December`)
`%m`	Month, numeric (`00`..`12`)
`%p`	`AM` or `PM`
`%r`	Time, 12-hour (`hh:mm:ss` followed by `AM` or `PM`)
`%S`	Seconds (`00`..`59`)
`%s`	Seconds (`00`..`59`)
`%T`	Time, 24-hour (`hh:mm:ss`)

Specifier	Description
%U	Week (00..53), where Sunday is the first day of the week; WEEK() mode 0
%u	Week (00..53), where Monday is the first day of the week; WEEK() mode 1
%V	Week (01..53), where Sunday is the first day of the week; WEEK() mode 2; used with %X
%v	Week (01..53), where Monday is the first day of the week; WEEK() mode 3; used with %x
%W	Weekday name (Sunday..Saturday)
%w	Day of the week (0=Sunday..6=Saturday)
%X	Year for the week where Sunday is the first day of the week, numeric, four digits; used with %V
%x	Year for the week, where Monday is the first day of the week, numeric, four digits; used with %v
%Y	Year, numeric, four digits
%y	Year, numeric (two digits)
%%	A literal "%" character
%*x*	*x*, for any "*x*" not listed above

Ranges for the month and day specifiers begin with zero due to the fact that MySQL permits the storing of incomplete dates such as '2014-00-00'.

The language used for day and month names and abbreviations is controlled by the value of the lc_time_names system variable (Section 10.7, "MySQL Server Locale Support").

For the %U, %u, %V, and %v specifiers, see the description of the WEEK() function for information about the mode values. The mode affects how week numbering occurs.

DATE_FORMAT() returns a string with a character set and collation given by character_set_connection and collation_connection so that it can return month and weekday names containing non-ASCII characters.

```
mysql> SELECT DATE_FORMAT('2009-10-04 22:23:00', '%W %M %Y');
        -> 'Sunday October 2009'
mysql> SELECT DATE_FORMAT('2007-10-04 22:23:00', '%H:%i:%s');
        -> '22:23:00'
mysql> SELECT DATE_FORMAT('1900-10-04 22:23:00',
    ->                 '%D %y %a %d %m %b %j');
        -> '4th 00 Thu 04 10 Oct 277'
mysql> SELECT DATE_FORMAT('1997-10-04 22:23:00',
    ->                 '%H %k %I %r %T %S %w');
        -> '22 22 10 10:23:00 PM 22:23:00 00 6'
mysql> SELECT DATE_FORMAT('1999-01-01', '%X %V');
        -> '1998 52'
mysql> SELECT DATE_FORMAT('2006-06-00', '%d');
        -> '00'
```

- DATE_SUB(*date*,INTERVAL *expr* *unit*)

  See the description for DATE_ADD().

- DAY(*date*)

DAY() is a synonym for DAYOFMONTH().

- DAYNAME(*date*)

  Returns the name of the weekday for *date*. The language used for the name is controlled by the value of the lc_time_names system variable (Section 10.7, “MySQL Server Locale Support”).

  ```
  mysql> SELECT DAYNAME('2007-02-03');
          -> 'Saturday'
  ```

- DAYOFMONTH(*date*)

  Returns the day of the month for *date*, in the range 1 to 31, or 0 for dates such as '0000-00-00' or '2008-00-00' that have a zero day part.

  ```
  mysql> SELECT DAYOFMONTH('2007-02-03');
          -> 3
  ```

- DAYOFWEEK(*date*)

  Returns the weekday index for *date* (1 = Sunday, 2 = Monday, …, 7 = Saturday). These index values correspond to the ODBC standard.

  ```
  mysql> SELECT DAYOFWEEK('2007-02-03');
          -> 7
  ```

- DAYOFYEAR(*date*)

  Returns the day of the year for *date*, in the range 1 to 366.

  ```
  mysql> SELECT DAYOFYEAR('2007-02-03');
          -> 34
  ```

- EXTRACT(*unit* FROM *date*)

  The EXTRACT() function uses the same kinds of unit specifiers as DATE_ADD() or DATE_SUB(), but extracts parts from the date rather than performing date arithmetic.

  ```
  mysql> SELECT EXTRACT(YEAR FROM '2009-07-02');
         -> 2009
  mysql> SELECT EXTRACT(YEAR_MONTH FROM '2009-07-02 01:02:03');
         -> 200907
  mysql> SELECT EXTRACT(DAY_MINUTE FROM '2009-07-02 01:02:03');
         -> 20102
  mysql> SELECT EXTRACT(MICROSECOND
      ->                FROM '2003-01-02 10:30:00.000123');
          -> 123
  ```

- FROM_DAYS(*N*)

  Given a day number *N*, returns a DATE value.

  ```
  mysql> SELECT FROM_DAYS(730669);
          -> '2007-07-03'
  ```

Use FROM_DAYS() with caution on old dates. It is not intended for use with values that precede the advent of the Gregorian calendar (1582). See Section 12.8, “What Calendar Is Used By MySQL?”.

- FROM_UNIXTIME(*unix_timestamp*), FROM_UNIXTIME(*unix_timestamp*,*format*)

  Returns a representation of the *unix_timestamp* argument as a value in 'YYYY-MM-DD HH:MM:SS' or YYYYMMDDHHMMSS format, depending on whether the function is used in a string or numeric context. The value is expressed in the current time zone. *unix_timestamp* is an internal timestamp value such as is produced by the UNIX_TIMESTAMP() function.

  If *format* is given, the result is formatted according to the *format* string, which is used the same way as listed in the entry for the DATE_FORMAT() function.

```
mysql> SELECT FROM_UNIXTIME(1196440219);
        -> '2007-11-30 10:30:19'
mysql> SELECT FROM_UNIXTIME(1196440219) + 0;
        -> 20071130103019.000000
mysql> SELECT FROM_UNIXTIME(UNIX_TIMESTAMP(),
    ->                      '%Y %D %M %h:%i:%s %x');
        -> '2007 30th November 10:30:59 2007'
```

  Note: If you use UNIX_TIMESTAMP() and FROM_UNIXTIME() to convert between TIMESTAMP values and Unix timestamp values, the conversion is lossy because the mapping is not one-to-one in both directions. For details, see the description of the UNIX_TIMESTAMP() function.

- GET_FORMAT({DATE|TIME|DATETIME}, {'EUR'|'USA'|'JIS'|'ISO'|'INTERNAL'})

  Returns a format string. This function is useful in combination with the DATE_FORMAT() and the STR_TO_DATE() functions.

  The possible values for the first and second arguments result in several possible format strings (for the specifiers used, see the table in the DATE_FORMAT() function description). ISO format refers to ISO 9075, not ISO 8601.

Function Call	Result
GET_FORMAT(DATE,'USA')	'%m.%d.%Y'
GET_FORMAT(DATE,'JIS')	'%Y-%m-%d'
GET_FORMAT(DATE,'ISO')	'%Y-%m-%d'
GET_FORMAT(DATE,'EUR')	'%d.%m.%Y'
GET_FORMAT(DATE,'INTERNAL')	'%Y%m%d'
GET_FORMAT(DATETIME,'USA')	'%Y-%m-%d %H.%i.%s'
GET_FORMAT(DATETIME,'JIS')	'%Y-%m-%d %H:%i:%s'
GET_FORMAT(DATETIME,'ISO')	'%Y-%m-%d %H:%i:%s'
GET_FORMAT(DATETIME,'EUR')	'%Y-%m-%d %H.%i.%s'
GET_FORMAT(DATETIME,'INTERNAL')	'%Y%m%d%H%i%s'
GET_FORMAT(TIME,'USA')	'%h:%i:%s %p'
GET_FORMAT(TIME,'JIS')	'%H:%i:%s'
GET_FORMAT(TIME,'ISO')	'%H:%i:%s'
GET_FORMAT(TIME,'EUR')	'%H.%i.%s'

Function Call	Result
GET_FORMAT(TIME,'INTERNAL')	'%H%i%s'

TIMESTAMP can also be used as the first argument to GET_FORMAT(), in which case the function returns the same values as for DATETIME.

```
mysql> SELECT DATE_FORMAT('2003-10-03',GET_FORMAT(DATE,'EUR'));
        -> '03.10.2003'
mysql> SELECT STR_TO_DATE('10.31.2003',GET_FORMAT(DATE,'USA'));
        -> '2003-10-31'
```

- HOUR(*time*)

  Returns the hour for *time*. The range of the return value is 0 to 23 for time-of-day values. However, the range of TIME values actually is much larger, so HOUR can return values greater than 23.

```
mysql> SELECT HOUR('10:05:03');
        -> 10
mysql> SELECT HOUR('272:59:59');
        -> 272
```

- LAST_DAY(*date*)

  Takes a date or datetime value and returns the corresponding value for the last day of the month. Returns NULL if the argument is invalid.

```
mysql> SELECT LAST_DAY('2003-02-05');
        -> '2003-02-28'
mysql> SELECT LAST_DAY('2004-02-05');
        -> '2004-02-29'
mysql> SELECT LAST_DAY('2004-01-01 01:01:01');
        -> '2004-01-31'
mysql> SELECT LAST_DAY('2003-03-32');
        -> NULL
```

- LOCALTIME, LOCALTIME([*fsp*])

  LOCALTIME and LOCALTIME() are synonyms for NOW().

- LOCALTIMESTAMP, LOCALTIMESTAMP([*fsp*])

  LOCALTIMESTAMP and LOCALTIMESTAMP() are synonyms for NOW().

- MAKEDATE(*year*,*dayofyear*)

  Returns a date, given year and day-of-year values. *dayofyear* must be greater than 0 or the result is NULL.

```
mysql> SELECT MAKEDATE(2011,31), MAKEDATE(2011,32);
        -> '2011-01-31', '2011-02-01'
mysql> SELECT MAKEDATE(2011,365), MAKEDATE(2014,365);
        -> '2011-12-31', '2014-12-31'
mysql> SELECT MAKEDATE(2011,0);
        -> NULL
```

- MAKETIME(*hour*,*minute*,*second*)

  Returns a time value calculated from the *hour*, *minute*, and *second* arguments.

The *second* argument can have a fractional part.

```
mysql> SELECT MAKETIME(12,15,30);
        -> '12:15:30'
```

- `MICROSECOND(expr)`

  Returns the microseconds from the time or datetime expression *expr* as a number in the range from `0` to `999999`.

```
mysql> SELECT MICROSECOND('12:00:00.123456');
        -> 123456
mysql> SELECT MICROSECOND('2009-12-31 23:59:59.000010');
        -> 10
```

- `MINUTE(time)`

  Returns the minute for *time*, in the range `0` to `59`.

```
mysql> SELECT MINUTE('2008-02-03 10:05:03');
        -> 5
```

- `MONTH(date)`

  Returns the month for *date*, in the range `1` to `12` for January to December, or `0` for dates such as `'0000-00-00'` or `'2008-00-00'` that have a zero month part.

```
mysql> SELECT MONTH('2008-02-03');
        -> 2
```

- `MONTHNAME(date)`

  Returns the full name of the month for *date*. The language used for the name is controlled by the value of the `lc_time_names` system variable (Section 10.7, “MySQL Server Locale Support”).

```
mysql> SELECT MONTHNAME('2008-02-03');
        -> 'February'
```

- `NOW([fsp])`

  Returns the current date and time as a value in `'YYYY-MM-DD HH:MM:SS'` or `YYYYMMDDHHMMSS` format, depending on whether the function is used in a string or numeric context. The value is expressed in the current time zone.

  If the *fsp* argument is given to specify a fractional seconds precision from 0 to 6, the return value includes a fractional seconds part of that many digits.

```
mysql> SELECT NOW();
        -> '2007-12-15 23:50:26'
mysql> SELECT NOW() + 0;
        -> 20071215235026.000000
```

  `NOW()` returns a constant time that indicates the time at which the statement began to execute. (Within a stored function or trigger, `NOW()` returns the time at which the function or triggering statement began to execute.) This differs from the behavior for `SYSDATE()`, which returns the exact time at which it executes.

```
mysql> SELECT NOW(), SLEEP(2), NOW();
+---------------------+----------+---------------------+
| NOW()               | SLEEP(2) | NOW()               |
+---------------------+----------+---------------------+
| 2006-04-12 13:47:36 |        0 | 2006-04-12 13:47:36 |
+---------------------+----------+---------------------+

mysql> SELECT SYSDATE(), SLEEP(2), SYSDATE();
+---------------------+----------+---------------------+
| SYSDATE()           | SLEEP(2) | SYSDATE()           |
+---------------------+----------+---------------------+
| 2006-04-12 13:47:44 |        0 | 2006-04-12 13:47:46 |
+---------------------+----------+---------------------+
```

In addition, the `SET TIMESTAMP` statement affects the value returned by `NOW()` but not by `SYSDATE()`. This means that timestamp settings in the binary log have no effect on invocations of `SYSDATE()`. Setting the timestamp to a nonzero value causes each subsequent invocation of `NOW()` to return that value. Setting the timestamp to zero cancels this effect so that `NOW()` once again returns the current date and time.

See the description for `SYSDATE()` for additional information about the differences between the two functions.

- `PERIOD_ADD(`*`P`*`,`*`N`*`)`

  Adds *`N`* months to period *`P`* (in the format `YYMM` or `YYYYMM`). Returns a value in the format `YYYYMM`. Note that the period argument *`P`* is *not* a date value.

  ```
  mysql> SELECT PERIOD_ADD(200801,2);
          -> 200803
  ```

- `PERIOD_DIFF(`*`P1`*`,`*`P2`*`)`

  Returns the number of months between periods *`P1`* and *`P2`*. *`P1`* and *`P2`* should be in the format `YYMM` or `YYYYMM`. Note that the period arguments *`P1`* and *`P2`* are *not* date values.

  ```
  mysql> SELECT PERIOD_DIFF(200802,200703);
          -> 11
  ```

- `QUARTER(`*`date`*`)`

  Returns the quarter of the year for *`date`*, in the range `1` to `4`.

  ```
  mysql> SELECT QUARTER('2008-04-01');
          -> 2
  ```

- `SECOND(`*`time`*`)`

  Returns the second for *`time`*, in the range `0` to `59`.

  ```
  mysql> SELECT SECOND('10:05:03');
          -> 3
  ```

- `SEC_TO_TIME(`*`seconds`*`)`

Returns the *seconds* argument, converted to hours, minutes, and seconds, as a TIME value. The range of the result is constrained to that of the TIME data type. A warning occurs if the argument corresponds to a value outside that range.

```
mysql> SELECT SEC_TO_TIME(2378);
        -> '00:39:38'
mysql> SELECT SEC_TO_TIME(2378) + 0;
        -> 3938
```

- STR_TO_DATE(*str*,*format*)

This is the inverse of the DATE_FORMAT() function. It takes a string *str* and a format string *format*. STR_TO_DATE() returns a DATETIME value if the format string contains both date and time parts, or a DATE or TIME value if the string contains only date or time parts. If the date, time, or datetime value extracted from *str* is illegal, STR_TO_DATE() returns NULL and produces a warning.

The server scans *str* attempting to match *format* to it. The format string can contain literal characters and format specifiers beginning with %. Literal characters in *format* must match literally in *str*. Format specifiers in *format* must match a date or time part in *str*. For the specifiers that can be used in *format*, see the DATE_FORMAT() function description.

```
mysql> SELECT STR_TO_DATE('01,5,2013','%d,%m,%Y');
        -> '2013-05-01'
mysql> SELECT STR_TO_DATE('May 1, 2013','%M %d,%Y');
        -> '2013-05-01'
```

Scanning starts at the beginning of *str* and fails if *format* is found not to match. Extra characters at the end of *str* are ignored.

```
mysql> SELECT STR_TO_DATE('a09:30:17','a%h:%i:%s');
        -> '09:30:17'
mysql> SELECT STR_TO_DATE('a09:30:17','%h:%i:%s');
        -> NULL
mysql> SELECT STR_TO_DATE('09:30:17a','%h:%i:%s');
        -> '09:30:17'
```

Unspecified date or time parts have a value of 0, so incompletely specified values in *str* produce a result with some or all parts set to 0:

```
mysql> SELECT STR_TO_DATE('abc','abc');
        -> '0000-00-00'
mysql> SELECT STR_TO_DATE('9','%m');
        -> '0000-09-00'
mysql> SELECT STR_TO_DATE('9','%s');
        -> '00:00:09'
```

Range checking on the parts of date values is as described in Section 11.3.1, “The DATE, DATETIME, and TIMESTAMP Types”. This means, for example, that “zero” dates or dates with part values of 0 are permitted unless the SQL mode is set to disallow such values.

```
mysql> SELECT STR_TO_DATE('00/00/0000', '%m/%d/%Y');
        -> '0000-00-00'
mysql> SELECT STR_TO_DATE('04/31/2004', '%m/%d/%Y');
        -> '2004-04-31'
```

Note

You cannot use format "%X%V" to convert a year-week string to a date because the combination of a year and week does not uniquely identify a year and month if the week crosses a month boundary. To convert a year-week to a date, you should also specify the weekday:

```
mysql> SELECT STR_TO_DATE('200442 Monday', '%X%V %W');
        -> '2004-10-18'
```

- SUBDATE(*date*,INTERVAL *expr* *unit*), SUBDATE(*expr*,*days*)

  When invoked with the INTERVAL form of the second argument, SUBDATE() is a synonym for DATE_SUB(). For information on the INTERVAL *unit* argument, see the discussion for DATE_ADD().

```
mysql> SELECT DATE_SUB('2008-01-02', INTERVAL 31 DAY);
        -> '2007-12-02'
mysql> SELECT SUBDATE('2008-01-02', INTERVAL 31 DAY);
        -> '2007-12-02'
```

  The second form enables the use of an integer value for *days*. In such cases, it is interpreted as the number of days to be subtracted from the date or datetime expression *expr*.

```
mysql> SELECT SUBDATE('2008-01-02 12:00:00', 31);
        -> '2007-12-02 12:00:00'
```

- SUBTIME(*expr1*,*expr2*)

  SUBTIME() returns *expr1* – *expr2* expressed as a value in the same format as *expr1*. *expr1* is a time or datetime expression, and *expr2* is a time expression.

```
mysql> SELECT SUBTIME('2007-12-31 23:59:59.999999','1 1:1:1.000002');
        -> '2007-12-30 22:58:58.999997'
mysql> SELECT SUBTIME('01:00:00.999999', '02:00:00.999998');
        -> '-00:59:59.999999'
```

- SYSDATE([*fsp*])

  Returns the current date and time as a value in 'YYYY-MM-DD HH:MM:SS' or YYYYMMDDHHMMSS format, depending on whether the function is used in a string or numeric context.

  If the *fsp* argument is given to specify a fractional seconds precision from 0 to 6, the return value includes a fractional seconds part of that many digits. Before 5.6.4, any argument is ignored.

  SYSDATE() returns the time at which it executes. This differs from the behavior for NOW(), which returns a constant time that indicates the time at which the statement began to execute. (Within a stored function or trigger, NOW() returns the time at which the function or triggering statement began to execute.)

```
mysql> SELECT NOW(), SLEEP(2), NOW();
+---------------------+----------+---------------------+
| NOW()               | SLEEP(2) | NOW()               |
+---------------------+----------+---------------------+
| 2006-04-12 13:47:36 |        0 | 2006-04-12 13:47:36 |
+---------------------+----------+---------------------+

mysql> SELECT SYSDATE(), SLEEP(2), SYSDATE();
```

```
+---------------------+----------+---------------------+
| SYSDATE()           | SLEEP(2) | SYSDATE()           |
+---------------------+----------+---------------------+
| 2006-04-12 13:47:44 |        0 | 2006-04-12 13:47:46 |
+---------------------+----------+---------------------+
```

In addition, the `SET TIMESTAMP` statement affects the value returned by `NOW()` but not by `SYSDATE()`. This means that timestamp settings in the binary log have no effect on invocations of `SYSDATE()`.

Because `SYSDATE()` can return different values even within the same statement, and is not affected by `SET TIMESTAMP`, it is nondeterministic and therefore unsafe for replication if statement-based binary logging is used. If that is a problem, you can use row-based logging.

Alternatively, you can use the `--sysdate-is-now` option to cause `SYSDATE()` to be an alias for `NOW()`. This works if the option is used on both the master and the slave.

The nondeterministic nature of `SYSDATE()` also means that indexes cannot be used for evaluating expressions that refer to it.

- `TIME(`*`expr`*`)`

  Extracts the time part of the time or datetime expression *`expr`* and returns it as a string.

  This function is unsafe for statement-based replication. In MySQL 5.7, a warning is logged if you use this function when `binlog_format` is set to `STATEMENT`. (Bug #47995)

```
mysql> SELECT TIME('2003-12-31 01:02:03');
        -> '01:02:03'
mysql> SELECT TIME('2003-12-31 01:02:03.000123');
        -> '01:02:03.000123'
```

- `TIMEDIFF(`*`expr1`*`,`*`expr2`*`)`

  `TIMEDIFF()` returns *`expr1`* – *`expr2`* expressed as a time value. *`expr1`* and *`expr2`* are time or date-and-time expressions, but both must be of the same type.

  The result returned by `TIMEDIFF()` is limited to the range allowed for `TIME` values. Alternatively, you can use either of the functions `TIMESTAMPDIFF()` and `UNIX_TIMESTAMP()`, both of which return integers.

```
mysql> SELECT TIMEDIFF('2000:01:01 00:00:00',
    ->                 '2000:01:01 00:00:00.000001');
        -> '-00:00:00.000001'
mysql> SELECT TIMEDIFF('2008-12-31 23:59:59.000001',
    ->                 '2008-12-30 01:01:01.000002');
        -> '46:58:57.999999'
```

- `TIMESTAMP(`*`expr`*`)`, `TIMESTAMP(`*`expr1`*`,`*`expr2`*`)`

  With a single argument, this function returns the date or datetime expression *`expr`* as a datetime value. With two arguments, it adds the time expression *`expr2`* to the date or datetime expression *`expr1`* and returns the result as a datetime value.

```
mysql> SELECT TIMESTAMP('2003-12-31');
        -> '2003-12-31 00:00:00'
mysql> SELECT TIMESTAMP('2003-12-31 12:00:00','12:00:00');
        -> '2004-01-01 00:00:00'
```

- `TIMESTAMPADD(unit,interval,datetime_expr)`

  Adds the integer expression *`interval`* to the date or datetime expression *`datetime_expr`*. The unit for *`interval`* is given by the *`unit`* argument, which should be one of the following values: `MICROSECOND` (microseconds), `SECOND`, `MINUTE`, `HOUR`, `DAY`, `WEEK`, `MONTH`, `QUARTER`, or `YEAR`.

  The *`unit`* value may be specified using one of keywords as shown, or with a prefix of `SQL_TSI_`. For example, `DAY` and `SQL_TSI_DAY` both are legal.

```
mysql> SELECT TIMESTAMPADD(MINUTE,1,'2003-01-02');
        -> '2003-01-02 00:01:00'
mysql> SELECT TIMESTAMPADD(WEEK,1,'2003-01-02');
        -> '2003-01-09'
```

- `TIMESTAMPDIFF(unit,datetime_expr1,datetime_expr2)`

  Returns *`datetime_expr2`* – *`datetime_expr1`*, where *`datetime_expr1`* and *`datetime_expr2`* are date or datetime expressions. One expression may be a date and the other a datetime; a date value is treated as a datetime having the time part `'00:00:00'` where necessary. The unit for the result (an integer) is given by the *`unit`* argument. The legal values for *`unit`* are the same as those listed in the description of the `TIMESTAMPADD()` function.

```
mysql> SELECT TIMESTAMPDIFF(MONTH,'2003-02-01','2003-05-01');
        -> 3
mysql> SELECT TIMESTAMPDIFF(YEAR,'2002-05-01','2001-01-01');
        -> -1
mysql> SELECT TIMESTAMPDIFF(MINUTE,'2003-02-01','2003-05-01 12:05:55');
        -> 128885
```

  Note

  The order of the date or datetime arguments for this function is the opposite of that used with the `TIMESTAMP()` function when invoked with 2 arguments.

- `TIME_FORMAT(time,format)`

  This is used like the `DATE_FORMAT()` function, but the *`format`* string may contain format specifiers only for hours, minutes, seconds, and microseconds. Other specifiers produce a `NULL` value or `0`.

  If the *`time`* value contains an hour part that is greater than `23`, the `%H` and `%k` hour format specifiers produce a value larger than the usual range of `0..23`. The other hour format specifiers produce the hour value modulo 12.

```
mysql> SELECT TIME_FORMAT('100:00:00', '%H %k %h %I %l');
        -> '100 100 04 04 4'
```

- `TIME_TO_SEC(time)`

  Returns the *`time`* argument, converted to seconds.

```
mysql> SELECT TIME_TO_SEC('22:23:00');
        -> 80580
mysql> SELECT TIME_TO_SEC('00:39:38');
        -> 2378
```

- `TO_DAYS(date)`

Given a date *date*, returns a day number (the number of days since year 0).

```
mysql> SELECT TO_DAYS(950501);
        -> 728779
mysql> SELECT TO_DAYS('2007-10-07');
        -> 733321
```

TO_DAYS() is not intended for use with values that precede the advent of the Gregorian calendar (1582), because it does not take into account the days that were lost when the calendar was changed. For dates before 1582 (and possibly a later year in other locales), results from this function are not reliable. See Section 12.8, “What Calendar Is Used By MySQL?”, for details.

Remember that MySQL converts two-digit year values in dates to four-digit form using the rules in Section 11.3, “Date and Time Types”. For example, '2008-10-07' and '08-10-07' are seen as identical dates:

```
mysql> SELECT TO_DAYS('2008-10-07'), TO_DAYS('08-10-07');
        -> 733687, 733687
```

In MySQL, the zero date is defined as '0000-00-00', even though this date is itself considered invalid. This means that, for '0000-00-00' and '0000-01-01', TO_DAYS() returns the values shown here:

```
mysql> SELECT TO_DAYS('0000-00-00');
+-----------------------+
| to_days('0000-00-00') |
+-----------------------+
|                  NULL |
+-----------------------+
1 row in set, 1 warning (0.00 sec)

mysql> SHOW WARNINGS;
+---------+------+----------------------------------------+
| Level   | Code | Message                                |
+---------+------+----------------------------------------+
| Warning | 1292 | Incorrect datetime value: '0000-00-00' |
+---------+------+----------------------------------------+
1 row in set (0.00 sec)


mysql> SELECT TO_DAYS('0000-01-01');
+-----------------------+
| to_days('0000-01-01') |
+-----------------------+
|                     1 |
+-----------------------+
1 row in set (0.00 sec)
```

This is true whether or not the ALLOW_INVALID_DATES SQL server mode is enabled.

- TO_SECONDS(*expr*)

  Given a date or datetime *expr*, returns a the number of seconds since the year 0. If *expr* is not a valid date or datetime value, returns NULL.

```
mysql> SELECT TO_SECONDS(950501);
        -> 62966505600
mysql> SELECT TO_SECONDS('2009-11-29');
        -> 63426672000
```

```
mysql> SELECT TO_SECONDS('2009-11-29 13:43:32');
        -> 63426721412
mysql> SELECT TO_SECONDS( NOW() );
        -> 63426721458
```

Like TO_DAYS(), TO_SECONDS() is not intended for use with values that precede the advent of the Gregorian calendar (1582), because it does not take into account the days that were lost when the calendar was changed. For dates before 1582 (and possibly a later year in other locales), results from this function are not reliable. See Section 12.8, “What Calendar Is Used By MySQL?”, for details.

Like TO_DAYS(), TO_SECONDS(), converts two-digit year values in dates to four-digit form using the rules in Section 11.3, “Date and Time Types”.

In MySQL, the zero date is defined as '0000-00-00', even though this date is itself considered invalid. This means that, for '0000-00-00' and '0000-01-01', TO_SECONDS() returns the values shown here:

```
mysql> SELECT TO_SECONDS('0000-00-00');
+--------------------------+
| TO_SECONDS('0000-00-00') |
+--------------------------+
|                     NULL |
+--------------------------+
1 row in set, 1 warning (0.00 sec)

mysql> SHOW WARNINGS;
+---------+------+----------------------------------------+
| Level   | Code | Message                                |
+---------+------+----------------------------------------+
| Warning | 1292 | Incorrect datetime value: '0000-00-00' |
+---------+------+----------------------------------------+
1 row in set (0.00 sec)


mysql> SELECT TO_SECONDS('0000-01-01');
+--------------------------+
| TO_SECONDS('0000-01-01') |
+--------------------------+
|                    86400 |
+--------------------------+
1 row in set (0.00 sec)
```

This is true whether or not the ALLOW_INVALID_DATES SQL server mode is enabled.

- UNIX_TIMESTAMP(), UNIX_TIMESTAMP(*date*)

If called with no argument, returns a Unix timestamp (seconds since '1970-01-01 00:00:00' UTC) as an unsigned integer. If UNIX_TIMESTAMP() is called with a *date* argument, it returns the value of the argument as seconds since '1970-01-01 00:00:00' UTC. *date* may be a DATE string, a DATETIME string, a TIMESTAMP, or a number in the format YYMMDD or YYYYMMDD. The server interprets *date* as a value in the current time zone and converts it to an internal value in UTC. Clients can set their time zone as described in Section 10.6, “MySQL Server Time Zone Support”.

```
mysql> SELECT UNIX_TIMESTAMP();
        -> 1196440210
mysql> SELECT UNIX_TIMESTAMP('2007-11-30 10:30:19');
        -> 1196440219
```

When UNIX_TIMESTAMP() is used on a TIMESTAMP column, the function returns the internal timestamp value directly, with no implicit "string-to-Unix-timestamp" conversion. If you pass an out-of-range date to UNIX_TIMESTAMP(), it returns 0.

Note: If you use UNIX_TIMESTAMP() and FROM_UNIXTIME() to convert between TIMESTAMP values and Unix timestamp values, the conversion is lossy because the mapping is not one-to-one in both directions. For example, due to conventions for local time zone changes, it is possible for two UNIX_TIMESTAMP() to map two TIMESTAMP values to the same Unix timestamp value. FROM_UNIXTIME() will map that value back to only one of the original TIMESTAMP values. Here is an example, using TIMESTAMP values in the CET time zone:

```
mysql> SELECT UNIX_TIMESTAMP('2005-03-27 03:00:00');
+---------------------------------------+
| UNIX_TIMESTAMP('2005-03-27 03:00:00') |
+---------------------------------------+
|                            1111885200 |
+---------------------------------------+
mysql> SELECT UNIX_TIMESTAMP('2005-03-27 02:00:00');
+---------------------------------------+
| UNIX_TIMESTAMP('2005-03-27 02:00:00') |
+---------------------------------------+
|                            1111885200 |
+---------------------------------------+
mysql> SELECT FROM_UNIXTIME(1111885200);
+---------------------------+
| FROM_UNIXTIME(1111885200) |
+---------------------------+
| 2005-03-27 03:00:00       |
+---------------------------+
```

If you want to subtract UNIX_TIMESTAMP() columns, you might want to cast the result to signed integers. See Section 12.10, "Cast Functions and Operators".

- UTC_DATE, UTC_DATE()

  Returns the current UTC date as a value in 'YYYY-MM-DD' or YYYYMMDD format, depending on whether the function is used in a string or numeric context.

  ```
  mysql> SELECT UTC_DATE(), UTC_DATE() + 0;
          -> '2003-08-14', 20030814
  ```

- UTC_TIME, UTC_TIME([*fsp*])

  Returns the current UTC time as a value in 'HH:MM:SS' or HHMMSS format, depending on whether the function is used in a string or numeric context.

  If the *fsp* argument is given to specify a fractional seconds precision from 0 to 6, the return value includes a fractional seconds part of that many digits.

  ```
  mysql> SELECT UTC_TIME(), UTC_TIME() + 0;
          -> '18:07:53', 180753.000000
  ```

- UTC_TIMESTAMP, UTC_TIMESTAMP([*fsp*])

  Returns the current UTC date and time as a value in 'YYYY-MM-DD HH:MM:SS' or YYYYMMDDHHMMSS format, depending on whether the function is used in a string or numeric context.

If the *fsp* argument is given to specify a fractional seconds precision from 0 to 6, the return value includes a fractional seconds part of that many digits.

```
mysql> SELECT UTC_TIMESTAMP(), UTC_TIMESTAMP() + 0;
        -> '2003-08-14 18:08:04', 20030814180804.000000
```

- `WEEK(date[,mode])`

  This function returns the week number for *date*. The two-argument form of `WEEK()` enables you to specify whether the week starts on Sunday or Monday and whether the return value should be in the range from `0` to `53` or from `1` to `53`. If the *mode* argument is omitted, the value of the `default_week_format` system variable is used. See Section 5.1.4, “Server System Variables”.

  The following table describes how the *mode* argument works.

Mode	First day of week	Range	Week 1 is the first week …
0	Sunday	0-53	with a Sunday in this year
1	Monday	0-53	with 4 or more days this year
2	Sunday	1-53	with a Sunday in this year
3	Monday	1-53	with 4 or more days this year
4	Sunday	0-53	with 4 or more days this year
5	Monday	0-53	with a Monday in this year
6	Sunday	1-53	with 4 or more days this year
7	Monday	1-53	with a Monday in this year

  For *mode* values with a meaning of “with 4 or more days this year,” weeks are numbered according to ISO 8601:1988:

  - If the week containing January 1 has 4 or more days in the new year, it is week 1.
  - Otherwise, it is the last week of the previous year, and the next week is week 1.

```
mysql> SELECT WEEK('2008-02-20');
        -> 7
mysql> SELECT WEEK('2008-02-20',0);
        -> 7
mysql> SELECT WEEK('2008-02-20',1);
        -> 8
mysql> SELECT WEEK('2008-12-31',1);
        -> 53
```

  Note that if a date falls in the last week of the previous year, MySQL returns `0` if you do not use `2`, `3`, `6`, or `7` as the optional *mode* argument:

```
mysql> SELECT YEAR('2000-01-01'), WEEK('2000-01-01',0);
        -> 2000, 0
```

  One might argue that `WEEK()` should return `52` because the given date actually occurs in the 52nd week of 1999. `WEEK()` returns `0` instead so that the return value is “the week number in the given year.” This makes use of the `WEEK()` function reliable when combined with other functions that extract a date part from a date.

If you prefer a result evaluated with respect to the year that contains the first day of the week for the given date, use `0`, `2`, `5`, or `7` as the optional *`mode`* argument.

```
mysql> SELECT WEEK('2000-01-01',2);
        -> 52
```

Alternatively, use the `YEARWEEK()` function:

```
mysql> SELECT YEARWEEK('2000-01-01');
        -> 199952
mysql> SELECT MID(YEARWEEK('2000-01-01'),5,2);
        -> '52'
```

- `WEEKDAY(date)`

  Returns the weekday index for *`date`* (`0` = Monday, `1` = Tuesday, … `6` = Sunday).

  ```
  mysql> SELECT WEEKDAY('2008-02-03 22:23:00');
          -> 6
  mysql> SELECT WEEKDAY('2007-11-06');
          -> 1
  ```

- `WEEKOFYEAR(date)`

  Returns the calendar week of the date as a number in the range from `1` to `53`. `WEEKOFYEAR()` is a compatibility function that is equivalent to `WEEK(date,3)`.

  ```
  mysql> SELECT WEEKOFYEAR('2008-02-20');
          -> 8
  ```

- `YEAR(date)`

  Returns the year for *`date`*, in the range `1000` to `9999`, or `0` for the “zero” date.

  ```
  mysql> SELECT YEAR('1987-01-01');
          -> 1987
  ```

- `YEARWEEK(date)`, `YEARWEEK(date,mode)`

  Returns year and week for a date. The *`mode`* argument works exactly like the *`mode`* argument to `WEEK()`. The year in the result may be different from the year in the date argument for the first and the last week of the year.

  ```
  mysql> SELECT YEARWEEK('1987-01-01');
          -> 198653
  ```

  Note that the week number is different from what the `WEEK()` function would return (`0`) for optional arguments `0` or `1`, as `WEEK()` then returns the week in the context of the given year.

12.8 What Calendar Is Used By MySQL?

MySQL uses what is known as a *proleptic Gregorian calendar*.

Every country that has switched from the Julian to the Gregorian calendar has had to discard at least ten days during the switch. To see how this works, consider the month of October 1582, when the first Julian-to-Gregorian switch occurred.

Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday
1	2	3	4	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

There are no dates between October 4 and October 15. This discontinuity is called the *cutover*. Any dates before the cutover are Julian, and any dates following the cutover are Gregorian. Dates during a cutover are nonexistent.

A calendar applied to dates when it was not actually in use is called *proleptic*. Thus, if we assume there was never a cutover and Gregorian rules always rule, we have a proleptic Gregorian calendar. This is what is used by MySQL, as is required by standard SQL. For this reason, dates prior to the cutover stored as MySQL `DATE` or `DATETIME` values must be adjusted to compensate for the difference. It is important to realize that the cutover did not occur at the same time in all countries, and that the later it happened, the more days were lost. For example, in Great Britain, it took place in 1752, when Wednesday September 2 was followed by Thursday September 14. Russia remained on the Julian calendar until 1918, losing 13 days in the process, and what is popularly referred to as its “October Revolution” occurred in November according to the Gregorian calendar.

12.9 Full-Text Search Functions

`MATCH (col1,col2,...) AGAINST (expr [search_modifier])` [1271]

```
search_modifier:
  {
       IN NATURAL LANGUAGE MODE
     | IN NATURAL LANGUAGE MODE WITH QUERY EXPANSION
     | IN BOOLEAN MODE
     | WITH QUERY EXPANSION
  }
```

MySQL has support for full-text indexing and searching:

- A full-text index in MySQL is an index of type `FULLTEXT`.
- Full-text indexes can be used only with `InnoDB` or `MyISAM` tables, and can be created only for `CHAR`, `VARCHAR`, or `TEXT` columns.
- A `FULLTEXT` index definition can be given in the `CREATE TABLE` statement when a table is created, or added later using `ALTER TABLE` or `CREATE INDEX`.
- For large data sets, it is much faster to load your data into a table that has no `FULLTEXT` index and then create the index after that, than to load data into a table that has an existing `FULLTEXT` index.

Full-text searching is performed using `MATCH() ... AGAINST` [1271] syntax. `MATCH()` [1271] takes a comma-separated list that names the columns to be searched. `AGAINST` takes a string to search for, and an optional modifier that indicates what type of search to perform. The search string must be a string value that is constant during query evaluation. This rules out, for example, a table column because that can differ for each row.

There are three types of full-text searches:

- A natural language search interprets the search string as a phrase in natural human language (a phrase in free text). There are no special operators. The stopword list applies, controlled by innodb_ft_enable_stopword, innodb_ft_server_stopword_table, and innodb_ft_user_stopword_table for InnoDB search indexes, and ft_stopword_file for MyISAM ones.

- Full-text searches are natural language searches if the IN NATURAL LANGUAGE MODE modifier is given or if no modifier is given. For more information, see Section 12.9.1, “Natural Language Full-Text Searches”.

- A boolean search interprets the search string using the rules of a special query language. The string contains the words to search for. It can also contain operators that specify requirements such that a word must be present or absent in matching rows, or that it should be weighted higher or lower than usual. Certain common words (stopwords) are omitted from the search index and do not match if present in the search string. The IN BOOLEAN MODE modifier specifies a boolean search. For more information, see Section 12.9.2, “Boolean Full-Text Searches”.

- A query expansion search is a modification of a natural language search. The search string is used to perform a natural language search. Then words from the most relevant rows returned by the search are added to the search string and the search is done again. The query returns the rows from the second search. The IN NATURAL LANGUAGE MODE WITH QUERY EXPANSION or WITH QUERY EXPANSION modifier specifies a query expansion search. For more information, see Section 12.9.3, “Full-Text Searches with Query Expansion”.

For information about FULLTEXT query performance, see Section 8.3.4, “Column Indexes”.

For more technical details about processing for InnoDB FULLTEXT indexes, see FULLTEXT Indexes.

Constraints on full-text searching are listed in Section 12.9.5, “Full-Text Restrictions”.

The myisam_ftdump utility dumps the contents of a MyISAM full-text index. This may be helpful for debugging full-text queries. See Section 4.6.2, “myisam_ftdump — Display Full-Text Index information”.

12.9.1 Natural Language Full-Text Searches

By default or with the IN NATURAL LANGUAGE MODE modifier, the MATCH() [1271] function performs a natural language search for a string against a *text collection*. A collection is a set of one or more columns included in a FULLTEXT index. The search string is given as the argument to AGAINST(). For each row in the table, MATCH() [1271] returns a relevance value; that is, a similarity measure between the search string and the text in that row in the columns named in the MATCH() [1271] list.

```
mysql> CREATE TABLE articles (
      id INT UNSIGNED AUTO_INCREMENT NOT NULL PRIMARY KEY,
      title VARCHAR(200),
      body TEXT,
      FULLTEXT (title,body)
    ) ENGINE=InnoDB;
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO articles (title,body) VALUES
    ('MySQL Tutorial','DBMS stands for DataBase ...'),
    ('How To Use MySQL Well','After you went through a ...'),
    ('Optimizing MySQL','In this tutorial we will show ...'),
    ('1001 MySQL Tricks','1. Never run mysqld as root. 2. ...'),
    ('MySQL vs. YourSQL','In the following database comparison ...'),
    ('MySQL Security','When configured properly, MySQL ...');
Query OK, 6 rows affected (0.00 sec)
Records: 6  Duplicates: 0  Warnings: 0
```

```
mysql> SELECT * FROM articles
    WHERE MATCH (title,body)
    AGAINST ('database' IN NATURAL LANGUAGE MODE);
+----+-------------------+------------------------------------------+
| id | title             | body                                     |
+----+-------------------+------------------------------------------+
|  1 | MySQL Tutorial    | DBMS stands for DataBase ...             |
|  5 | MySQL vs. YourSQL | In the following database comparison ... |
+----+-------------------+------------------------------------------+
2 rows in set (0.00 sec)
```

By default, the search is performed in case-insensitive fashion. To perform a case-sensitive full-text search, use a binary collation for the indexed columns. For example, a column that uses the latin1 character set of can be assigned a collation of latin1_bin to make it case sensitive for full-text searches.

When MATCH() [1271] is used in a WHERE clause, as in the example shown earlier, the rows returned are automatically sorted with the highest relevance first. Relevance values are nonnegative floating-point numbers. Zero relevance means no similarity. Relevance is computed based on the number of words in the row, the number of unique words in that row, the total number of words in the collection, and the number of documents (rows) that contain a particular word.

To simply count matches, you could use a query like this:

```
mysql> SELECT COUNT(*) FROM articles
    WHERE MATCH (title,body)
    AGAINST ('database' IN NATURAL LANGUAGE MODE);
+----------+
| COUNT(*) |
+----------+
|        2 |
+----------+
1 row in set (0.00 sec)
```

You might find it quicker to rewrite the query as follows:

```
mysql> SELECT
    COUNT(IF(MATCH (title,body) AGAINST ('database' IN NATURAL LANGUAGE MODE), 1, NULL))
    AS count
    FROM articles;
+-------+
| count |
+-------+
|     2 |
+-------+
1 row in set (0.03 sec)
```

The first query does some extra work (sorting the results by relevance) but also can use an index lookup based on the WHERE clause. The index lookup might make the first query faster if the search matches few rows. The second query performs a full table scan, which might be faster than the index lookup if the search term was present in most rows.

For natural-language full-text searches, the columns named in the MATCH() [1271] function must be the same columns included in some FULLTEXT index in your table. For the preceding query, note that the columns named in the MATCH() [1271] function (title and body) are the same as those named in the definition of the article table's FULLTEXT index. To search the title or body separately, you would create separate FULLTEXT indexes for each column.

You can also perform a boolean search or a search with query expansion. These search types are described in Section 12.9.2, “Boolean Full-Text Searches”, and Section 12.9.3, “Full-Text Searches with Query Expansion”.

A full-text search that uses an index can name columns only from a single table in the MATCH() [1271] clause because an index cannot span multiple tables. For MyISAM tables, a boolean search can be done in the absence of an index (albeit more slowly), in which case it is possible to name columns from multiple tables.

The preceding example is a basic illustration that shows how to use the MATCH() [1271] function where rows are returned in order of decreasing relevance. The next example shows how to retrieve the relevance values explicitly. Returned rows are not ordered because the SELECT statement includes neither WHERE nor ORDER BY clauses:

```
mysql> SELECT id, MATCH (title,body)
    AGAINST ('Tutorial' IN NATURAL LANGUAGE MODE) AS score
    FROM articles;
+----+---------------------+
| id | score               |
+----+---------------------+
|  1 | 0.22764469683170319 |
|  2 |                   0 |
|  3 | 0.22764469683170319 |
|  4 |                   0 |
|  5 |                   0 |
|  6 |                   0 |
+----+---------------------+
6 rows in set (0.00 sec)
```

The following example is more complex. The query returns the relevance values and it also sorts the rows in order of decreasing relevance. To achieve this result, specify MATCH() [1271] twice: once in the SELECT list and once in the WHERE clause. This causes no additional overhead, because the MySQL optimizer notices that the two MATCH() [1271] calls are identical and invokes the full-text search code only once.

```
mysql> SELECT id, body, MATCH (title,body) AGAINST
    ('Security implications of running MySQL as root'
    IN NATURAL LANGUAGE MODE) AS score
    FROM articles WHERE MATCH (title,body) AGAINST
    ('Security implications of running MySQL as root'
    IN NATURAL LANGUAGE MODE);
+----+-------------------------------------+-----------------+
| id | body                                | score           |
+----+-------------------------------------+-----------------+
|  4 | 1. Never run mysqld as root. 2. ... | 1.5219271183014 |
|  6 | When configured properly, MySQL ... | 1.3114095926285 |
+----+-------------------------------------+-----------------+
2 rows in set (0.00 sec)
```

The MySQL FULLTEXT implementation regards any sequence of true word characters (letters, digits, and underscores) as a word. That sequence may also contain apostrophes (“'”), but not more than one in a row. This means that aaa'bbb is regarded as one word, but aaa''bbb is regarded as two words. Apostrophes at the beginning or the end of a word are stripped by the FULLTEXT parser; 'aaa'bbb' would be parsed as aaa'bbb.

The FULLTEXT parser determines where words start and end by looking for certain delimiter characters; for example, “ ” (space), “,” (comma), and “.” (period). If words are not separated by delimiters (as in, for example, Chinese), the FULLTEXT parser cannot determine where a word begins or ends. To be able to add words or other indexed terms in such languages to a FULLTEXT index, you must preprocess them so that they are separated by some arbitrary delimiter such as “"”.

In MySQL 5.7, it is possible to write a plugin that replaces the built-in full-text parser. For details, see Section 22.2, “The MySQL Plugin API”. For example parser plugin source code, see the plugin/fulltext directory of a MySQL source distribution.

Some words are ignored in full-text searches:

- Any word that is too short is ignored. The default minimum length of words that are found by full-text searches is three characters for InnoDB search indexes, or four characters for MyISAM. You can control the cutoff by setting a configuration option before creating the index: innodb_ft_min_token_size configuration option for InnoDB search indexes, or ft_min_word_len for MyISAM.
- Words in the stopword list are ignored. A stopword is a word such as “the” or “some” that is so common that it is considered to have zero semantic value. There is a built-in stopword list, but it can be overridden by a user-defined list. The stopword lists and related configuration options are different for InnoDB search indexes and MyISAM ones. Stopword processing is controlled by the configuration options innodb_ft_enable_stopword, innodb_ft_server_stopword_table, and innodb_ft_user_stopword_table for InnoDB search indexes, and ft_stopword_file for MyISAM ones.

The default stopword lists are shown in Section 12.9.4, “Full-Text Stopwords”. The default minimum word length and stopword list can be changed as described in Section 12.9.6, “Fine-Tuning MySQL Full-Text Search”.

Every correct word in the collection and in the query is weighted according to its significance in the collection or query. Thus, a word that is present in many documents has a lower weight, because it has lower semantic value in this particular collection. Conversely, if the word is rare, it receives a higher weight. The weights of the words are combined to compute the relevance of the row. This technique works best with large collections.

MyISAM Limitation

For very small tables, word distribution does not adequately reflect their semantic value, and this model may sometimes produce bizarre results for search indexes on MyISAM tables. For example, although the word “MySQL” is present in every row of the articles table shown earlier, a search for the word in a MyISAM search index produces no results:

```
mysql> SELECT * FROM articles
    WHERE MATCH (title,body)
    AGAINST ('MySQL' IN NATURAL LANGUAGE MODE);
Empty set (0.00 sec)
```

The search result is empty because the word “MySQL” is present in at least 50% of the rows, and so is effectively treated as a stopword. This filtering technique is more suitable for large data sets, where you might not want the result set to return every second row from a 1GB table, than for small data sets where it might cause poor results for popular terms.

The 50% threshold can surprise you when you first try full-text searching to see how it works, and makes InnoDB tables more suited to experimentation with full-text searches. If you create a MyISAM table and insert only one or two rows of text into it, every word in the text occurs in at least 50% of the rows. As a result, no search returns any results until the table contains more rows. Users who need to bypass the 50% limitation can build search indexes on InnoDB tables, or the boolean search mode explained in Section 12.9.2, “Boolean Full-Text Searches”.

12.9.2 Boolean Full-Text Searches

MySQL can perform boolean full-text searches using the IN BOOLEAN MODE modifier. With this modifier, certain characters have special meaning at the beginning or end of words in the search string. In the following query, the + and - operators indicate that a word must be present or absent, respectively, for a match to occur. Thus, the query retrieves all the rows that contain the word “MySQL” but that do *not* contain the word “YourSQL”:

```
mysql> SELECT * FROM articles WHERE MATCH (title,body)
    AGAINST ('+MySQL -YourSQL' IN BOOLEAN MODE);
+----+-----------------------+-------------------------------------+
| id | title                 | body                                |
+----+-----------------------+-------------------------------------+
|  1 | MySQL Tutorial        | DBMS stands for DataBase ...        |
|  2 | How To Use MySQL Well | After you went through a ...        |
|  3 | Optimizing MySQL      | In this tutorial we will show ...   |
|  4 | 1001 MySQL Tricks     | 1. Never run mysqld as root. 2. ... |
|  6 | MySQL Security        | When configured properly, MySQL ... |
+----+-----------------------+-------------------------------------+
```

Note

In implementing this feature, MySQL uses what is sometimes referred to as *implied Boolean logic*, in which

- + stands for AND
- - stands for NOT
- [*no operator*] implies OR

Boolean full-text searches have these characteristics:

- They do not use the 50% threshold that applies to MyISAM search indexes.
- They do not automatically sort rows in order of decreasing relevance.
- Boolean queries against a MyISAM search index can work even without a FULLTEXT index, although a search executed in this fashion would be quite slow. InnoDB tables require a FULLTEXT index on all columns of the MATCH() [1271] expression to perform boolean queries.
- The minimum and maximum word length full-text parameters apply: innodb_ft_min_token_size and innodb_ft_max_token_size for InnoDB search indexes, and ft_min_word_len and ft_max_word_len for MyISAM ones.
- The stopword list applies, controlled by innodb_ft_enable_stopword, innodb_ft_server_stopword_table, and innodb_ft_user_stopword_table for InnoDB search indexes, and ft_stopword_file for MyISAM ones.
- InnoDB full-text search does not support the use of multiple operators on a single search word, as in this example: '++apple'. Use of multiple operators on a single search word returns a syntax error to standard out. MyISAM full-text search will successfully process the same search ignoring all operators except for the operator immediately adjacent to the search word.

The boolean full-text search capability supports the following operators:

- +

  A leading plus sign indicates that this word *must* be present in each row that is returned.

- -

  A leading minus sign indicates that this word must *not* be present in any of the rows that are returned.

Note: The - operator acts only to exclude rows that are otherwise matched by other search terms. Thus, a boolean-mode search that contains only terms preceded by - returns an empty result. It does not return “all rows except those containing any of the excluded terms.”

- (no operator)

  By default (when neither + nor - is specified), the word is optional, but the rows that contain it are rated higher. This mimics the behavior of `MATCH() ... AGAINST()` [1271] without the `IN BOOLEAN MODE` modifier.

- *`@distance`*

  This operator works on `InnoDB` tables only. It tests whether two or more words all start within a specified distance from each other, measured in words. Specify the search words within a double-quoted string immediately before the *`@distance`* operator, for example, `MATCH(col1) AGAINST('"word1 word2 word3" @8' IN BOOLEAN MODE)`

- `> <`

  These two operators are used to change a word's contribution to the relevance value that is assigned to a row. The > operator increases the contribution and the < operator decreases it. See the example following this list.

- `( )`

  Parentheses group words into subexpressions. Parenthesized groups can be nested.

- `~`

  A leading tilde acts as a negation operator, causing the word's contribution to the row's relevance to be negative. This is useful for marking “noise” words. A row containing such a word is rated lower than others, but is not excluded altogether, as it would be with the - operator.

- `*`

  The asterisk serves as the truncation (or wildcard) operator. Unlike the other operators, it is *appended* to the word to be affected. Words match if they begin with the word preceding the * operator.

  If a word is specified with the truncation operator, it is not stripped from a boolean query, even if it is too short or a stopword. Whether a word is too short is determined from the `innodb_ft_min_token_size` setting for `InnoDB` tables, or `ft_min_word_len` for `MyISAM` tables. The wildcarded word is considered as a prefix that must be present at the start of one or more words. If the minimum word length is 4, a search for `'+`*`word`*` +the*'` could return fewer rows than a search for `'+`*`word`*` +the'`, because the second query ignores the too-short search term `the`.

- `"`

  A phrase that is enclosed within double quote (“`"`”) characters matches only rows that contain the phrase *literally, as it was typed*. The full-text engine splits the phrase into words and performs a search in the `FULLTEXT` index for the words. Nonword characters need not be matched exactly: Phrase searching requires only that matches contain exactly the same words as the phrase and in the same order. For example, `"test phrase"` matches `"test, phrase"`.

  If the phrase contains no words that are in the index, the result is empty. The words might not be in the index because of a combination of factors: if they do not exist in the text, are stopwords, or are shorter than the minimum length of indexed words.

The following examples demonstrate some search strings that use boolean full-text operators:

- `'apple banana'`

  Find rows that contain at least one of the two words.

- `'+apple +juice'`

  Find rows that contain both words.

- `'+apple macintosh'`

  Find rows that contain the word “apple”, but rank rows higher if they also contain “macintosh”.

- `'+apple -macintosh'`

  Find rows that contain the word “apple” but not “macintosh”.

- `'+apple ~macintosh'`

  Find rows that contain the word “apple”, but if the row also contains the word “macintosh”, rate it lower than if row does not. This is “softer” than a search for `'+apple -macintosh'`, for which the presence of “macintosh” causes the row not to be returned at all.

- `'+apple +(>turnover <strudel)'`

  Find rows that contain the words “apple” and “turnover”, or “apple” and “strudel” (in any order), but rank “apple turnover” higher than “apple strudel”.

- `'apple*'`

  Find rows that contain words such as “apple”, “apples”, “applesauce”, or “applet”.

- `'"some words"'`

  Find rows that contain the exact phrase “some words” (for example, rows that contain “some words of wisdom” but not “some noise words”). Note that the “`"`” characters that enclose the phrase are operator characters that delimit the phrase. They are not the quotation marks that enclose the search string itself.

12.9.3 Full-Text Searches with Query Expansion

Full-text search supports query expansion (and in particular, its variant “blind query expansion”). This is generally useful when a search phrase is too short, which often means that the user is relying on implied knowledge that the full-text search engine lacks. For example, a user searching for “database” may really mean that “MySQL”, “Oracle”, “DB2”, and “RDBMS” all are phrases that should match “databases” and should be returned, too. This is implied knowledge.

Blind query expansion (also known as automatic relevance feedback) is enabled by adding `WITH QUERY EXPANSION` or `IN NATURAL LANGUAGE MODE WITH QUERY EXPANSION` following the search phrase. It works by performing the search twice, where the search phrase for the second search is the original search phrase concatenated with the few most highly relevant documents from the first search. Thus, if one of these documents contains the word “databases” and the word “MySQL”, the second search finds the documents that contain the word “MySQL” even if they do not contain the word “database”. The following example shows this difference:

```
mysql> SELECT * FROM articles
    WHERE MATCH (title,body)
    AGAINST ('database' IN NATURAL LANGUAGE MODE);
+----+-------------------+------------------------------------------+
| id | title             | body                                     |
+----+-------------------+------------------------------------------+
|  1 | MySQL Tutorial    | DBMS stands for DataBase ...             |
|  5 | MySQL vs. YourSQL | In the following database comparison ... |
+----+-------------------+------------------------------------------+
2 rows in set (0.00 sec)

mysql> SELECT * FROM articles
    WHERE MATCH (title,body)
    AGAINST ('database' WITH QUERY EXPANSION);
+----+-----------------------+------------------------------------------+
| id | title                 | body                                     |
+----+-----------------------+------------------------------------------+
|  5 | MySQL vs. YourSQL     | In the following database comparison ... |
|  1 | MySQL Tutorial        | DBMS stands for DataBase ...             |
|  3 | Optimizing MySQL      | In this tutorial we will show ...        |
|  6 | MySQL Security        | When configured properly, MySQL ...      |
|  2 | How To Use MySQL Well | After you went through a ...             |
|  4 | 1001 MySQL Tricks     | 1. Never run mysqld as root. 2. ...      |
+----+-----------------------+------------------------------------------+
6 rows in set (0.00 sec)
```

Another example could be searching for books by Georges Simenon about Maigret, when a user is not sure how to spell “Maigret”. A search for “Megre and the reluctant witnesses” finds only “Maigret and the Reluctant Witnesses” without query expansion. A search with query expansion finds all books with the word “Maigret” on the second pass.

Note

Because blind query expansion tends to increase noise significantly by returning nonrelevant documents, use it only when a search phrase is short.

12.9.4 Full-Text Stopwords

The stopword list is loaded and searched for full-text queries using the server character set and collation (the values of the `character_set_server` and `collation_server` system variables). False hits or misses might occur for stopword lookups if the stopword file or columns used for full-text indexing or searches have a character set or collation different from `character_set_server` or `collation_server`.

Case sensitivity of stopword lookups depends on the server collation. For example, lookups are case insensitive if the collation is `latin1_swedish_ci`, whereas lookups are case sensitive if the collation is `latin1_general_cs` or `latin1_bin`.

Stopwords for `InnoDB` Search Indexes

`InnoDB` has a relatively short list of default stopwords, because documents from technical, literary, and so on sources often use short words as keywords or in significant phrases. For example, you might search for “to be or not to be” and expect to get a sensible result, rather than having all those words ignored.

To see the list, query the table `information_schema.innodb_ft_default_stopword`. To define your own stopword list used for all `InnoDB` tables, define a table with the same structure as `innodb_ft_default_stopword`, fill it with the desired stopwords, and set the value of the `innodb_ft_server_stopword_table` option to a value of the form *`db_name/table_name`* before creating the search index. To create special stopword lists on a table-by-table basis, define other tables to hold these lists and specify the appropriate one in the `innodb_ft_user_stopword_table` option before creating the search index.

Stopwords for `MyISAM` Search Indexes

In MySQL 5.7, the stopword file is loaded and searched using `latin1` if `character_set_server` is `ucs2`, `utf16`, `utf16le`, or `utf32`.

The following table shows the default list of stopwords for `MyISAM` search indexes. In a MySQL source distribution, you can find this list in the `storage/myisam/ft_static.c` file.

a's	able	about	above	according
accordingly	across	actually	after	afterwards
again	against	ain't	all	allow
allows	almost	alone	along	already
also	although	always	am	among
amongst	an	and	another	any
anybody	anyhow	anyone	anything	anyway
anyways	anywhere	apart	appear	appreciate
appropriate	are	aren't	around	as
aside	ask	asking	associated	at
available	away	awfully	be	became
because	become	becomes	becoming	been
before	beforehand	behind	being	believe
below	beside	besides	best	better
between	beyond	both	brief	but
by	c'mon	c's	came	can
can't	cannot	cant	cause	causes
certain	certainly	changes	clearly	co
com	come	comes	concerning	consequently
consider	considering	contain	containing	contains
corresponding	could	couldn't	course	currently
definitely	described	despite	did	didn't
different	do	does	doesn't	doing
don't	done	down	downwards	during
each	edu	eg	eight	either
else	elsewhere	enough	entirely	especially
et	etc	even	ever	every
everybody	everyone	everything	everywhere	ex
exactly	example	except	far	few
fifth	first	five	followed	following
follows	for	former	formerly	forth
four	from	further	furthermore	get
gets	getting	given	gives	go
goes	going	gone	got	gotten

greetings	had	hadn't	happens	hardly
has	hasn't	have	haven't	having
he	he's	hello	help	hence
her	here	here's	hereafter	hereby
herein	hereupon	hers	herself	hi
him	himself	his	hither	hopefully
how	howbeit	however	i'd	i'll
i'm	i've	ie	if	ignored
immediate	in	inasmuch	inc	indeed
indicate	indicated	indicates	inner	insofar
instead	into	inward	is	isn't
it	it'd	it'll	it's	its
itself	just	keep	keeps	kept
know	known	knows	last	lately
later	latter	latterly	least	less
lest	let	let's	like	liked
likely	little	look	looking	looks
ltd	mainly	many	may	maybe
me	mean	meanwhile	merely	might
more	moreover	most	mostly	much
must	my	myself	name	namely
nd	near	nearly	necessary	need
needs	neither	never	nevertheless	new
next	nine	no	nobody	non
none	noone	nor	normally	not
nothing	novel	now	nowhere	obviously
of	off	often	oh	ok
okay	old	on	once	one
ones	only	onto	or	other
others	otherwise	ought	our	ours
ourselves	out	outside	over	overall
own	particular	particularly	per	perhaps
placed	please	plus	possible	presumably
probably	provides	que	quite	qv
rather	rd	re	really	reasonably
regarding	regardless	regards	relatively	respectively
right	said	same	saw	say
saying	says	second	secondly	see
seeing	seem	seemed	seeming	seems

seen	self	selves	sensible	sent
serious	seriously	seven	several	shall
she	should	shouldn't	since	six
so	some	somebody	somehow	someone
something	sometime	sometimes	somewhat	somewhere
soon	sorry	specified	specify	specifying
still	sub	such	sup	sure
t's	take	taken	tell	tends
th	than	thank	thanks	thanx
that	that's	thats	the	their
theirs	them	themselves	then	thence
there	there's	thereafter	thereby	therefore
therein	theres	thereupon	these	they
they'd	they'll	they're	they've	think
third	this	thorough	thoroughly	those
though	three	through	throughout	thru
thus	to	together	too	took
toward	towards	tried	tries	truly
try	trying	twice	two	un
under	unfortunately	unless	unlikely	until
unto	up	upon	us	use
used	useful	uses	using	usually
value	various	very	via	viz
vs	want	wants	was	wasn't
way	we	we'd	we'll	we're
we've	welcome	well	went	were
weren't	what	what's	whatever	when
whence	whenever	where	where's	whereafter
whereas	whereby	wherein	whereupon	wherever
whether	which	while	whither	who
who's	whoever	whole	whom	whose
why	will	willing	wish	with
within	without	won't	wonder	would
wouldn't	yes	yet	you	you'd
you'll	you're	you've	your	yours
yourself	yourselves	zero		

12.9.5 Full-Text Restrictions

- Full-text searches are supported for `InnoDB` and `MyISAM` tables only.

- Full-text searches are not supported for partitioned tables. See Section 17.6, “Restrictions and Limitations on Partitioning”.
- Full-text searches can be used with most multi-byte character sets. The exception is that for Unicode, the utf8 character set can be used, but not the ucs2 character set. Although FULLTEXT indexes on ucs2 columns cannot be used, you can perform IN BOOLEAN MODE searches on a ucs2 column that has no such index.

  The remarks for utf8 also apply to utf8mb4, and the remarks for ucs2 also apply to utf16, utf16le, and utf32.
- Ideographic languages such as Chinese and Japanese do not have word delimiters. Therefore, the FULLTEXT parser *cannot determine where words begin and end in these and other such languages*. The implications of this and some workarounds for the problem are described in Section 12.9, “Full-Text Search Functions”.
- Although the use of multiple character sets within a single table is supported, all columns in a FULLTEXT index must use the same character set and collation.
- The MATCH() [1271] column list must match exactly the column list in some FULLTEXT index definition for the table, unless this MATCH() [1271] is IN BOOLEAN MODE on a MyISAM table. For MyISAM tables, boolean-mode searches can be done on nonindexed columns, although they are likely to be slow.
- The argument to AGAINST() must be a string value that is constant during query evaluation. This rules out, for example, a table column because that can differ for each row.
- Index hints are more limited for FULLTEXT searches than for non-FULLTEXT searches. See Section 13.2.9.3, “Index Hint Syntax”.

12.9.6 Fine-Tuning MySQL Full-Text Search

MySQL's full-text search capability has few user-tunable parameters. You can exert more control over full-text searching behavior if you have a MySQL source distribution because some changes require source code modifications. See Section 2.8, “Installing MySQL from Source”.

Note that full-text search is carefully tuned for effectiveness. Modifying the default behavior in most cases can actually decrease effectiveness. *Do not alter the MySQL sources unless you know what you are doing*.

Most full-text variables described in this section must be set at server startup time. A server restart is required to change them; they cannot be modified while the server is running.

Some variable changes require that you rebuild the FULLTEXT indexes in your tables. Instructions for doing so are given later in this section.

- The minimum and maximum lengths of words to be indexed are defined by the innodb_ft_min_token_size and innodb_ft_max_token_size for InnoDB search indexes, and ft_min_word_len and ft_max_word_len for MyISAM ones. After changing any of these options, rebuild your FULLTEXT indexes for the change to take effect. For example, to make two-character words searchable, you could put the following lines in an option file:

  ```
  [mysqld]
  innodb_ft_min_token_size=2
  ft_min_word_len=2
  ```

  Then restart the server and rebuild your FULLTEXT indexes. For MyISAM tables, note particularly the remarks regarding myisamchk in the instructions following this list.

- To override the default stopword list, set the ft_stopword_file system variable. (See Section 5.1.4, “Server System Variables”.) The variable value should be the path name of the file containing the stopword list, or the empty string to disable stopword filtering. The server looks for the file in the data directory unless an absolute path name is given to specify a different directory. After changing the value of this variable or the contents of the stopword file, restart the server and rebuild your FULLTEXT indexes.

  The stopword list is free-form, separating stopwords with any nonalphanumeric character such as newline, space, or comma. Exceptions are the underscore character (“_”) and a single apostrophe (“'”) which are treated as part of a word. The character set of the stopword list is the server's default character set; see Section 10.1.3.1, “Server Character Set and Collation”.

- The 50% threshold for natural language searches is determined by the particular weighting scheme chosen. To disable it, look for the following line in storage/myisam/ftdefs.h:

  ```
  #define GWS_IN_USE GWS_PROB
  ```

  Change that line to this:

  ```
  #define GWS_IN_USE GWS_FREQ
  ```

  Then recompile MySQL. There is no need to rebuild the indexes in this case.

  Note

  By making this change, you *severely* decrease MySQL's ability to provide adequate relevance values for the MATCH() [1271] function. If you really need to search for such common words, it would be better to search using IN BOOLEAN MODE instead, which does not observe the 50% threshold.

- To change the operators used for boolean full-text searches on MyISAM tables, set the ft_boolean_syntax system variable. (InnoDB does not have an equivalent setting.) This variable can be changed while the server is running, but you must have the SUPER privilege to do so. No rebuilding of indexes is necessary in this case. See Section 5.1.4, “Server System Variables”, which describes the rules governing how to set this variable.

- You can change the set of characters that are considered word characters in several ways, as described in the following list. After making the modification, rebuild the indexes for each table that contains any FULLTEXT indexes. Suppose that you want to treat the hyphen character ('-') as a word character. Use one of these methods:

  - Modify the MySQL source: In storage/myisam/ftdefs.h, see the true_word_char() and misc_word_char() macros. Add '-' to one of those macros and recompile MySQL.

  - Modify a character set file: This requires no recompilation. The true_word_char() macro uses a “character type” table to distinguish letters and numbers from other characters. . You can edit the contents of the <ctype><map> array in one of the character set XML files to specify that '-' is a “letter.” Then use the given character set for your FULLTEXT indexes. For information about the <ctype><map> array format, see Section 10.3.1, “Character Definition Arrays”.

  - Add a new collation for the character set used by the indexed columns, and alter the columns to use that collation. For general information about adding collations, see Section 10.4, “Adding a Collation to a Character Set”. For an example specific to full-text indexing, see Section 12.9.7, “Adding a Collation for Full-Text Indexing”.

If you modify full-text variables that affect indexing (innodb_ft_min_token_size, innodb_ft_max_token_size, innodb_ft_server_stopword_table, innodb_ft_user_stopword_table, innodb_ft_enable_stopword, ft_min_word_len, ft_max_word_len, or ft_stopword_file), or if you change the stopword file itself, you must rebuild your FULLTEXT indexes after making the changes and restarting the server.

To rebuild the FULLTEXT indexes for an InnoDB table, use ALTER TABLE with the DROP INDEX and ADD INDEX options to drop and re-create each index.

To rebuild the FULLTEXT indexes for a MyISAM table, it is sufficient to do a QUICK repair operation:

```
mysql> REPAIR TABLE tbl_name QUICK;
```

Alternatively, use ALTER TABLE as just described. In some cases, this may be faster than a repair operation.

Each table that contains any FULLTEXT index must be repaired as just shown. Otherwise, queries for the table may yield incorrect results, and modifications to the table will cause the server to see the table as corrupt and in need of repair.

Note that if you use myisamchk to perform an operation that modifies MyISAM table indexes (such as repair or analyze), the FULLTEXT indexes are rebuilt using the *default* full-text parameter values for minimum word length, maximum word length, and stopword file unless you specify otherwise. This can result in queries failing.

The problem occurs because these parameters are known only by the server. They are not stored in MyISAM index files. To avoid the problem if you have modified the minimum or maximum word length or stopword file values used by the server, specify the same ft_min_word_len, ft_max_word_len, and ft_stopword_file values for myisamchk that you use for mysqld. For example, if you have set the minimum word length to 3, you can repair a table with myisamchk like this:

```
shell> myisamchk --recover --ft_min_word_len=3 tbl_name.MYI
```

To ensure that myisamchk and the server use the same values for full-text parameters, place each one in both the [mysqld] and [myisamchk] sections of an option file:

```
[mysqld]
ft_min_word_len=3

[myisamchk]
ft_min_word_len=3
```

An alternative to using myisamchk for MyISAM table index modification is to use the REPAIR TABLE, ANALYZE TABLE, OPTIMIZE TABLE, or ALTER TABLE statements. These statements are performed by the server, which knows the proper full-text parameter values to use.

12.9.7 Adding a Collation for Full-Text Indexing

This section describes how to add a new collation for full-text searches. The sample collation is like latin1_swedish_ci but treats the '-' character as a letter rather than as a punctuation character so that it can be indexed as a word character. General information about adding collations is given in Section 10.4, “Adding a Collation to a Character Set”; it is assumed that you have read it and are familiar with the files involved.

To add a collation for full-text indexing, use this procedure:

1. Add a collation to the `Index.xml` file. The collation ID must be unused, so choose a value different from 1000 if that ID is already taken on your system.

   ```
   <charset name="latin1">
   ...
   <collation name="latin1_fulltext_ci" id="1000"/>
   </charset>
   ```

2. Declare the sort order for the collation in the `latin1.xml` file. In this case, the order can be copied from `latin1_swedish_ci`:

   ```
   <collation name="latin1_fulltext_ci">
   <map>
   00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
   10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F
   20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F
   30 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F
   40 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F
   50 51 52 53 54 55 56 57 58 59 5A 5B 5C 5D 5E 5F
   60 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F
   50 51 52 53 54 55 56 57 58 59 5A 7B 7C 7D 7E 7F
   80 81 82 83 84 85 86 87 88 89 8A 8B 8C 8D 8E 8F
   90 91 92 93 94 95 96 97 98 99 9A 9B 9C 9D 9E 9F
   A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 AA AB AC AD AE AF
   B0 B1 B2 B3 B4 B5 B6 B7 B8 B9 BA BB BC BD BE BF
   41 41 41 41 5C 5B 5C 43 45 45 45 45 49 49 49 49
   44 4E 4F 4F 4F 4F 5D D7 D8 55 55 55 59 59 DE DF
   41 41 41 41 5C 5B 5C 43 45 45 45 45 49 49 49 49
   44 4E 4F 4F 4F 4F 5D F7 D8 55 55 55 59 59 DE FF
   </map>
   </collation>
   ```

3. Modify the `ctype` array in `latin1.xml`. Change the value corresponding to 0x2D (which is the code for the `'-'` character) from 10 (punctuation) to 01 (small letter). In the following array, this is the element in the fourth row down, third value from the end.

   ```
   <ctype>
   <map>
   00
   20 20 20 20 20 20 20 20 20 28 28 28 28 28 20 20
   20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
   48 10 10 10 10 10 10 10 10 10 10 10 10 01 10 10
   84 84 84 84 84 84 84 84 84 84 10 10 10 10 10 10
   10 81 81 81 81 81 81 01 01 01 01 01 01 01 01 01
   01 01 01 01 01 01 01 01 01 01 01 10 10 10 10 10
   10 82 82 82 82 82 82 02 02 02 02 02 02 02 02 02
   02 02 02 02 02 02 02 02 02 02 02 10 10 10 10 20
   10 00 10 02 10 10 10 10 10 10 01 10 01 00 01 00
   00 10 10 10 10 10 10 10 10 10 02 10 02 00 02 01
   48 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10
   10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10
   01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01
   01 01 01 01 01 01 01 10 01 01 01 01 01 01 01 02
   02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02
   02 02 02 02 02 02 02 10 02 02 02 02 02 02 02 02
   </map>
   </ctype>
   ```

4. Restart the server.

5. To employ the new collation, include it in the definition of columns that are to use it:

```
mysql> DROP TABLE IF EXISTS t1;
Query OK, 0 rows affected (0.13 sec)

mysql> CREATE TABLE t1 (
    a TEXT CHARACTER SET latin1 COLLATE latin1_fulltext_ci,
    FULLTEXT INDEX(a)
    ) ENGINE=InnoDB;
Query OK, 0 rows affected (0.47 sec)
```

6. Test the collation to verify that hyphen is considered as a word character:

```
mysql> INSERT INTO t1 VALUEs ('----'),('....'),('abcd');
Query OK, 3 rows affected (0.22 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM t1 WHERE MATCH a AGAINST ('----' IN BOOLEAN MODE);
+------+
| a    |
+------+
| ---- |
+------+
1 row in set (0.00 sec)
```

12.10 Cast Functions and Operators

Table 12.14 Cast Functions

Name	Description
`BINARY`	Cast a string to a binary string
`CAST()`	Cast a value as a certain type
`CONVERT()`	Cast a value as a certain type

- `BINARY`

  The `BINARY` operator casts the string following it to a binary string. This is an easy way to force a column comparison to be done byte by byte rather than character by character. This causes the comparison to be case sensitive even if the column is not defined as `BINARY` or `BLOB`. `BINARY` also causes trailing spaces to be significant.

```
mysql> SELECT 'a' = 'A';
        -> 1
mysql> SELECT BINARY 'a' = 'A';
        -> 0
mysql> SELECT 'a' = 'a ';
        -> 1
mysql> SELECT BINARY 'a' = 'a ';
        -> 0
```

  In a comparison, `BINARY` affects the entire operation; it can be given before either operand with the same result.

  `BINARY` *`str`* is shorthand for `CAST(`*`str`*` AS BINARY)`.

  Note that in some contexts, if you cast an indexed column to `BINARY`, MySQL is not able to use the index efficiently.

- `CAST(`*`expr`*` AS `*`type`*`)`

The `CAST()` function takes an expression of any type and produces a result value of a specified type, similar to `CONVERT()`. See the description of `CONVERT()` for more information.

- `CONVERT(expr,type)`, `CONVERT(expr USING transcoding_name)`

  The `CONVERT()` and `CAST()` functions take an expression of any type and produce a result value of a specified type.

  The *type* for the result can be one of the following values:

  - `BINARY[(N)]`
  - `CHAR[(N)]`
  - `DATE`
  - `DATETIME`
  - `DECIMAL[(M[,D])]`
  - `SIGNED [INTEGER]`
  - `TIME`
  - `UNSIGNED [INTEGER]`

  `BINARY` produces a string with the `BINARY` data type. See Section 11.4.2, “The `BINARY` and `VARBINARY` Types” for a description of how this affects comparisons. If the optional length *N* is given, `BINARY(N)` causes the cast to use no more than *N* bytes of the argument. Values shorter than *N* bytes are padded with `0x00` bytes to a length of *N*.

  `CHAR(N)` causes the cast to use no more than *N* characters of the argument.

  `CAST()` and `CONVERT(... USING ...)` are standard SQL syntax. The non-`USING` form of `CONVERT()` is ODBC syntax.

  `CONVERT()` with `USING` is used to convert data between different character sets. In MySQL, transcoding names are the same as the corresponding character set names. For example, this statement converts the string `'abc'` in the default character set to the corresponding string in the `utf8` character set:

  ```
  SELECT CONVERT('abc' USING utf8);
  ```

Normally, you cannot compare a `BLOB` value or other binary string in case-insensitive fashion because binary strings have no character set, and thus no concept of lettercase. To perform a case-insensitive comparison, use the `CONVERT()` function to convert the value to a nonbinary string. Comparisons of the result use the string collation. For example, if the character set of the result has a case-insensitive collation, a `LIKE` operation is not case sensitive:

```
SELECT 'A' LIKE CONVERT(blob_col USING latin1) FROM tbl_name;
```

To use a different character set, substitute its name for `latin1` in the preceding statement. To specify a particular collation for the converted string, use a `COLLATE` clause following the `CONVERT()` call, as described in Section 10.1.9.2, “`CONVERT()` and `CAST()`”. For example, to use `latin1_german1_ci`:

```
SELECT 'A' LIKE CONVERT(blob_col USING latin1) COLLATE latin1_german1_ci
```

```
  FROM tbl_name;
```

CONVERT() can be used more generally for comparing strings that are represented in different character sets.

LOWER() (and UPPER()) are ineffective when applied to binary strings (BINARY, VARBINARY, BLOB). To perform lettercase conversion, convert the string to a nonbinary string:

```
mysql> SET @str = BINARY 'New York';
mysql> SELECT LOWER(@str), LOWER(CONVERT(@str USING latin1));
+-------------+-----------------------------------+
| LOWER(@str) | LOWER(CONVERT(@str USING latin1)) |
+-------------+-----------------------------------+
| New York    | new york                          |
+-------------+-----------------------------------+
```

The cast functions are useful when you want to create a column with a specific type in a CREATE TABLE ... SELECT statement:

```
CREATE TABLE new_table SELECT CAST('2000-01-01' AS DATE);
```

The functions also can be useful for sorting ENUM columns in lexical order. Normally, sorting of ENUM columns occurs using the internal numeric values. Casting the values to CHAR results in a lexical sort:

```
SELECT enum_col FROM tbl_name ORDER BY CAST(enum_col AS CHAR);
```

CAST(*str* AS BINARY) is the same thing as BINARY *str*. CAST(*expr* AS CHAR) treats the expression as a string with the default character set.

CAST() also changes the result if you use it as part of a more complex expression such as CONCAT('Date: ',CAST(NOW() AS DATE)).

You should not use CAST() to extract data in different formats but instead use string functions like LEFT() or EXTRACT(). See Section 12.7, “Date and Time Functions”.

To cast a string to a numeric value in numeric context, you normally do not have to do anything other than to use the string value as though it were a number:

```
mysql> SELECT 1+'1';
       -> 2
```

If you use a string in an arithmetic operation, it is converted to a floating-point number during expression evaluation.

If you use a number in string context, the number automatically is converted to a string:

```
mysql> SELECT CONCAT('hello you ',2);
        -> 'hello you 2'
```

For information about implicit conversion of numbers to strings, see Section 12.2, “Type Conversion in Expression Evaluation”.

MySQL supports arithmetic with both signed and unsigned 64-bit values. If you are using numeric operators (such as + or -) and one of the operands is an unsigned integer, the result is unsigned by default (see Section 12.6.1, “Arithmetic Operators”). You can override this by using the SIGNED or UNSIGNED cast operator to cast a value to a signed or unsigned 64-bit integer, respectively.

```
mysql> SELECT CAST(1-2 AS UNSIGNED)
        -> 18446744073709551615
mysql> SELECT CAST(CAST(1-2 AS UNSIGNED) AS SIGNED);
        -> -1
```

If either operand is a floating-point value, the result is a floating-point value and is not affected by the preceding rule. (In this context, DECIMAL column values are regarded as floating-point values.)

```
mysql> SELECT CAST(1 AS UNSIGNED) - 2.0;
        -> -1.0
```

The SQL mode affects the result of conversion operations. Examples:

- If you convert a “zero” date string to a date, CONVERT() and CAST() return NULL and produce a warning if strict SQL mode is enabled (as of MySQL 5.7.4) or the NO_ZERO_DATE mode is enabled (before MySQL 5.7.4).

- For integer subtraction, if the NO_UNSIGNED_SUBTRACTION SQL mode is enabled, the subtraction result is signed even if any operand is unsigned.

For more information, see Section 5.1.7, “Server SQL Modes”.

12.11 XML Functions

Table 12.15 XML Functions

Name	Description
ExtractValue()	Extracts a value from an XML string using XPath notation
UpdateXML()	Return replaced XML fragment

This section discusses XML and related functionality in MySQL.

> **Note**
>
> It is possible to obtain XML-formatted output from MySQL in the mysql and mysqldump clients by invoking them with the --xml option. See Section 4.5.1, “mysql — The MySQL Command-Line Tool”, and Section 4.5.4, “mysqldump — A Database Backup Program”.

Two functions providing basic XPath 1.0 (XML Path Language, version 1.0) capabilities are available. Some basic information about XPath syntax and usage is provided later in this section; however, an in-depth discussion of these topics is beyond the scope of this Manual, and you should refer to the XML Path Language (XPath) 1.0 standard for definitive information. A useful resource for those new to XPath or who desire a refresher in the basics is the Zvon.org XPath Tutorial, which is available in several languages.

> **Note**
>
> These functions remain under development. We continue to improve these and other aspects of XML and XPath functionality in MySQL 5.7 and onwards. You may discuss these, ask questions about them, and obtain help from other users with them in the MySQL XML User Forum.

XPath expressions used with these functions support user variables and local stored program variables. User variables are weakly checked; variables local to stored programs are strongly checked (see also Bug #26518):

- **User variables (weak checking).** Variables using the syntax `$@variable_name` (that is, user variables) are not checked. No warnings or errors are issued by the server if a variable has the wrong type or has previously not been assigned a value. This also means the user is fully responsible for any typographical errors, since no warnings will be given if (for example) `$@myvariable` is used where `$@myvariable` was intended.

  Example:

```
mysql> SET @xml = '<a><b>X</b><b>Y</b></a>';
Query OK, 0 rows affected (0.00 sec)

mysql> SET @i =1, @j = 2;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @i, ExtractValue(@xml, '//b[$@i]');
+------+--------------------------------+
| @i   | ExtractValue(@xml, '//b[$@i]') |
+------+--------------------------------+
|    1 | X                              |
+------+--------------------------------+
1 row in set (0.00 sec)

mysql> SELECT @j, ExtractValue(@xml, '//b[$@j]');
+------+--------------------------------+
| @j   | ExtractValue(@xml, '//b[$@j]') |
+------+--------------------------------+
|    2 | Y                              |
+------+--------------------------------+
1 row in set (0.00 sec)

mysql> SELECT @k, ExtractValue(@xml, '//b[$@k]');
+------+--------------------------------+
| @k   | ExtractValue(@xml, '//b[$@k]') |
+------+--------------------------------+
| NULL |                                |
+------+--------------------------------+
1 row in set (0.00 sec)
```

- **Variables in stored programs (strong checking).** Variables using the syntax `$variable_name` can be declared and used with these functions when they are called inside stored programs. Such variables are local to the stored program in which they are defined, and are strongly checked for type and value.

  Example:

```
mysql> DELIMITER |

mysql> CREATE PROCEDURE myproc ()
    -> BEGIN
    ->   DECLARE i INT DEFAULT 1;
    ->   DECLARE xml VARCHAR(25) DEFAULT '<a>X</a><a>Y</a><a>Z</a>';
    ->
    ->   WHILE i < 4 DO
    ->     SELECT xml, i, ExtractValue(xml, '//a[$i]');
    ->     SET i = i+1;
    ->   END WHILE;
    -> END |
Query OK, 0 rows affected (0.01 sec)
```

```
mysql> DELIMITER ;

mysql> CALL myproc();
+--------------------------+---+------------------------------+
| xml                      | i | ExtractValue(xml, '//a[$i]') |
+--------------------------+---+------------------------------+
| <a>X</a><a>Y</a><a>Z</a> | 1 | X                            |
+--------------------------+---+------------------------------+
1 row in set (0.00 sec)

+--------------------------+---+------------------------------+
| xml                      | i | ExtractValue(xml, '//a[$i]') |
+--------------------------+---+------------------------------+
| <a>X</a><a>Y</a><a>Z</a> | 2 | Y                            |
+--------------------------+---+------------------------------+
1 row in set (0.01 sec)

+--------------------------+---+------------------------------+
| xml                      | i | ExtractValue(xml, '//a[$i]') |
+--------------------------+---+------------------------------+
| <a>X</a><a>Y</a><a>Z</a> | 3 | Z                            |
+--------------------------+---+------------------------------+
1 row in set (0.01 sec)
```

Parameters. Variables used in XPath expressions inside stored routines that are passed in as parameters are also subject to strong checking.

Expressions containing user variables or variables local to stored programs must otherwise (except for notation) conform to the rules for XPath expressions containing variables as given in the XPath 1.0 specification.

Note

Currently, a user variable used to store an XPath expression is treated as an empty string. Because of this, it is not possible to store an XPath expression as a user variable. (Bug #32911)

- `ExtractValue(xml_frag, xpath_expr)`

  `ExtractValue()` takes two string arguments, a fragment of XML markup *`xml_frag`* and an XPath expression *`xpath_expr`* (also known as a *locator*); it returns the text (`CDATA`) of the first text node which is a child of the elements or elements matched by the XPath expression.

  Using this function is the equivalent of performing a match using the *`xpath_expr`* after appending `/text()`. In other words, `ExtractValue('<a><b>Sakila</b></a>', '/a/b')` and `ExtractValue('<a><b>Sakila</b></a>', '/a/b/text()')` produce the same result.

  If multiple matches are found, the content of the first child text node of each matching element is returned (in the order matched) as a single, space-delimited string.

  If no matching text node is found for the expression (including the implicit `/text()`)—for whatever reason, as long as *`xpath_expr`* is valid, and *`xml_frag`* consists of elements which are properly nested and closed—an empty string is returned. No distinction is made between a match on an empty element and no match at all. This is by design.

  If you need to determine whether no matching element was found in *`xml_frag`* or such an element was found but contained no child text nodes, you should test the result of an expression that uses the XPath `count()` function. For example, both of these statements return an empty string, as shown here:

```
mysql> SELECT ExtractValue('<a><b/></a>', '/a/b');
+-------------------------------------+
| ExtractValue('<a><b/></a>', '/a/b') |
+-------------------------------------+
|                                     |
+-------------------------------------+
1 row in set (0.00 sec)

mysql> SELECT ExtractValue('<a><c/></a>', '/a/b');
+-------------------------------------+
| ExtractValue('<a><c/></a>', '/a/b') |
+-------------------------------------+
|                                     |
+-------------------------------------+
1 row in set (0.00 sec)
```

However, you can determine whether there was actually a matching element using the following:

```
mysql> SELECT ExtractValue('<a><b/></a>', 'count(/a/b)');
+-------------------------------------+
| ExtractValue('<a><b/></a>', 'count(/a/b)') |
+-------------------------------------+
| 1                                   |
+-------------------------------------+
1 row in set (0.00 sec)

mysql> SELECT ExtractValue('<a><c/></a>', 'count(/a/b)');
+-------------------------------------+
| ExtractValue('<a><c/></a>', 'count(/a/b)') |
+-------------------------------------+
| 0                                   |
+-------------------------------------+
1 row in set (0.01 sec)
```

⚠

Important

ExtractValue() returns only CDATA, and does not return any tags that might be contained within a matching tag, nor any of their content (see the result returned as val1 in the following example).

```
mysql> SELECT
    ->   ExtractValue('<a>ccc<b>ddd</b></a>', '/a') AS val1,
    ->   ExtractValue('<a>ccc<b>ddd</b></a>', '/a/b') AS val2,
    ->   ExtractValue('<a>ccc<b>ddd</b></a>', '//b') AS val3,
    ->   ExtractValue('<a>ccc<b>ddd</b></a>', '/b') AS val4,
    ->   ExtractValue('<a>ccc<b>ddd</b><b>eee</b></a>', '//b') AS val5;

+------+------+------+------+---------+
| val1 | val2 | val3 | val4 | val5    |
+------+------+------+------+---------+
| ccc  | ddd  | ddd  |      | ddd eee |
+------+------+------+------+---------+
```

This function uses the current SQL collation for making comparisons with contains(), performing the same collation aggregation as other string functions (such as CONCAT()), in taking into account the collation coercibility of their arguments; see Section 10.1.7.5, “Collation of Expressions”, for an explanation of the rules governing this behavior.

(Previously, binary—that is, case-sensitive—comparison was always used.)

`NULL` is returned if *`xml_frag`* contains elements which are not properly nested or closed, and a warning is generated, as shown in this example:

```
mysql> SELECT ExtractValue('<a>c</a><b', '//a');
+-----------------------------------+
| ExtractValue('<a>c</a><b', '//a') |
+-----------------------------------+
| NULL                              |
+-----------------------------------+
1 row in set, 1 warning (0.00 sec)

mysql> SHOW WARNINGS\G
*************************** 1. row ***************************
  Level: Warning
   Code: 1525
Message: Incorrect XML value: 'parse error at line 1 pos 11:
         END-OF-INPUT unexpected ('>' wanted)'
1 row in set (0.00 sec)

mysql> SELECT ExtractValue('<a>c</a><b/>', '//a');
+-------------------------------------+
| ExtractValue('<a>c</a><b/>', '//a') |
+-------------------------------------+
| c                                   |
+-------------------------------------+
1 row in set (0.00 sec)
```

- `UpdateXML(`*`xml_target`*`, `*`xpath_expr`*`, `*`new_xml`*`)`

  This function replaces a single portion of a given fragment of XML markup *`xml_target`* with a new XML fragment *`new_xml`*, and then returns the changed XML. The portion of *`xml_target`* that is replaced matches an XPath expression *`xpath_expr`* supplied by the user.

  If no expression matching *`xpath_expr`* is found, or if multiple matches are found, the function returns the original *`xml_target`* XML fragment. All three arguments should be strings.

```
mysql> SELECT
    ->   UpdateXML('<a><b>ccc</b><d></d></a>', '/a', '<e>fff</e>') AS val1,
    ->   UpdateXML('<a><b>ccc</b><d></d></a>', '/b', '<e>fff</e>') AS val2,
    ->   UpdateXML('<a><b>ccc</b><d></d></a>', '//b', '<e>fff</e>') AS val3,
    ->   UpdateXML('<a><b>ccc</b><d></d></a>', '/a/d', '<e>fff</e>') AS val4,
    ->   UpdateXML('<a><d></d><b>ccc</b><d></d></a>', '/a/d', '<e>fff</e>') AS val5
    -> \G

*************************** 1. row ***************************
val1: <e>fff</e>
val2: <a><b>ccc</b><d></d></a>
val3: <a><e>fff</e><d></d></a>
val4: <a><b>ccc</b><e>fff</e></a>
val5: <a><d></d><b>ccc</b><d></d></a>
```

Note

A discussion in depth of XPath syntax and usage are beyond the scope of this Manual. Please see the XML Path Language (XPath) 1.0 specification for definitive information. A useful resource for those new to XPath or who are wishing a refresher in the basics is the Zvon.org XPath Tutorial, which is available in several languages.

Descriptions and examples of some basic XPath expressions follow:

- `/tag`

  Matches `<tag/>` if and only if `<tag/>` is the root element.

  Example: `/a` has a match in `<a><b/></a>` because it matches the outermost (root) tag. It does not match the inner *a* element in `<b><a/></b>` because in this instance it is the child of another element.

- `/tag1/tag2`

  Matches `<tag2/>` if and only if it is a child of `<tag1/>`, and `<tag1/>` is the root element.

  Example: `/a/b` matches the *b* element in the XML fragment `<a><b/></a>` because it is a child of the root element *a*. It does not have a match in `<b><a/></b>` because in this case, *b* is the root element (and hence the child of no other element). Nor does the XPath expression have a match in `<a><c><b/></c></a>`; here, *b* is a descendant of *a*, but not actually a child of *a*.

  This construct is extendable to three or more elements. For example, the XPath expression `/a/b/c` matches the *c* element in the fragment `<a><b><c/></b></a>`.

- `//tag`

  Matches any instance of `<tag>`.

  Example: `//a` matches the *a* element in any of the following: `<a><b><c/></b></a>`; `<c><a><b/></a></b>`; `<c><b><a/></b></c>`.

  `//` can be combined with `/`. For example, `//a/b` matches the *b* element in either of the fragments `<a><b/></a>` or `<a><b><c/></b></a>`

Note

`//tag` is the equivalent of `/descendant-or-self::*/tag`. A common error is to confuse this with `/descendant-or-self::tag`, although the latter expression can actually lead to very different results, as can be seen here:

```
mysql> SET @xml = '<a><b><c>w</c><b>x</b><d>y</d>z</b></a>';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @xml;
+-----------------------------------------+
| @xml                                    |
+-----------------------------------------+
| <a><b><c>w</c><b>x</b><d>y</d>z</b></a> |
+-----------------------------------------+
1 row in set (0.00 sec)

mysql> SELECT ExtractValue(@xml, '//b[1]');
+------------------------------+
| ExtractValue(@xml, '//b[1]') |
+------------------------------+
| x z                          |
+------------------------------+
1 row in set (0.00 sec)

mysql> SELECT ExtractValue(@xml, '//b[2]');
+------------------------------+
| ExtractValue(@xml, '//b[2]') |
+------------------------------+
|                              |
+------------------------------+
1 row in set (0.01 sec)

mysql> SELECT ExtractValue(@xml, '/descendant-or-self::*/b[1]');
+---------------------------------------------------+
| ExtractValue(@xml, '/descendant-or-self::*/b[1]') |
+---------------------------------------------------+
| x z                                               |
+---------------------------------------------------+
1 row in set (0.06 sec)

mysql> SELECT ExtractValue(@xml, '/descendant-or-self::*/b[2]');
+---------------------------------------------------+
| ExtractValue(@xml, '/descendant-or-self::*/b[2]') |
+---------------------------------------------------+
|                                                   |
+---------------------------------------------------+
1 row in set (0.00 sec)


mysql> SELECT ExtractValue(@xml, '/descendant-or-self::b[1]');
+-------------------------------------------------+
| ExtractValue(@xml, '/descendant-or-self::b[1]') |
+-------------------------------------------------+
| z                                               |
+-------------------------------------------------+
1 row in set (0.00 sec)

mysql> SELECT ExtractValue(@xml, '/descendant-or-self::b[2]');
+-------------------------------------------------+
| ExtractValue(@xml, '/descendant-or-self::b[2]') |
+-------------------------------------------------+
| x                                               |
+-------------------------------------------------+
1 row in set (0.00 sec)
```

- The `*` operator acts as a “wildcard” that matches any element. For example, the expression `/*/b` matches the *b* element in either of the XML fragments `<a><b/></a>` or `<c><b/></c>`. However, the expression does not produce a match in the fragment `<b><a/></b>` because *b* must be a child of some other element. The wildcard may be used in any position: The expression `/*/b/*` will match any child of a *b* element that is itself not the root element.

- You can match any of several locators using the `|` (`UNION`) operator. For example, the expression `//b|//c` matches all *b* and *c* elements in the XML target.

- It is also possible to match an element based on the value of one or more of its attributes. This done using the syntax `tag[@attribute="value"]`. For example, the expression `//b[@id="idB"]` matches the second *b* element in the fragment `<a><b id="idA"/><c/><b id="idB"/></a>`. To match against *any* element having `attribute="value"`, use the XPath expression `//*[attribute="value"]`.

  To filter multiple attribute values, simply use multiple attribute-comparison clauses in succession. For example, the expression `//b[@c="x"][@d="y"]` matches the element `<b c="x" d="y"/>` occurring anywhere in a given XML fragment.

  To find elements for which the same attribute matches any of several values, you can use multiple locators joined by the `|` operator. For example, to match all *b* elements whose *c* attributes have either of the values 23 or 17, use the expression `//b[@c="23"]|//b[@c="17"]`. You can also use the logical `or` operator for this purpose: `//b[@c="23" or @c="17"]`.

Note

The difference between `or` and `|` is that `or` joins conditions, while `|` joins result sets.

XPath Limitations. The XPath syntax supported by these functions is currently subject to the following limitations:

- Nodeset-to-nodeset comparison (such as `'/a/b[@c=@d]'`) is not supported.
- All of the standard XPath comparison operators are supported. (Bug #22823)
- Relative locator expressions are resolved in the context of the root node. For example, consider the following query and result:

  ```
  mysql> SELECT ExtractValue(
      ->   '<a><b c="1">X</b><b c="2">Y</b></a>',
      ->    'a/b'
      -> ) AS result;
  +--------+
  | result |
  +--------+
  | X Y    |
  +--------+
  1 row in set (0.03 sec)
  ```

  In this case, the locator `a/b` resolves to `/a/b`.

  Relative locators are also supported within predicates. In the following example, `d[../@c="1"]` is resolved as `/a/b[@c="1"]/d`:

  ```
  mysql> SELECT ExtractValue(
      ->      '<a>
      ->        <b c="1"><d>X</d></b>
      ->        <b c="2"><d>X</d></b>
      ->      </a>',
      ->      'a/b/d[../@c="1"]')
      -> AS result;
  +--------+
  | result |
  +--------+
  | X      |
  +--------+
  1 row in set (0.00 sec)
  ```

- Locators prefixed with expressions that evaluate as scalar values—including variable references, literals, numbers, and scalar function calls—are not permitted, and their use results in an error.
- The `::` operator is not supported in combination with node types such as the following:
  - `*axis*::comment()`
  - `*axis*::text()`
  - `*axis*::processing-instructions()`
  - `*axis*::node()`

  However, name tests (such as `*axis*::*name*` and `*axis*::*`) are supported, as shown in these examples:

```
mysql> SELECT ExtractValue('<a><b>x</b><c>y</c></a>','/a/child::b');
+-------------------------------------------------------+
| ExtractValue('<a><b>x</b><c>y</c></a>','/a/child::b') |
+-------------------------------------------------------+
| x                                                     |
+-------------------------------------------------------+
1 row in set (0.02 sec)

mysql> SELECT ExtractValue('<a><b>x</b><c>y</c></a>','/a/child::*');
+-------------------------------------------------------+
| ExtractValue('<a><b>x</b><c>y</c></a>','/a/child::*') |
+-------------------------------------------------------+
| x y                                                   |
+-------------------------------------------------------+
1 row in set (0.01 sec)
```

- "Up-and-down" navigation is not supported in cases where the path would lead "above" the root element. That is, you cannot use expressions which match on descendants of ancestors of a given element, where one or more of the ancestors of the current element is also an ancestor of the root element (see Bug #16321).

- The following XPath functions are not supported, or have known issues as indicated:

  - `id()`
  - `lang()`
  - `local-name()`
  - `name()`
  - `namespace-uri()`
  - `normalize-space()`
  - `starts-with()`
  - `string()`
  - `substring-after()`
  - `substring-before()`
  - `translate()`

- The following axes are not supported:

  - `following-sibling`
  - `following`
  - `preceding-sibling`
  - `preceding`

XPath expressions passed as arguments to `ExtractValue()` and `UpdateXML()` may contain the colon character ("`:`") in element selectors, which enables their use with markup employing XML namespaces notation. For example:

```
mysql> SET @xml = '<a>111<b:c>222<d>333</d><e:f>444</e:f></b:c></a>';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT ExtractValue(@xml, '//e:f');
+-----------------------------+
| ExtractValue(@xml, '//e:f') |
+-----------------------------+
| 444                         |
+-----------------------------+
1 row in set (0.00 sec)

mysql> SELECT UpdateXML(@xml, '//b:c', '<g:h>555</g:h>');
+--------------------------------------------+
| UpdateXML(@xml, '//b:c', '<g:h>555</g:h>') |
+--------------------------------------------+
| <a>111<g:h>555</g:h></a>                   |
+--------------------------------------------+
1 row in set (0.00 sec)
```

This is similar in some respects to what is permitted by Apache Xalan and some other parsers, and is much simpler than requiring namespace declarations or the use of the `namespace-uri()` and `local-name()` functions.

Error handling. For both `ExtractValue()` and `UpdateXML()`, the XPath locator used must be valid and the XML to be searched must consist of elements which are properly nested and closed. If the locator is invalid, an error is generated:

```
mysql> SELECT ExtractValue('<a>c</a><b/>', '/&a');
ERROR 1105 (HY000): XPATH syntax error: '&a'
```

If *`xml_frag`* does not consist of elements which are properly nested and closed, `NULL` is returned and a warning is generated, as shown in this example:

```
mysql> SELECT ExtractValue('<a>c</a><b', '//a');
+-----------------------------------+
| ExtractValue('<a>c</a><b', '//a') |
+-----------------------------------+
| NULL                              |
+-----------------------------------+
1 row in set, 1 warning (0.00 sec)

mysql> SHOW WARNINGS\G
*************************** 1. row ***************************
  Level: Warning
   Code: 1525
Message: Incorrect XML value: 'parse error at line 1 pos 11:
         END-OF-INPUT unexpected ('>' wanted)'
1 row in set (0.00 sec)

mysql> SELECT ExtractValue('<a>c</a><b/>', '//a');
+-------------------------------------+
| ExtractValue('<a>c</a><b/>', '//a') |
+-------------------------------------+
| c                                   |
+-------------------------------------+
1 row in set (0.00 sec)
```

Important

The replacement XML used as the third argument to `UpdateXML()` is *not* checked to determine whether it consists solely of elements which are properly nested and closed.

XPath Injection. *code injection* occurs when malicious code is introduced into the system to gain unauthorized access to privileges and data. It is based on exploiting assumptions made by developers about the type and content of data input from users. XPath is no exception in this regard.

A common scenario in which this can happen is the case of application which handles authorization by matching the combination of a login name and password with those found in an XML file, using an XPath expression like this one:

```
//user[login/text()='neapolitan' and password/text()='1c3cr34m']/attribute::id
```

This is the XPath equivalent of an SQL statement like this one:

```
SELECT id FROM users WHERE login='neapolitan' AND password='1c3cr34m';
```

A PHP application employing XPath might handle the login process like this:

```
<?php

  $file     =   "users.xml";

  $login    =   $POST["login"];
  $password =   $POST["password"];

  $xpath = "//user[login/text()=$login and password/text()=$password]/attribute::id";

  if( file_exists($file) )
  {
    $xml = simplexml_load_file($file);

    if($result = $xml->xpath($xpath))
      echo "You are now logged in as user $result[0].";
    else
      echo "Invalid login name or password.";
  }
  else
    exit("Failed to open $file.");

?>
```

No checks are performed on the input. This means that a malevolent user can “short-circuit” the test by entering `' or 1=1` for both the login name and password, resulting in `$xpath` being evaluated as shown here:

```
//user[login/text()='' or 1=1 and password/text()='' or 1=1]/attribute::id
```

Since the expression inside the square brackets always evaluates as `true`, it is effectively the same as this one, which matches the `id` attribute of every `user` element in the XML document:

```
//user/attribute::id
```

One way in which this particular attack can be circumvented is simply by quoting the variable names to be interpolated in the definition of `$xpath`, forcing the values passed from a Web form to be converted to strings:

```
$xpath = "//user[login/text()='$login' and password/text()='$password']/attribute::id";
```

This is the same strategy that is often recommended for preventing SQL injection attacks. In general, the practices you should follow for preventing XPath injection attacks are the same as for preventing SQL injection:

- Never accepted untested data from users in your application.
- Check all user-submitted data for type; reject or convert data that is of the wrong type
- Test numeric data for out of range values; truncate, round, or reject values that are out of range. Test strings for illegal characters and either strip them out or reject input containing them.
- Do not output explicit error messages that might provide an unauthorized user with clues that could be used to compromise the system; log these to a file or database table instead.

Just as SQL injection attacks can be used to obtain information about database schemas, so can XPath injection be used to traverse XML files to uncover their structure, as discussed in Amit Klein's paper Blind XPath Injection (PDF file, 46KB).

It is also important to check the output being sent back to the client. Consider what can happen when we use the MySQL `ExtractValue()` function:

```
mysql> SELECT ExtractValue(
    ->     LOAD_FILE('users.xml'),
    ->     '//user[login/text()="" or 1=1 and password/text()="" or 1=1]/attribute::id'
    -> ) AS id;
+-------------------------------+
| id                            |
+-------------------------------+
| 00327 13579 02403 42354 28570 |
+-------------------------------+
1 row in set (0.01 sec)
```

Because `ExtractValue()` returns multiple matches as a single space-delimited string, this injection attack provides every valid ID contained within `users.xml` to the user as a single row of output. As an extra safeguard, you should also test output before returning it to the user. Here is a simple example:

```
mysql> SELECT @id = ExtractValue(
    ->     LOAD_FILE('users.xml'),
    ->     '//user[login/text()="" or 1=1 and password/text()="" or 1=1]/attribute::id'
    -> );
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT IF(
    ->     INSTR(@id, ' ') = 0,
    ->     @id,
    ->     'Unable to retrieve user ID')
    -> AS singleID;
+----------------------------+
| singleID                   |
+----------------------------+
| Unable to retrieve user ID |
+----------------------------+
1 row in set (0.00 sec)
```

In general, the guidelines for returning data to users securely are the same as for accepting user input. These can be summed up as:

- Always test outgoing data for type and permissible values.
- Never permit unauthorized users to view error messages that might provide information about the application that could be used to exploit it.

12.12 Bit Functions

Table 12.16 Bitwise Functions

Name	Description
BIT_COUNT()	Return the number of bits that are set
&	Bitwise AND
~	Invert bits
|	Bitwise OR
^	Bitwise XOR
<<	Left shift
>>	Right shift

MySQL uses BIGINT (64-bit) arithmetic for bit operations, so these operators have a maximum range of 64 bits.

- |

  Bitwise OR:

  ```
  mysql> SELECT 29 | 15;
          -> 31
  ```

  The result is an unsigned 64-bit integer.

- &

  Bitwise AND:

  ```
  mysql> SELECT 29 & 15;
          -> 13
  ```

  The result is an unsigned 64-bit integer.

- ^

  Bitwise XOR:

  ```
  mysql> SELECT 1 ^ 1;
          -> 0
  mysql> SELECT 1 ^ 0;
          -> 1
  mysql> SELECT 11 ^ 3;
          -> 8
  ```

  The result is an unsigned 64-bit integer.

- <<

  Shifts a longlong (BIGINT) number to the left.

  ```
  mysql> SELECT 1 << 2;
  ```

```
        -> 4
```

The result is an unsigned 64-bit integer. The value is truncated to 64 bits. In particular, if the shift count is greater or equal to the width of an unsigned 64-bit number, the result is zero.

- `>>`

  Shifts a longlong (`BIGINT`) number to the right.

  ```
  mysql> SELECT 4 >> 2;
          -> 1
  ```

  The result is an unsigned 64-bit integer. The value is truncated to 64 bits. In particular, if the shift count is greater or equal to the width of an unsigned 64-bit number, the result is zero.

- `~`

  Invert all bits.

  ```
  mysql> SELECT 5 & ~1;
          -> 4
  ```

  The result is an unsigned 64-bit integer.

- `BIT_COUNT(N)`

  Returns the number of bits that are set in the argument *`N`*.

  ```
  mysql> SELECT BIT_COUNT(29), BIT_COUNT(b'101010');
          -> 4, 3
  ```

12.13 Encryption and Compression Functions

Table 12.17 Encryption Functions

Name	Description
`AES_DECRYPT()`	Decrypt using AES
`AES_ENCRYPT()`	Encrypt using AES
`COMPRESS()`	Return result as a binary string
`DECODE()`	Decodes a string encrypted using ENCODE()
`DES_DECRYPT()`	Decrypt a string
`DES_ENCRYPT()`	Encrypt a string
`ENCODE()`	Encode a string
`ENCRYPT()`	Encrypt a string
`MD5()`	Calculate MD5 checksum
`OLD_PASSWORD()`	Return the value of the pre-4.1 implementation of PASSWORD
`PASSWORD()`	Calculate and return a password string
`RANDOM_BYTES()`	Return a random byte vector
`SHA1()`, `SHA()`	Calculate an SHA-1 160-bit checksum

Name	Description
SHA2()	Calculate an SHA-2 checksum
UNCOMPRESS()	Uncompress a string compressed
UNCOMPRESSED_LENGTH()	Return the length of a string before compression
VALIDATE_PASSWORD_STRENGTH()	Determine strength of password

Many encryption and compression functions return strings for which the result might contain arbitrary byte values. If you want to store these results, use a column with a VARBINARY or BLOB binary string data type. This will avoid potential problems with trailing space removal or character set conversion that would change data values, such as may occur if you use a nonbinary string data type (CHAR, VARCHAR, TEXT).

Some encryption functions return strings of ASCII characters: MD5(), OLD_PASSWORD(), PASSWORD(), SHA(), SHA1(), SHA2(). In MySQL 5.7, their return value is a nonbinary string that has a character set and collation determined by the character_set_connection and collation_connection system variables.

For versions in which functions such as MD5() or SHA1() return a string of hex digits as a binary string, the return value cannot be converted to uppercase or compared in case-insensitive fashion as is. You must convert the value to a nonbinary string. See the discussion of binary string conversion in Section 12.10, “Cast Functions and Operators”.

If an application stores values from a function such as MD5() or SHA1() that returns a string of hex digits, more efficient storage and comparisons can be obtained by converting the hex representation to binary using UNHEX() and storing the result in a BINARY(*N*) column. Each pair of hex digits requires one byte in binary form, so the value of *N* depends on the length of the hex string. *N* is 16 for an MD5() value and 20 for a SHA1() value. For SHA2(), *N* ranges from 28 to 32 depending on the argument specifying the desired bit length of the result.

The size penalty for storing the hex string in a CHAR column is at least two times, up to eight times if the value is stored in a column that uses the utf8 character set (where each character uses 4 bytes). Storing the string also results in slower comparisons because of the larger values and the need to take character set collation rules into account.

Suppose that an application stores MD5() string values in a CHAR(32) column:

```
CREATE TABLE md5_tbl (md5_val CHAR(32), ...);
INSERT INTO md5_tbl (md5_val, ...) VALUES(MD5('abcdef'), ...);
```

To convert hex strings to more compact form, modify the application to use UNHEX() and BINARY(16) instead as follows:

```
CREATE TABLE md5_tbl (md5_val BINARY(16), ...);
INSERT INTO md5_tbl (md5_val, ...) VALUES(UNHEX(MD5('abcdef')), ...);
```

Applications should be prepared to handle the very rare case that a hashing function produces the same value for two different input values. One way to make collisions detectable is to make the hash column a primary key.

Note

Exploits for the MD5 and SHA-1 algorithms have become known. You may wish to consider using one of the other encryption functions described in this section instead, such as SHA2().

Caution

Passwords or other sensitive values supplied as arguments to encryption functions are sent in plaintext to the MySQL server unless an SSL connection is used. Also, such values will appear in any MySQL logs to which they are written. To avoid these types of exposure, applications can encrypt sensitive values on the client side before sending them to the server. The same considerations apply to encryption keys. To avoid exposing these, applications can use stored procedures to encrypt and decrypt values on the server side.

- `AES_DECRYPT(crypt_str,key_str[,init_vector])`

  This function decrypts data using the official AES (Advanced Encryption Standard) algorithm. For more information, see the description of `AES_ENCRYPT()`.

  The optional initialization vector argument, *`init_vector`*, is available as of MySQL 5.7.4. As of that version, statements that use `AES_DECRYPT()` are unsafe for statement-based replication and cannot be stored in the query cache.

- `AES_ENCRYPT(str,key_str[,init_vector])`

  `AES_ENCRYPT()` and `AES_DECRYPT()` implement encryption and decryption of data using the official AES (Advanced Encryption Standard) algorithm, previously known as “Rijndael.” The AES standard permits various key lengths. By default these functions implement AES with a 128-bit key length. As of MySQL 5.7.4, key lengths of 196 or 256 bits can be used, as described later. The key length is a trade off between performance and security.

  `AES_ENCRYPT()` encrypts the string *`str`* using the key string *`key_str`* and returns a binary string containing the encrypted output. `AES_DECRYPT()` decrypts the encrypted string *`crypt_str`* using the key string *`key_str`* and returns the original plaintext string. If either function argument is `NULL`, the function returns `NULL`.

  The *`str`* and *`crypt_str`* arguments can be any length, and padding is automatically added to *`str`* so it is a multiple of a block as required by block-based algorithms such as AES. This padding is automatically removed by the `AES_DECRYPT()` function. The length of *`crypt_str`* can be calculated using this formula:

  ```
  16 * (trunc(string_length / 16) + 1)
  ```

  For a key length of 128 bits, the most secure way to pass a key to the *`key_str`* argument is to create a truly random 128-bit value and pass it as a binary value. For example:

  ```
  INSERT INTO t
  VALUES (1,AES_ENCRYPT('text',UNHEX('F3229A0B371ED2D9441B830D21A390C3')));
  ```

  A passphrase can be used to generate an AES key by hashing the passphrase. For example:

  ```
  INSERT INTO t VALUES (1,AES_ENCRYPT('text', SHA2('My secret passphrase',512)));
  ```

  Do not pass a password or passphrase directly to *`crypt_str`*, hash it first. Previous versions of this documentation suggested the former approach, but it is no longer recommended as the examples shown here are more secure.

  If `AES_DECRYPT()` detects invalid data or incorrect padding, it returns `NULL`. However, it is possible for `AES_DECRYPT()` to return a non-`NULL` value (possibly garbage) if the input data or the key is invalid.

As of MySQL 5.7.4, AES_ENCRYPT() and AES_DECRYPT() permit control of the block encryption mode and take an optional *init_vector* initialization vector argument:

- The block_encryption_mode system variable controls the mode for block-based encryption algorithms. Its default value is aes-128-ecb, which signifies encryption using a key length of 128 bits and ECB mode. For a description of the permitted values of this variable, see Section 5.1.4, “Server System Variables”.

- The optional *init_vector* argument provides an initialization vector for block encryption modes that require it.

For modes that require the optional *init_vector* argument, it must be 16 bytes or longer (bytes in excess of 16 are ignored). An error occurs if *init_vector* is missing.

For modes that do not require *init_vector*, it is ignored and a warning is generated if it is specified.

A random string of bytes to use for the initialization vector can be produced by calling RANDOM_BYTES(16). For encryption modes that require an initialization vector, the same vector must be used for encryption and decryption.

```
mysql> SET block_encryption_mode = 'aes-256-cbc';
mysql> SET @key_str = SHA2('My secret passphrase',512);
mysql> SET @init_vector = RANDOM_BYTES(16);
mysql> SET @crypt_str = AES_ENCRYPT('text',@key_str,@init_vector);
mysql> SELECT AES_DECRYPT(@crypt_str,@key_str,@init_vector);
+-----------------------------------------------+
| AES_DECRYPT(@crypt_str,@key_str,@init_vector) |
+-----------------------------------------------+
| text                                          |
+-----------------------------------------------+
```

The following table lists each permitted block encryption mode, the SSL libraries that support it, and whether the initialization vector argument is required.

Block Encryption Mode	SSL Libraries that Support Mode	Initialization Vector Required
ECB	OpenSSL, yaSSL	No
CBC	OpenSSL, yaSSL	Yes
CFB1	OpenSSL	Yes
CFB8	OpenSSL	Yes
CFB128	OpenSSL	Yes
OFB	OpenSSL	Yes

As of MySQL 5.7.4, statements that use AES_ENCRYPT() or AES_DECRYPT() are unsafe for statement-based replication and cannot be stored in the query cache.

- COMPRESS(*string_to_compress*)

  Compresses a string and returns the result as a binary string. This function requires MySQL to have been compiled with a compression library such as zlib. Otherwise, the return value is always NULL. The compressed string can be uncompressed with UNCOMPRESS().

```
mysql> SELECT LENGTH(COMPRESS(REPEAT('a',1000)));
        -> 21
```

```
mysql> SELECT LENGTH(COMPRESS(''));
        -> 0
mysql> SELECT LENGTH(COMPRESS('a'));
        -> 13
mysql> SELECT LENGTH(COMPRESS(REPEAT('a',16)));
        -> 15
```

The compressed string contents are stored the following way:

- Empty strings are stored as empty strings.
- Nonempty strings are stored as a 4-byte length of the uncompressed string (low byte first), followed by the compressed string. If the string ends with space, an extra "`.`" character is added to avoid problems with endspace trimming should the result be stored in a `CHAR` or `VARCHAR` column. (However, use of nonbinary string data types such as `CHAR` or `VARCHAR` to store compressed strings is not recommended anyway because character set conversion may occur. Use a `VARBINARY` or `BLOB` binary string column instead.)

- `DECODE(crypt_str,pass_str)`

  `DECODE()` decrypts the encrypted string *`crypt_str`* using *`pass_str`* as the password. *`crypt_str`* should be a string returned from `ENCODE()`.

  Note

  The `ENCODE()` and `DECODE()` functions are deprecated in MySQL 5.7, will be removed in a future MySQL release, and should no longer be used.

- `DES_DECRYPT(crypt_str[,key_str])`

  Decrypts a string encrypted with `DES_ENCRYPT()`. If an error occurs, this function returns `NULL`.

  This function works only if MySQL has been configured with SSL support. See Section 6.3.11, "Using SSL for Secure Connections".

  If no *`key_str`* argument is given, `DES_DECRYPT()` examines the first byte of the encrypted string to determine the DES key number that was used to encrypt the original string, and then reads the key from the DES key file to decrypt the message. For this to work, the user must have the `SUPER` privilege. The key file can be specified with the `--des-key-file` server option.

  If you pass this function a *`key_str`* argument, that string is used as the key for decrypting the message.

  If the *`crypt_str`* argument does not appear to be an encrypted string, MySQL returns the given *`crypt_str`*.

- `DES_ENCRYPT(str[,{key_num|key_str}])`

  Encrypts the string with the given key using the Triple-DES algorithm.

  This function works only if MySQL has been configured with SSL support. See Section 6.3.11, "Using SSL for Secure Connections".

  The encryption key to use is chosen based on the second argument to `DES_ENCRYPT()`, if one was given. With no argument, the first key from the DES key file is used. With a *`key_num`* argument, the given key number (0 to 9) from the DES key file is used. With a *`key_str`* argument, the given key string is used to encrypt *`str`*.

  The key file can be specified with the `--des-key-file` server option.

The return string is a binary string where the first character is `CHAR(128 | key_num)`. If an error occurs, `DES_ENCRYPT()` returns `NULL`.

The 128 is added to make it easier to recognize an encrypted key. If you use a string key, *`key_num`* is 127.

The string length for the result is given by this formula:

```
new_len = orig_len + (8 - (orig_len % 8)) + 1
```

Each line in the DES key file has the following format:

```
key_num des_key_str
```

Each *`key_num`* value must be a number in the range from `0` to `9`. Lines in the file may be in any order. *`des_key_str`* is the string that is used to encrypt the message. There should be at least one space between the number and the key. The first key is the default key that is used if you do not specify any key argument to `DES_ENCRYPT()`.

You can tell MySQL to read new key values from the key file with the `FLUSH DES_KEY_FILE` statement. This requires the `RELOAD` privilege.

One benefit of having a set of default keys is that it gives applications a way to check for the existence of encrypted column values, without giving the end user the right to decrypt those values.

```
mysql> SELECT customer_address FROM customer_table
     > WHERE crypted_credit_card = DES_ENCRYPT('credit_card_number');
```

- `ENCODE(str,pass_str)`

  `ENCODE()` encrypts *`str`* using *`pass_str`* as the password. The result is a binary string of the same length as *`str`*. To decrypt the result, use `DECODE()`.

  Note

  The `ENCODE()` and `DECODE()` functions are deprecated in MySQL 5.7, will be removed in a future MySQL release, and should no longer be used.

  If you still need to use `ENCODE()`, a salt value must be used with it to reduce risk. For example:

```
ENCODE('plaintext', CONCAT('my_random_salt','my_secret_password'))
```

  A new random salt value must be used whenever a password is updated.

- `ENCRYPT(str[,salt])`

  Encrypts *`str`* using the Unix `crypt()` system call and returns a binary string. The *`salt`* argument must be a string with at least two characters or the result will be `NULL`. If no *`salt`* argument is given, a random value is used.

```
mysql> SELECT ENCRYPT('hello');
        -> 'VxuFAJXVARROc'
```

ENCRYPT() ignores all but the first eight characters of *str*, at least on some systems. This behavior is determined by the implementation of the underlying crypt() system call.

The use of ENCRYPT() with the ucs2, utf16, utf16le, or utf32 multi-byte character sets is not recommended because the system call expects a string terminated by a zero byte.

If crypt() is not available on your system (as is the case with Windows), ENCRYPT() always returns NULL.

- MD5(*str*)

  Calculates an MD5 128-bit checksum for the string. The value is returned as a string of 32 hex digits, or NULL if the argument was NULL. The return value can, for example, be used as a hash key. See the notes at the beginning of this section about storing hash values efficiently.

  The return value is a nonbinary string in the connection character set.

  ```
  mysql> SELECT MD5('testing');
          -> 'ae2b1fca515949e5d54fb22b8ed95575'
  ```

  This is the "RSA Data Security, Inc. MD5 Message-Digest Algorithm."

  See the note regarding the MD5 algorithm at the beginning this section.

- OLD_PASSWORD(*str*)

  OLD_PASSWORD() was added when the implementation of PASSWORD() was changed in MySQL 4.1 to improve security. OLD_PASSWORD() returns the value of the pre-4.1 implementation of PASSWORD() as a string, and is intended to permit you to reset passwords for any pre-4.1 clients that need to connect to your version 5.7 MySQL server without locking them out. See Section 6.1.2.4, "Password Hashing in MySQL".

  The return value is a nonbinary string in the connection character set.

  Note

  Passwords that use the pre-4.1 hashing method are less secure than passwords that use the native password hashing method and should be avoided. Pre-4.1 passwords are deprecated and support for them will be removed in a future MySQL release.

- PASSWORD(*str*)

  Returns a hashed password string calculated from the cleartext password *str*. The return value is a nonbinary string in the connection character set, or NULL if the argument is NULL. This function is the SQL interface to the algorithm used by the server to encrypt MySQL passwords for storage in the mysql.user grant table.

  The old_passwords system variable controls the password hashing method used by the PASSWORD() function. It also influences password hashing performed by CREATE USER and GRANT statements that specify a password using an IDENTIFIED BY clause.

  The following table shows the permitted values of old_passwords, the password hashing method for each value, and which authentication plugins use passwords hashed with each method.

Value	Password Hashing Method	Associated Authentication Plugin
0	MySQL 4.1 native hashing	mysql_native_password
1	Pre-4.1 (“old”) hashing	mysql_old_password
2	SHA-256 hashing	sha256_password

If old_passwords=1, PASSWORD(*str*) returns the same value as OLD_PASSWORD(*str*). The latter function is not affected by the value of old_passwords.

```
mysql> SET old_passwords = 0;
mysql> SELECT PASSWORD('mypass'), OLD_PASSWORD('mypass');
+-------------------------------------------+------------------------+
| PASSWORD('mypass')                        | OLD_PASSWORD('mypass') |
+-------------------------------------------+------------------------+
| *6C8989366EAF75BB670AD8EA7A7FC1176A95CEF4 | 6f8c114b58f2ce9e       |
+-------------------------------------------+------------------------+

mysql> SET old_passwords = 1;
mysql> SELECT PASSWORD('mypass'), OLD_PASSWORD('mypass');
+--------------------+------------------------+
| PASSWORD('mypass') | OLD_PASSWORD('mypass') |
+--------------------+------------------------+
| 6f8c114b58f2ce9e   | 6f8c114b58f2ce9e       |
+--------------------+------------------------+
```

SHA-256 password hashing (old_passwords=2) uses a random salt value, which makes the result from PASSWORD() nondeterministic. Consequently, statements that use this function are not safe for statement-based replication and cannot be stored in the query cache.

Encryption performed by PASSWORD() is one-way (not reversible). It is not the same type of encryption used for Unix passwords; for that, use ENCRYPT().

Note

The PASSWORD() function is used by the authentication system in MySQL Server; you should *not* use it in your own applications. For that purpose, consider MD5() or SHA2() instead. Also see RFC 2195, section 2 (Challenge-Response Authentication Mechanism (CRAM)), for more information about handling passwords and authentication securely in your applications.

Note

Passwords that use the pre-4.1 hashing method are less secure than passwords that use the native password hashing method and should be avoided. Pre-4.1 passwords are deprecated and support for them will be removed in a future MySQL release. For account upgrade instructions, see Section 6.3.9.3, “Migrating Away from Pre-4.1 Password Hashing and the mysql_old_password Plugin”.

Caution

Statements that invoke PASSWORD() may be recorded in server logs or in a history file such as ~/.mysql_history, which means that cleartext passwords may be read by anyone having read access to that information. See Section 6.1.2, “Keeping Passwords Secure”.

- RANDOM_BYTES(*len*)

This function returns a binary string of *len* random bytes generated using the random number generator of the SSL library (OpenSSL or yaSSL). Permitted values of *len* range from 1 to 1024. For values outside that range, RANDOM_BYTES() generates a warning and returns NULL.

RANDOM_BYTES() can be used to provide the initialization vector for the AES_DECRYPT() and AES_ENCRYPT() functions. For use in that context, *len* must be at least 16. Larger values are permitted, but bytes in excess of 16 are ignored.

RANDOM_BYTES() generates a random value, which makes its result nondeterministic. Consequently, statements that use this function are unsafe for statement-based replication and cannot be stored in the query cache.

This function is available as of MySQL 5.7.4.

- SHA1(*str*), SHA(*str*)

  Calculates an SHA-1 160-bit checksum for the string, as described in RFC 3174 (Secure Hash Algorithm). The value is returned as a string of 40 hex digits, or NULL if the argument was NULL. One of the possible uses for this function is as a hash key. See the notes at the beginning of this section about storing hash values efficiently. You can also use SHA1() as a cryptographic function for storing passwords. SHA() is synonymous with SHA1().

  The return value is a nonbinary string in the connection character set.

```
mysql> SELECT SHA1('abc');
        -> 'a9993e364706816aba3e25717850c26c9cd0d89d'
```

  SHA1() can be considered a cryptographically more secure equivalent of MD5(). However, see the note regarding the MD5 and SHA-1 algorithms at the beginning this section.

- SHA2(*str*, *hash_length*)

  Calculates the SHA-2 family of hash functions (SHA-224, SHA-256, SHA-384, and SHA-512). The first argument is the cleartext string to be hashed. The second argument indicates the desired bit length of the result, which must have a value of 224, 256, 384, 512, or 0 (which is equivalent to 256). If either argument is NULL or the hash length is not one of the permitted values, the return value is NULL. Otherwise, the function result is a hash value containing the desired number of bits. See the notes at the beginning of this section about storing hash values efficiently.

  The return value is a nonbinary string in the connection character set.

```
mysql> SELECT SHA2('abc', 224);
        -> '23097d223405d8228642a477bda255b32aadbce4bda0b3f7e36c9da7'
```

  This function works only if MySQL has been configured with SSL support. See Section 6.3.11, “Using SSL for Secure Connections”.

  SHA2() can be considered cryptographically more secure than MD5() or SHA1().

- UNCOMPRESS(*string_to_uncompress*)

  Uncompresses a string compressed by the COMPRESS() function. If the argument is not a compressed value, the result is NULL. This function requires MySQL to have been compiled with a compression library such as zlib. Otherwise, the return value is always NULL.

```
mysql> SELECT UNCOMPRESS(COMPRESS('any string'));
        -> 'any string'
mysql> SELECT UNCOMPRESS('any string');
        -> NULL
```

- UNCOMPRESSED_LENGTH(*compressed_string*)

  Returns the length that the compressed string had before being compressed.

  ```
  mysql> SELECT UNCOMPRESSED_LENGTH(COMPRESS(REPEAT('a',30)));
          -> 30
  ```

- VALIDATE_PASSWORD_STRENGTH(*str*)

  Given an argument representing a cleartext password, this function returns an integer to indicate how strong the password is. The return value ranges from 0 (weak) to 100 (strong).

  The password is subjected to increasingly strict tests and the return value reflects which tests were satisfied, as shown in the following table.

Password Test	**Return Value**
Length < 4	0
Length ≥ 4 and < validate_password_length	25
Satisfies policy 1 (LOW)	50
Satisfies policy 2 (MEDIUM)	75
Satisfies policy 3 (STRONG)	100

  Password assessment by VALIDATE_PASSWORD_STRENGTH() is done by the validate_password plugin. If that plugin is not installed, the function always returns 0. For information about installing the validate_password plugin, see Section 6.1.2.6, “The Password Validation Plugin”. To examine or configure the parameters that affect password testing, check or set the system variables implemented by validate_password plugin. See Password Validation Plugin Options and Variables.

12.14 Information Functions

Table 12.18 Information Functions

Name	**Description**
BENCHMARK()	Repeatedly execute an expression
CHARSET()	Return the character set of the argument
COERCIBILITY()	Return the collation coercibility value of the string argument
COLLATION()	Return the collation of the string argument
CONNECTION_ID()	Return the connection ID (thread ID) for the connection
CURRENT_USER(), CURRENT_USER	The authenticated user name and host name
DATABASE()	Return the default (current) database name
FOUND_ROWS()	For a SELECT with a LIMIT clause, the number of rows that would be returned were there no LIMIT clause
LAST_INSERT_ID()	Value of the AUTOINCREMENT column for the last INSERT
ROW_COUNT()	The number of rows updated

Name	Description
SCHEMA()	Synonym for DATABASE()
SESSION_USER()	Synonym for USER()
SYSTEM_USER()	Synonym for USER()
USER()	The user name and host name provided by the client
VERSION()	Returns a string that indicates the MySQL server version

- BENCHMARK(*count*,*expr*)

  The BENCHMARK() function executes the expression *expr* repeatedly *count* times. It may be used to time how quickly MySQL processes the expression. The result value is always 0. The intended use is from within the mysql client, which reports query execution times:

  ```
  mysql> SELECT BENCHMARK(1000000,ENCODE('hello','goodbye'));
  +-----------------------------------------------+
  | BENCHMARK(1000000,ENCODE('hello','goodbye')) |
  +-----------------------------------------------+
  |                                             0 |
  +-----------------------------------------------+
  1 row in set (4.74 sec)
  ```

  The time reported is elapsed time on the client end, not CPU time on the server end. It is advisable to execute BENCHMARK() several times, and to interpret the result with regard to how heavily loaded the server machine is.

  BENCHMARK() is intended for measuring the runtime performance of scalar expressions, which has some significant implications for the way that you use it and interpret the results:

  - Only scalar expressions can be used. Although the expression can be a subquery, it must return a single column and at most a single row. For example, BENCHMARK(10, (SELECT * FROM t)) will fail if the table t has more than one column or more than one row.

  - Executing a SELECT *expr* statement *N* times differs from executing SELECT BENCHMARK(*N*, *expr*) in terms of the amount of overhead involved. The two have very different execution profiles and you should not expect them to take the same amount of time. The former involves the parser, optimizer, table locking, and runtime evaluation *N* times each. The latter involves only runtime evaluation *N* times, and all the other components just once. Memory structures already allocated are reused, and runtime optimizations such as local caching of results already evaluated for aggregate functions can alter the results. Use of BENCHMARK() thus measures performance of the runtime component by giving more weight to that component and removing the “noise” introduced by the network, parser, optimizer, and so forth.

- CHARSET(*str*)

  Returns the character set of the string argument.

  ```
  mysql> SELECT CHARSET('abc');
          -> 'latin1'
  mysql> SELECT CHARSET(CONVERT('abc' USING utf8));
          -> 'utf8'
  mysql> SELECT CHARSET(USER());
          -> 'utf8'
  ```

- COERCIBILITY(*str*)

Returns the collation coercibility value of the string argument.

```
mysql> SELECT COERCIBILITY('abc' COLLATE latin1_swedish_ci);
        -> 0
mysql> SELECT COERCIBILITY(USER());
        -> 3
mysql> SELECT COERCIBILITY('abc');
        -> 4
```

The return values have the meanings shown in the following table. Lower values have higher precedence.

Coercibility	Meaning	Example
`0`	Explicit collation	Value with `COLLATE` clause
`1`	No collation	Concatenation of strings with different collations
`2`	Implicit collation	Column value, stored routine parameter or local variable
`3`	System constant	`USER()` return value
`4`	Coercible	Literal string
`5`	Ignorable	`NULL` or an expression derived from `NULL`

- `COLLATION(`*`str`*`)`

  Returns the collation of the string argument.

```
mysql> SELECT COLLATION('abc');
        -> 'latin1_swedish_ci'
mysql> SELECT COLLATION(_utf8'abc');
        -> 'utf8_general_ci'
```

- `CONNECTION_ID()`

  Returns the connection ID (thread ID) for the connection. Every connection has an ID that is unique among the set of currently connected clients.

```
mysql> SELECT CONNECTION_ID();
        -> 23786
```

- `CURRENT_USER`, `CURRENT_USER()`

  Returns the user name and host name combination for the MySQL account that the server used to authenticate the current client. This account determines your access privileges. The return value is a string in the `utf8` character set.

  The value of `CURRENT_USER()` can differ from the value of `USER()`.

```
mysql> SELECT USER();
        -> 'davida@localhost'
mysql> SELECT * FROM mysql.user;
ERROR 1044: Access denied for user ''@'localhost' to
database 'mysql'
```

```
mysql> SELECT CURRENT_USER();
        -> '@localhost'
```

The example illustrates that although the client specified a user name of davida (as indicated by the value of the USER() function), the server authenticated the client using an anonymous user account (as seen by the empty user name part of the CURRENT_USER() value). One way this might occur is that there is no account listed in the grant tables for davida.

Within a stored program or view, CURRENT_USER() returns the account for the user who defined the object (as given by its DEFINER value) unless defined with the SQL SECURITY INVOKER characteristic. In the latter case, CURRENT_USER() returns the object's invoker.

Triggers and events have no option to define the SQL SECURITY characteristic, so for these objects, CURRENT_USER() returns the account for the user who defined the object. To return the invoker, use USER() or SESSION_USER().

The following statements support use of the CURRENT_USER() function to take the place of the name of (and, possibly, a host for) an affected user or a definer; in such cases, CURRENT_USER() is expanded where and as needed:

- DROP USER
- RENAME USER
- GRANT
- REVOKE
- CREATE FUNCTION
- CREATE PROCEDURE
- CREATE TRIGGER
- CREATE EVENT
- CREATE VIEW
- ALTER EVENT
- ALTER VIEW
- SET PASSWORD

For information about the implications that this expansion of CURRENT_USER() has for replication in different releases of MySQL 5.7, see Section 16.4.1.7, “Replication of CURRENT_USER()”.

- DATABASE()

Returns the default (current) database name as a string in the utf8 character set. If there is no default database, DATABASE() returns NULL. Within a stored routine, the default database is the database that the routine is associated with, which is not necessarily the same as the database that is the default in the calling context.

```
mysql> SELECT DATABASE();
        -> 'test'
```

If there is no default database, DATABASE() returns NULL.

- FOUND_ROWS()

  A SELECT statement may include a LIMIT clause to restrict the number of rows the server returns to the client. In some cases, it is desirable to know how many rows the statement would have returned without the LIMIT, but without running the statement again. To obtain this row count, include a SQL_CALC_FOUND_ROWS option in the SELECT statement, and then invoke FOUND_ROWS() afterward:

```
mysql> SELECT SQL_CALC_FOUND_ROWS * FROM tbl_name
    -> WHERE id > 100 LIMIT 10;
mysql> SELECT FOUND_ROWS();
```

  The second SELECT returns a number indicating how many rows the first SELECT would have returned had it been written without the LIMIT clause.

  In the absence of the SQL_CALC_FOUND_ROWS option in the most recent successful SELECT statement, FOUND_ROWS() returns the number of rows in the result set returned by that statement. If the statement includes a LIMIT clause, FOUND_ROWS() returns the number of rows up to the limit. For example, FOUND_ROWS() returns 10 or 60, respectively, if the statement includes LIMIT 10 or LIMIT 50, 10.

  The row count available through FOUND_ROWS() is transient and not intended to be available past the statement following the SELECT SQL_CALC_FOUND_ROWS statement. If you need to refer to the value later, save it:

```
mysql> SELECT SQL_CALC_FOUND_ROWS * FROM ... ;
mysql> SET @rows = FOUND_ROWS();
```

  If you are using SELECT SQL_CALC_FOUND_ROWS, MySQL must calculate how many rows are in the full result set. However, this is faster than running the query again without LIMIT, because the result set need not be sent to the client.

  SQL_CALC_FOUND_ROWS and FOUND_ROWS() can be useful in situations when you want to restrict the number of rows that a query returns, but also determine the number of rows in the full result set without running the query again. An example is a Web script that presents a paged display containing links to the pages that show other sections of a search result. Using FOUND_ROWS() enables you to determine how many other pages are needed for the rest of the result.

  The use of SQL_CALC_FOUND_ROWS and FOUND_ROWS() is more complex for UNION statements than for simple SELECT statements, because LIMIT may occur at multiple places in a UNION. It may be applied to individual SELECT statements in the UNION, or global to the UNION result as a whole.

  The intent of SQL_CALC_FOUND_ROWS for UNION is that it should return the row count that would be returned without a global LIMIT. The conditions for use of SQL_CALC_FOUND_ROWS with UNION are:

  - The SQL_CALC_FOUND_ROWS keyword must appear in the first SELECT of the UNION.

  - The value of FOUND_ROWS() is exact only if UNION ALL is used. If UNION without ALL is used, duplicate removal occurs and the value of FOUND_ROWS() is only approximate.

  - If no LIMIT is present in the UNION, SQL_CALC_FOUND_ROWS is ignored and returns the number of rows in the temporary table that is created to process the UNION.

  Beyond the cases described here, the behavior of FOUND_ROWS() is undefined (for example, its value following a SELECT statement that fails with an error).

⚠ **Important**

FOUND_ROWS() is not replicated reliably using statement-based replication. This function is automatically replicated using row-based replication.

- LAST_INSERT_ID(), LAST_INSERT_ID(*expr*)

  With no argument, LAST_INSERT_ID() returns a BIGINT UNSIGNED (64-bit) value representing the first automatically generated value successfully inserted for an AUTO_INCREMENT column as a result of the most recently executed INSERT statement. The value of LAST_INSERT_ID() remains unchanged if no rows are successfully inserted.

  With an argument, LAST_INSERT_ID() returns an unsigned integer.

  For example, after inserting a row that generates an AUTO_INCREMENT value, you can get the value like this:

  ```
  mysql> SELECT LAST_INSERT_ID();
          -> 195
  ```

  The currently executing statement does not affect the value of LAST_INSERT_ID(). Suppose that you generate an AUTO_INCREMENT value with one statement, and then refer to LAST_INSERT_ID() in a multiple-row INSERT statement that inserts rows into a table with its own AUTO_INCREMENT column. The value of LAST_INSERT_ID() will remain stable in the second statement; its value for the second and later rows is not affected by the earlier row insertions. (However, if you mix references to LAST_INSERT_ID() and LAST_INSERT_ID(*expr*), the effect is undefined.)

  If the previous statement returned an error, the value of LAST_INSERT_ID() is undefined. For transactional tables, if the statement is rolled back due to an error, the value of LAST_INSERT_ID() is left undefined. For manual ROLLBACK, the value of LAST_INSERT_ID() is not restored to that before the transaction; it remains as it was at the point of the ROLLBACK.

  Prior to MySQL 5.7.3, this function was not replicated correctly if replication filtering rules were in use. (Bug #17234370, Bug #69861)

  Within the body of a stored routine (procedure or function) or a trigger, the value of LAST_INSERT_ID() changes the same way as for statements executed outside the body of these kinds of objects. The effect of a stored routine or trigger upon the value of LAST_INSERT_ID() that is seen by following statements depends on the kind of routine:

  - If a stored procedure executes statements that change the value of LAST_INSERT_ID(), the changed value is seen by statements that follow the procedure call.

  - For stored functions and triggers that change the value, the value is restored when the function or trigger ends, so following statements will not see a changed value.

  The ID that was generated is maintained in the server on a *per-connection basis*. This means that the value returned by the function to a given client is the first AUTO_INCREMENT value generated for most recent statement affecting an AUTO_INCREMENT column *by that client*. This value cannot be affected by other clients, even if they generate AUTO_INCREMENT values of their own. This behavior ensures that each client can retrieve its own ID without concern for the activity of other clients, and without the need for locks or transactions.

  The value of LAST_INSERT_ID() is not changed if you set the AUTO_INCREMENT column of a row to a non-“magic” value (that is, a value that is not NULL and not 0).

⚠ **Important**

If you insert multiple rows using a single INSERT statement, LAST_INSERT_ID() returns the value generated for the *first* inserted row *only*. The reason for this is to make it possible to reproduce easily the same INSERT statement against some other server.

For example:

```
mysql> USE test;
Database changed
mysql> CREATE TABLE t (
    ->   id INT AUTO_INCREMENT NOT NULL PRIMARY KEY,
    ->   name VARCHAR(10) NOT NULL
    -> );
Query OK, 0 rows affected (0.09 sec)

mysql> INSERT INTO t VALUES (NULL, 'Bob');
Query OK, 1 row affected (0.01 sec)

mysql> SELECT * FROM t;
+----+------+
| id | name |
+----+------+
|  1 | Bob  |
+----+------+
1 row in set (0.01 sec)

mysql> SELECT LAST_INSERT_ID();
+------------------+
| LAST_INSERT_ID() |
+------------------+
|                1 |
+------------------+
1 row in set (0.00 sec)

mysql> INSERT INTO t VALUES
    -> (NULL, 'Mary'), (NULL, 'Jane'), (NULL, 'Lisa');
Query OK, 3 rows affected (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM t;
+----+------+
| id | name |
+----+------+
|  1 | Bob  |
|  2 | Mary |
|  3 | Jane |
|  4 | Lisa |
+----+------+
4 rows in set (0.01 sec)

mysql> SELECT LAST_INSERT_ID();
+------------------+
| LAST_INSERT_ID() |
+------------------+
|                2 |
+------------------+
1 row in set (0.00 sec)
```

Although the second INSERT statement inserted three new rows into t, the ID generated for the first of these rows was 2, and it is this value that is returned by LAST_INSERT_ID() for the following SELECT statement.

If you use INSERT IGNORE and the row is ignored, the AUTO_INCREMENT counter is not incremented and LAST_INSERT_ID() returns 0, which reflects that no row was inserted.

If *expr* is given as an argument to LAST_INSERT_ID(), the value of the argument is returned by the function and is remembered as the next value to be returned by LAST_INSERT_ID(). This can be used to simulate sequences:

1. Create a table to hold the sequence counter and initialize it:

   ```
   mysql> CREATE TABLE sequence (id INT NOT NULL);
   mysql> INSERT INTO sequence VALUES (0);
   ```

2. Use the table to generate sequence numbers like this:

   ```
   mysql> UPDATE sequence SET id=LAST_INSERT_ID(id+1);
   mysql> SELECT LAST_INSERT_ID();
   ```

   The UPDATE statement increments the sequence counter and causes the next call to LAST_INSERT_ID() to return the updated value. The SELECT statement retrieves that value. The mysql_insert_id() C API function can also be used to get the value. See Section 21.8.7.38, “mysql_insert_id()”.

You can generate sequences without calling LAST_INSERT_ID(), but the utility of using the function this way is that the ID value is maintained in the server as the last automatically generated value. It is multi-user safe because multiple clients can issue the UPDATE statement and get their own sequence value with the SELECT statement (or mysql_insert_id()), without affecting or being affected by other clients that generate their own sequence values.

Note that mysql_insert_id() is only updated after INSERT and UPDATE statements, so you cannot use the C API function to retrieve the value for LAST_INSERT_ID(*expr*) after executing other SQL statements like SELECT or SET.

- ROW_COUNT()

  In MySQL 5.7, ROW_COUNT() returns a value as follows:

  - DDL statements: 0. This applies to statements such as CREATE TABLE or DROP TABLE.
  - DML statements other than SELECT: The number of affected rows. This applies to statements such as UPDATE, INSERT, or DELETE (as before), but now also to statements such as ALTER TABLE and LOAD DATA INFILE.
  - SELECT: -1 if the statement returns a result set, or the number of rows “affected” if it does not. For example, for SELECT * FROM t1, ROW_COUNT() returns -1. For SELECT * FROM t1 INTO OUTFILE '*file_name*', ROW_COUNT() returns the number of rows written to the file.
  - SIGNAL statements: 0.

  For UPDATE statements, the affected-rows value by default is the number of rows actually changed. If you specify the CLIENT_FOUND_ROWS flag to mysql_real_connect() when connecting to mysqld, the affected-rows value is the number of rows “found”; that is, matched by the WHERE clause.

  For REPLACE statements, the affected-rows value is 2 if the new row replaced an old row, because in this case, one row was inserted after the duplicate was deleted.

For INSERT ... ON DUPLICATE KEY UPDATE statements, the affected-rows value per row is 1 if the row is inserted as a new row, 2 if an existing row is updated, and 0 if an existing row is set to its current values. If you specify the CLIENT_FOUND_ROWS flag, the affected-rows value is 1 (not 0) if an existing row is set to its current values.

The ROW_COUNT() value is similar to the value from the mysql_affected_rows() C API function and the row count that the mysql client displays following statement execution.

```
mysql> INSERT INTO t VALUES(1),(2),(3);
Query OK, 3 rows affected (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> SELECT ROW_COUNT();
+-------------+
| ROW_COUNT() |
+-------------+
|           3 |
+-------------+
1 row in set (0.00 sec)

mysql> DELETE FROM t WHERE i IN(1,2);
Query OK, 2 rows affected (0.00 sec)

mysql> SELECT ROW_COUNT();
+-------------+
| ROW_COUNT() |
+-------------+
|           2 |
+-------------+
1 row in set (0.00 sec)
```

Important

ROW_COUNT() is not replicated reliably using statement-based replication. This function is automatically replicated using row-based replication.

- SCHEMA()

  This function is a synonym for DATABASE().

- SESSION_USER()

  SESSION_USER() is a synonym for USER().

- SYSTEM_USER()

  SYSTEM_USER() is a synonym for USER().

- USER()

  Returns the current MySQL user name and host name as a string in the utf8 character set.

```
mysql> SELECT USER();
        -> 'davida@localhost'
```

  The value indicates the user name you specified when connecting to the server, and the client host from which you connected. The value can be different from that of CURRENT_USER().

  You can extract only the user name part like this:

```
mysql> SELECT SUBSTRING_INDEX(USER(),'@',1);
        -> 'davida'
```

- `VERSION()`

  Returns a string that indicates the MySQL server version. The string uses the `utf8` character set. The value might have a suffix in addition to the version number. See the description of the `version` system variable in Section 5.1.4, “Server System Variables”.

  This function is unsafe for statement-based replication. In MySQL 5.7, a warning is logged if you use this function when `binlog_format` is set to `STATEMENT`. (Bug #47995)

  ```
  mysql> SELECT VERSION();
          -> '5.7.5-standard'
  ```

12.15 Functions Used with Global Transaction IDs

The functions described in this section are used with GTID-based replication. It is important to keep in mind that all of these functions take string representations of GTID sets as arguments—as such, the GTID sets must always be quoted when used with them.

The union of two GTID sets is simply their representations as strings, joined together with an interposed comma. In other words, you can define a very simple function for obtaining the union of two GTID sets, similar to that created here:

```
CREATE FUNCTION GTID_UNION(g1 TEXT, g2 TEXT)
    RETURNS TEXT DETERMINISTIC
    RETURN CONCAT(g1,',',g2);
```

For more information about GTIDs and how these GTID functions are used in practice, see Section 16.1.3, “Replication with Global Transaction Identifiers”.

Table 12.19 GTID Functions

Name	Description
`GTID_SUBSET()`	Return true if all GTIDs in subset are also in set; otherwise false.
`GTID_SUBTRACT()`	Return all GTIDs in set that are not in subset.
`WAIT_UNTIL_SQL_THREAD_AFTER_GTIDS()`	Wait until the slave SQL thread has executed all the given GTIDs. Returns: the number of events that were executed (or NULL, if GTID mode is not enabled).

- `GTID_SUBSET(`*`subset`*`,`*`set`*`)`

  Given two sets of global transaction IDs *`subset`* and *`set`*, returns true if all GTIDs in *`subset`* are also in *`set`*. Returns false otherwise.

  The GTID sets used with this function are represented as strings, as shown in the following examples:

  ```
  mysql> SELECT GTID_SUBSET('3E11FA47-71CA-11E1-9E33-C80AA9429562:23',
      ->     '3E11FA47-71CA-11E1-9E33-C80AA9429562:21-57')\G
  ```

```
*************************** 1. row ***************************
GTID_SUBSET('3E11FA47-71CA-11E1-9E33-C80AA9429562:23',
    '3E11FA47-71CA-11E1-9E33-C80AA9429562:21-57'): 1
1 row in set (0.00 sec)

mysql> SELECT GTID_SUBSET('3E11FA47-71CA-11E1-9E33-C80AA9429562:23-25',
    ->     '3E11FA47-71CA-11E1-9E33-C80AA9429562:21-57')\G
*************************** 1. row ***************************
GTID_SUBSET('3E11FA47-71CA-11E1-9E33-C80AA9429562:23-25',
    '3E11FA47-71CA-11E1-9E33-C80AA9429562:21-57'): 1
1 row in set (0.00 sec)

mysql> SELECT GTID_SUBSET('3E11FA47-71CA-11E1-9E33-C80AA9429562:20-25',
    ->     '3E11FA47-71CA-11E1-9E33-C80AA9429562:21-57')\G
*************************** 1. row ***************************
GTID_SUBSET('3E11FA47-71CA-11E1-9E33-C80AA9429562:20-25',
    '3E11FA47-71CA-11E1-9E33-C80AA9429562:21-57'): 0
1 row in set (0.00 sec)
```

- `GTID_SUBTRACT(`*`set`*`,`*`subset`*`)`

  Given two sets of global transaction IDs *`subset`* and *`set`*, returns only those GTIDs from *`set`* that are not in *`subset`*.

  All GTID sets used with this function are represented as strings and must be quoted, as shown in these examples:

```
mysql> SELECT GTID_SUBTRACT('3E11FA47-71CA-11E1-9E33-C80AA9429562:21-57',
    ->     '3E11FA47-71CA-11E1-9E33-C80AA9429562:21')\G
*************************** 1. row ***************************
GTID_SUBTRACT('3E11FA47-71CA-11E1-9E33-C80AA9429562:21-57',
    '3E11FA47-71CA-11E1-9E33-C80AA9429562:21'): 3e11fa47-71ca-11e1-9e33-c80aa9429562:22-57
1 row in set (0.00 sec)

mysql> SELECT GTID_SUBTRACT('3E11FA47-71CA-11E1-9E33-C80AA9429562:21-57',
    ->     '3E11FA47-71CA-11E1-9E33-C80AA9429562:20-25')\G
*************************** 1. row ***************************
GTID_SUBTRACT('3E11FA47-71CA-11E1-9E33-C80AA9429562:21-57',
    '3E11FA47-71CA-11E1-9E33-C80AA9429562:20-25'): 3e11fa47-71ca-11e1-9e33-c80aa9429562:26-57
1 row in set (0.00 sec)

mysql> SELECT GTID_SUBTRACT('3E11FA47-71CA-11E1-9E33-C80AA9429562:21-57',
    ->     '3E11FA47-71CA-11E1-9E33-C80AA9429562:23-24')\G
*************************** 1. row ***************************
GTID_SUBTRACT('3E11FA47-71CA-11E1-9E33-C80AA9429562:21-57',
    '3E11FA47-71CA-11E1-9E33-C80AA9429562:23-24'): 3e11fa47-71ca-11e1-9e33-c80aa9429562:21-22:25-57
1 row in set (0.01 sec)
```

- `WAIT_UNTIL_SQL_THREAD_AFTER_GTIDS(`*`gtid_set`*`[, `*`timeout`*`])`

  Wait until the slave SQL thread has executed all of the transactions whose global transaction identifiers are contained in *`gtid_set`* (see Section 16.1.3.1, “GTID Concepts”, for a definition of “GTID sets”), or until *`timeout`* seconds have elapsed, whichever occurs first. *`timeout`* is optional; the default timeout is 0 seconds, in which case the master simply waits until all of the transactions in the GTID set have been executed.

  For more information, see Section 16.1.3, “Replication with Global Transaction Identifiers”.

  GTID sets used with this function are represented as strings and so must be quoted as shown in the following example:

```
mysql> SELECT WAIT_UNTIL_SQL_THREAD_AFTER_GTIDS('3E11FA47-71CA-11E1-9E33-C80AA9429562:1-5');
```

```
        -> 5
```

The return value is the number of transactional events that were executed. If GTID-based replication is not active (that is, if the value of the gtid_mode variable is OFF), then this value is undefined and thus WAIT_UNTIL_SQL_THREAD_AFTER_GTIDS() returns NULL.

12.16 Miscellaneous Functions

Table 12.20 Miscellaneous Functions

Name	Description
DEFAULT()	Return the default value for a table column
GET_LOCK()	Get a named lock
INET_ATON()	Return the numeric value of an IP address
INET_NTOA()	Return the IP address from a numeric value
INET6_ATON()	Return the numeric value of an IPv6 address
INET6_NTOA()	Return the IPv6 address from a numeric value
IS_FREE_LOCK()	Checks whether the named lock is free
IS_IPV4_COMPAT()	Return true if argument is an IPv4-compatible address
IS_IPV4_MAPPED()	Return true if argument is an IPv4-mapped address
IS_IPV4()	Return true if argument is an IPv4 address
IS_IPV6()	Return true if argument is an IPv6 address
IS_USED_LOCK()	Checks whether the named lock is in use. Return connection identifier if true.
MASTER_POS_WAIT()	Block until the slave has read and applied all updates up to the specified position
NAME_CONST()	Causes the column to have the given name
RAND()	Return a random floating-point value
RELEASE_LOCK()	Releases the named lock
SLEEP()	Sleep for a number of seconds
UUID_SHORT()	Return an integer-valued universal identifier
UUID()	Return a Universal Unique Identifier (UUID)
VALUES()	Defines the values to be used during an INSERT

- DEFAULT(*col_name*)

  Returns the default value for a table column. An error results if the column has no default value.

  ```
  mysql> UPDATE t SET i = DEFAULT(i)+1 WHERE id < 100;
  ```

- FORMAT(*X*,*D*)

  Formats the number *X* to a format like '#,###,###.##', rounded to *D* decimal places, and returns the result as a string. For details, see Section 12.5, “String Functions”.

- GET_LOCK(*str*,*timeout*)

Tries to obtain a lock with a name given by the string *str*, using a timeout of *timeout* seconds. Returns 1 if the lock was obtained successfully, 0 if the attempt timed out (for example, because another client has previously locked the name), or NULL if an error occurred (such as running out of memory or the thread was killed with mysqladmin kill). If you have a lock obtained with GET_LOCK(), it is released when you execute RELEASE_LOCK(), execute a new GET_LOCK(), or your connection terminates (either normally or abnormally). Locks obtained with GET_LOCK() do not interact with transactions. That is, committing a transaction does not release any such locks obtained during the transaction.

This function can be used to implement application locks or to simulate record locks. Names are locked on a server-wide basis. If a name has been locked by one client, GET_LOCK() blocks any request by another client for a lock with the same name. This enables clients that agree on a given lock name to use the name to perform cooperative advisory locking. But be aware that it also enables a client that is not among the set of cooperating clients to lock a name, either inadvertently or deliberately, and thus prevent any of the cooperating clients from locking that name. One way to reduce the likelihood of this is to use lock names that are database-specific or application-specific. For example, use lock names of the form *db_name.str* or *app_name.str*.

```
mysql> SELECT GET_LOCK('lock1',10);
        -> 1
mysql> SELECT IS_FREE_LOCK('lock2');
        -> 1
mysql> SELECT GET_LOCK('lock2',10);
        -> 1
mysql> SELECT RELEASE_LOCK('lock2');
        -> 1
mysql> SELECT RELEASE_LOCK('lock1');
        -> NULL
```

The second RELEASE_LOCK() call returns NULL because the lock 'lock1' was automatically released by the second GET_LOCK() call.

If multiple clients are waiting for a lock, the order in which they will acquire it is undefined and depends on factors such as the thread library in use. In particular, applications should not assume that clients will acquire the lock in the same order that they issued the lock requests.

This function is unsafe for statement-based replication. In MySQL 5.7, a warning is logged if you use this function when binlog_format is set to STATEMENT. (Bug #47995)

- INET_ATON(*expr*)

Given the dotted-quad representation of an IPv4 network address as a string, returns an integer that represents the numeric value of the address in network byte order (big endian). INET_ATON() returns NULL if it does not understand its argument.

```
mysql> SELECT INET_ATON('10.0.5.9');
        -> 167773449
```

For this example, the return value is calculated as $10 \times 256^3 + 0 \times 256^2 + 5 \times 256 + 9$.

INET_ATON() may or may not return a non-NULL result for short-form IP addresses (such as '127.1' as a representation of '127.0.0.1'). Because of this, INET_ATON()a should not be used for such addresses.

Note

To store values generated by INET_ATON(), use an INT UNSIGNED column rather than INT, which is signed. If you use a signed column, values corresponding to IP addresses for which the first octet is greater than 127 cannot be stored correctly. See Section 11.2.6, “Out-of-Range and Overflow Handling”.

- INET_NTOA(*expr*)

  Given a numeric IPv4 network address in network byte order, returns the dotted-quad string representation of the address as a nonbinary string in the connection character set. INET_NTOA() returns NULL if it does not understand its argument.

  ```
  mysql> SELECT INET_NTOA(167773449);
          -> '10.0.5.9'
  ```

- INET6_ATON(*expr*)

  Given an IPv6 or IPv4 network address as a string, returns a binary string that represents the numeric value of the address in network byte order (big endian). Because numeric-format IPv6 addresses require more bytes than the largest integer type, the representation returned by this function has the VARBINARY data type: VARBINARY(16) for IPv6 addresses and VARBINARY(4) for IPv4 addresses. If the argument is not a valid address, INET6_ATON() returns NULL.

  The following examples use HEX() to display the INET6_ATON() result in printable form:

  ```
  mysql> SELECT HEX(INET6_ATON('fdfe::5a55:caff:fefa:9089'));
          -> 'FDFE0000000000005A55CAFFFEFA9089'
  mysql> SELECT HEX(INET6_ATON('10.0.5.9'));
          -> '0A000509'
  ```

  INET6_ATON() observes several constraints on valid arguments. These are given in the following list along with examples.

  - A trailing zone ID is not permitted, as in fe80::3%1 or fe80::3%eth0.

  - A trailing network mask is not permitted, as in 2001:45f:3:ba::/64 or 192.168.1.0/24.

  - For values representing IPv4 addresses, only classless addresses are supported. Classful addresses such as 192.168.1 are rejected. A trailing port number is not permitted, as in 192.168.1.2:8080. Hexadecimal numbers in address components are not permitted, as in 192.0xa0.1.2. Octal numbers are not supported: 192.168.010.1 is treated as 192.168.10.1, not 192.168.8.1. These IPv4 constraints also apply to IPv6 addresses that have IPv4 address parts, such as IPv4-compatible or IPv4-mapped addresses.

  To convert an IPv4 address *expr* represented in numeric form as an INT value to an IPv6 address represented in numeric form as a VARBINARY value, use this expression:

  ```
  INET6_ATON(INET_NTOA(expr))
  ```

  For example:

  ```
  mysql> SELECT HEX(INET6_ATON(INET_NTOA(167773449)));
          -> '0A000509'
  ```

- `INET6_NTOA(expr)`

  Given an IPv6 or IPv4 network address represented in numeric form as a binary string, returns the string representation of the address as a nonbinary string in the connection character set. If the argument is not a valid address, `INET6_NTOA()` returns `NULL`.

  `INET6_NTOA()` has these properties:

  - It does not use operating system functions to perform conversions, thus the output string is platform independent.
  - The return string has a maximum length of 39 (4 x 8 + 7). Given this statement:

    ```
    CREATE TABLE t AS SELECT INET6_NTOA(expr) AS c1;
    ```

    The resulting table would have this definition:

    ```
    CREATE TABLE t (c1 VARCHAR(39) CHARACTER SET utf8 DEFAULT NULL);
    ```

  - The return string uses lowercase letters for IPv6 addresses.

  ```
  mysql> SELECT INET6_NTOA(INET6_ATON('fdfe::5a55:caff:fefa:9089'));
          -> 'fdfe::5a55:caff:fefa:9089'
  mysql> SELECT INET6_NTOA(INET6_ATON('10.0.5.9'));
          -> '10.0.5.9'

  mysql> SELECT INET6_NTOA(UNHEX('FDFE0000000000005A55CAFFFEFA9089'));
          -> 'fdfe::5a55:caff:fefa:9089'
  mysql> SELECT INET6_NTOA(UNHEX('0A000509'));
          -> '10.0.5.9'
  ```

- `IS_FREE_LOCK(str)`

  Checks whether the lock named *`str`* is free to use (that is, not locked). Returns `1` if the lock is free (no one is using the lock), `0` if the lock is in use, and `NULL` if an error occurs (such as an incorrect argument).

  This function is unsafe for statement-based replication. In MySQL 5.7, a warning is logged if you use this function when `binlog_format` is set to `STATEMENT`. (Bug #47995)

- `IS_IPV4(expr)`

  Returns 1 if the argument is a valid IPv4 address specified as a string, 0 otherwise.

  ```
  mysql> SELECT IS_IPV4('10.0.5.9'), IS_IPV4('10.0.5.256');
          -> 1, 0
  ```

  For a given argument, if `IS_IPV4()` returns 1, `INET_ATON()` (and `INET6_ATON()`) will return non-`NULL`. The converse statement is not true: In some cases, `INET_ATON()` returns non-`NULL` when `IS_IPV4()` returns 0.

  As implied by the preceding remarks, `IS_IPV4()` is more strict than `INET_ATON()` about what constitutes a valid IPv4 address, so it may be useful for applications that need to perform strong checks against invalid values. Alternatively, use `INET6_ATON()` to convert IPv4 addresses to internal form and check for a `NULL` result (which indicates an invalid address). `INET6_ATON()` is equally strong as `IS_IPV4()` about checking IPv4 addresses.

- `IS_IPV4_COMPAT(expr)`

  This function takes an IPv6 address represented in numeric form as a binary string, as returned by `INET6_ATON()`. It returns 1 if the argument is a valid IPv4-compatible IPv6 address, 0 otherwise. IPv4-compatible addresses have the form `::ipv4_address`.

  ```
  mysql> SELECT IS_IPV4_COMPAT(INET6_ATON('::10.0.5.9'));
          -> 1
  mysql> SELECT IS_IPV4_COMPAT(INET6_ATON('::ffff:10.0.5.9'));
          -> 0
  ```

  The IPv4 part of an IPv4-compatible address can also be represented using hexadecimal notation. For example, `192.168.0.1` has this raw hexadecimal value:

  ```
  mysql> SELECT HEX(INET6_ATON('192.168.0.1'));
          -> 'C0A80001'
  ```

  Expressed in IPv4-compatible form, `::192.168.0.1` is equivalent to `::c0a8:0001` or (without leading zeros) `::c0a8:1`

  ```
  mysql> SELECT
      ->   IS_IPV4_COMPAT(INET6_ATON('::192.168.0.1')),
      ->   IS_IPV4_COMPAT(INET6_ATON('::c0a8:0001')),
      ->   IS_IPV4_COMPAT(INET6_ATON('::c0a8:1'));
          -> 1, 1, 1
  ```

- `IS_IPV4_MAPPED(expr)`

  This function takes an IPv6 address represented in numeric form as a binary string, as returned by `INET6_ATON()`. It returns 1 if the argument is a valid IPv4-mapped IPv6 address, 0 otherwise. IPv4-mapped addresses have the form `::ffff:ipv4_address`.

  ```
  mysql> SELECT IS_IPV4_MAPPED(INET6_ATON('::10.0.5.9'));
          -> 0
  mysql> SELECT IS_IPV4_MAPPED(INET6_ATON('::ffff:10.0.5.9'));
          -> 1
  ```

  As with `IS_IPV4_COMPAT()` the IPv4 part of an IPv4-mapped address can also be represented using hexadecimal notation:

  ```
  mysql> SELECT
      ->   IS_IPV4_MAPPED(INET6_ATON('::ffff:192.168.0.1')),
      ->   IS_IPV4_MAPPED(INET6_ATON('::ffff:c0a8:0001')),
      ->   IS_IPV4_MAPPED(INET6_ATON('::ffff:c0a8:1'));
          -> 1, 1, 1
  ```

- `IS_IPV6(expr)`

  Returns 1 if the argument is a valid IPv6 address specified as a string, 0 otherwise. This function does not consider IPv4 addresses to be valid IPv6 addresses.

  ```
  mysql> SELECT IS_IPV6('10.0.5.9'), IS_IPV6('::1');
          -> 0, 1
  ```

  For a given argument, if `IS_IPV6()` returns 1, `INET6_ATON()` will return non-`NULL`.

- `IS_USED_LOCK(str)`

  Checks whether the lock named *`str`* is in use (that is, locked). If so, it returns the connection identifier of the client that holds the lock. Otherwise, it returns `NULL`.

  This function is unsafe for statement-based replication. In MySQL 5.7, a warning is logged if you use this function when `binlog_format` is set to `STATEMENT`. (Bug #47995)

- `MASTER_POS_WAIT(log_name,log_pos[,timeout])`

  This function is useful for control of master/slave synchronization. It blocks until the slave has read and applied all updates up to the specified position in the master log. The return value is the number of log events the slave had to wait for to advance to the specified position. The function returns `NULL` if the slave SQL thread is not started, the slave's master information is not initialized, the arguments are incorrect, or an error occurs. It returns `-1` if the timeout has been exceeded. If the slave SQL thread stops while `MASTER_POS_WAIT()` is waiting, the function returns `NULL`. If the slave is past the specified position, the function returns immediately.

  If a *`timeout`* value is specified, `MASTER_POS_WAIT()` stops waiting when *`timeout`* seconds have elapsed. *`timeout`* must be greater than 0; a zero or negative *`timeout`* means no timeout.

  This function is unsafe for statement-based replication. In MySQL 5.7, a warning is logged if you use this function when `binlog_format` is set to `STATEMENT`. (Bug #47995)

- `NAME_CONST(name,value)`

  Returns the given value. When used to produce a result set column, `NAME_CONST()` causes the column to have the given name. The arguments should be constants.

  ```
  mysql> SELECT NAME_CONST('myname', 14);
  +--------+
  | myname |
  +--------+
  |     14 |
  +--------+
  ```

  This function is for internal use only. The server uses it when writing statements from stored programs that contain references to local program variables, as described in Section 18.7, “Binary Logging of Stored Programs”, You might see this function in the output from `mysqlbinlog`.

- `RELEASE_LOCK(str)`

  Releases the lock named by the string *`str`* that was obtained with `GET_LOCK()`. Returns `1` if the lock was released, `0` if the lock was not established by this thread (in which case the lock is not released), and `NULL` if the named lock did not exist. The lock does not exist if it was never obtained by a call to `GET_LOCK()` or if it has previously been released.

  The `DO` statement is convenient to use with `RELEASE_LOCK()`. See Section 13.2.3, “`DO` Syntax”.

  This function is unsafe for statement-based replication. In MySQL 5.7, a warning is logged if you use this function when `binlog_format` is set to `STATEMENT`. (Bug #47995)

- `SLEEP(duration)`

  Sleeps (pauses) for the number of seconds given by the *`duration`* argument, then returns 0. If `SLEEP()` is interrupted, it returns 1. The duration may have a fractional part. If the argument is `NULL` or negative, `SLEEP()` produces a warning, or an error in strict SQL mode.

This function is unsafe for statement-based replication. In MySQL 5.7, a warning is logged if you use this function when `binlog_format` is set to `STATEMENT`. (Bug #47995)

- `UUID()`

  Returns a Universal Unique Identifier (UUID) generated according to “DCE 1.1: Remote Procedure Call” (Appendix A) CAE (Common Applications Environment) Specifications published by The Open Group in October 1997 (Document Number C706, http://www.opengroup.org/public/pubs/catalog/c706.htm).

  A UUID is designed as a number that is globally unique in space and time. Two calls to `UUID()` are expected to generate two different values, even if these calls are performed on two separate computers that are not connected to each other.

  A UUID is a 128-bit number represented by a `utf8` string of five hexadecimal numbers in `aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee` format:

  - The first three numbers are generated from a timestamp.

  - The fourth number preserves temporal uniqueness in case the timestamp value loses monotonicity (for example, due to daylight saving time).

  - The fifth number is an IEEE 802 node number that provides spatial uniqueness. A random number is substituted if the latter is not available (for example, because the host computer has no Ethernet card, or we do not know how to find the hardware address of an interface on your operating system). In this case, spatial uniqueness cannot be guaranteed. Nevertheless, a collision should have *very* low probability.

    Currently, the MAC address of an interface is taken into account only on FreeBSD and Linux. On other operating systems, MySQL uses a randomly generated 48-bit number.

  ```
  mysql> SELECT UUID();
          -> '6ccd780c-baba-1026-9564-0040f4311e29'
  ```

  Warning

  Although `UUID()` values are intended to be unique, they are not necessarily unguessable or unpredictable. If unpredictability is required, UUID values should be generated some other way.

  Note

  `UUID()` does not work with statement-based replication.

- `UUID_SHORT()`

  Returns a “short” universal identifier as a 64-bit unsigned integer (rather than a string-form 128-bit identifier as returned by the `UUID()` function).

  The value of `UUID_SHORT()` is guaranteed to be unique if the following conditions hold:

  - The `server_id` of the current host is unique among your set of master and slave servers

  - `server_id` is between 0 and 255

  - You do not set back your system time for your server between `mysqld` restarts

- You do not invoke UUID_SHORT() on average more than 16 million times per second between mysqld restarts

The UUID_SHORT() return value is constructed this way:

```
  (server_id & 255) << 56
+ (server_startup_time_in_seconds << 24)
+ incremented_variable++;
```

```
mysql> SELECT UUID_SHORT();
        -> 92395783831158784
```

Note that UUID_SHORT() does not work with statement-based replication.

- VALUES(*col_name*)

  In an INSERT ... ON DUPLICATE KEY UPDATE statement, you can use the VALUES(*col_name*) function in the UPDATE clause to refer to column values from the INSERT portion of the statement. In other words, VALUES(*col_name*) in the UPDATE clause refers to the value of *col_name* that would be inserted, had no duplicate-key conflict occurred. This function is especially useful in multiple-row inserts. The VALUES() function is meaningful only in the ON DUPLICATE KEY UPDATE clause of INSERT statements and returns NULL otherwise. See Section 13.2.5.3, "INSERT ... ON DUPLICATE KEY UPDATE Syntax".

```
mysql> INSERT INTO table (a,b,c) VALUES (1,2,3),(4,5,6)
    -> ON DUPLICATE KEY UPDATE c=VALUES(a)+VALUES(b);
```

12.17 Functions and Modifiers for Use with GROUP BY Clauses

12.17.1 GROUP BY (Aggregate) Functions

Table 12.21 Aggregate (GROUP BY) Functions

Name	Description
AVG()	Return the average value of the argument
BIT_AND()	Return bitwise and
BIT_OR()	Return bitwise or
BIT_XOR()	Return bitwise xor
COUNT(DISTINCT)	Return the count of a number of different values
COUNT()	Return a count of the number of rows returned
GROUP_CONCAT()	Return a concatenated string
MAX()	Return the maximum value
MIN()	Return the minimum value
STD()	Return the population standard deviation
STDDEV_POP()	Return the population standard deviation
STDDEV_SAMP()	Return the sample standard deviation
STDDEV()	Return the population standard deviation
SUM()	Return the sum

Name	Description
VAR_POP()	Return the population standard variance
VAR_SAMP()	Return the sample variance
VARIANCE()	Return the population standard variance

This section describes group (aggregate) functions that operate on sets of values. Unless otherwise stated, group functions ignore NULL values.

If you use a group function in a statement containing no GROUP BY clause, it is equivalent to grouping on all rows. For more information, see Section 12.17.3, “MySQL Extensions to GROUP BY”.

For numeric arguments, the variance and standard deviation functions return a DOUBLE value. The SUM() and AVG() functions return a DECIMAL value for exact-value arguments (integer or DECIMAL), and a DOUBLE value for approximate-value arguments (FLOAT or DOUBLE).

The SUM() and AVG() aggregate functions do not work with temporal values. (They convert the values to numbers, losing everything after the first nonnumeric character.) To work around this problem, convert to numeric units, perform the aggregate operation, and convert back to a temporal value. Examples:

```
SELECT SEC_TO_TIME(SUM(TIME_TO_SEC(time_col))) FROM tbl_name;
SELECT FROM_DAYS(SUM(TO_DAYS(date_col))) FROM tbl_name;
```

Functions such as SUM() or AVG() that expect a numeric argument cast the argument to a number if necessary. For SET or ENUM values, the cast operation causes the underlying numeric value to be used.

- AVG([DISTINCT] *expr*)

  Returns the average value of *expr*. The DISTINCT option can be used to return the average of the distinct values of *expr*.

  AVG() returns NULL if there were no matching rows.

  ```
  mysql> SELECT student_name, AVG(test_score)
      ->        FROM student
      ->        GROUP BY student_name;
  ```

- BIT_AND(*expr*)

  Returns the bitwise AND of all bits in *expr*. The calculation is performed with 64-bit (BIGINT) precision.

  This function returns 18446744073709551615 if there were no matching rows. (This is the value of an unsigned BIGINT value with all bits set to 1.)

- BIT_OR(*expr*)

  Returns the bitwise OR of all bits in *expr*. The calculation is performed with 64-bit (BIGINT) precision.

  This function returns 0 if there were no matching rows.

- BIT_XOR(*expr*)

  Returns the bitwise XOR of all bits in *expr*. The calculation is performed with 64-bit (BIGINT) precision.

  This function returns 0 if there were no matching rows.

- COUNT(*expr*)

Returns a count of the number of non-NULL values of *expr* in the rows retrieved by a SELECT statement. The result is a BIGINT value.

COUNT() returns 0 if there were no matching rows.

```
mysql> SELECT student.student_name,COUNT(*)
    ->        FROM student,course
    ->        WHERE student.student_id=course.student_id
    ->        GROUP BY student_name;
```

COUNT(*) is somewhat different in that it returns a count of the number of rows retrieved, whether or not they contain NULL values.

COUNT(*) is optimized to return very quickly if the SELECT retrieves from one table, no other columns are retrieved, and there is no WHERE clause. For example:

```
mysql> SELECT COUNT(*) FROM student;
```

This optimization applies only to MyISAM tables only, because an exact row count is stored for this storage engine and can be accessed very quickly. For transactional storage engines such as InnoDB, storing an exact row count is more problematic because multiple transactions may be occurring, each of which may affect the count.

- COUNT(DISTINCT *expr*,[*expr*...])

  Returns a count of the number of rows with different non-NULL *expr* values.

  COUNT(DISTINCT) returns 0 if there were no matching rows.

```
mysql> SELECT COUNT(DISTINCT results) FROM student;
```

  In MySQL, you can obtain the number of distinct expression combinations that do not contain NULL by giving a list of expressions. In standard SQL, you would have to do a concatenation of all expressions inside COUNT(DISTINCT ...).

- GROUP_CONCAT(*expr*)

  This function returns a string result with the concatenated non-NULL values from a group. It returns NULL if there are no non-NULL values. The full syntax is as follows:

```
GROUP_CONCAT([DISTINCT] expr [,expr ...]
             [ORDER BY {unsigned_integer | col_name | expr}
                 [ASC | DESC] [,col_name ...]]
             [SEPARATOR str_val])
```

```
mysql> SELECT student_name,
    ->     GROUP_CONCAT(test_score)
    ->     FROM student
    ->     GROUP BY student_name;
```

  Or:

```
mysql> SELECT student_name,
    ->     GROUP_CONCAT(DISTINCT test_score
    ->               ORDER BY test_score DESC SEPARATOR ' ')
    ->     FROM student
    ->     GROUP BY student_name;
```

In MySQL, you can get the concatenated values of expression combinations. To eliminate duplicate values, use the DISTINCT clause. To sort values in the result, use the ORDER BY clause. To sort in reverse order, add the DESC (descending) keyword to the name of the column you are sorting by in the ORDER BY clause. The default is ascending order; this may be specified explicitly using the ASC keyword. The default separator between values in a group is comma (“,”). To specify a separator explicitly, use SEPARATOR followed by the string literal value that should be inserted between group values. To eliminate the separator altogether, specify SEPARATOR ''.

The result is truncated to the maximum length that is given by the group_concat_max_len system variable, which has a default value of 1024. The value can be set higher, although the effective maximum length of the return value is constrained by the value of max_allowed_packet. The syntax to change the value of group_concat_max_len at runtime is as follows, where *val* is an unsigned integer:

```
SET [GLOBAL | SESSION] group_concat_max_len = val;
```

The return value is a nonbinary or binary string, depending on whether the arguments are nonbinary or binary strings. The result type is TEXT or BLOB unless group_concat_max_len is less than or equal to 512, in which case the result type is VARCHAR or VARBINARY.

See also CONCAT() and CONCAT_WS(): Section 12.5, “String Functions”.

- MAX([DISTINCT] *expr*)

  Returns the maximum value of *expr*. MAX() may take a string argument; in such cases, it returns the maximum string value. See Section 8.3.1, “How MySQL Uses Indexes”. The DISTINCT keyword can be used to find the maximum of the distinct values of *expr*, however, this produces the same result as omitting DISTINCT.

  MAX() returns NULL if there were no matching rows.

  ```
  mysql> SELECT student_name, MIN(test_score), MAX(test_score)
      ->        FROM student
      ->        GROUP BY student_name;
  ```

  For MAX(), MySQL currently compares ENUM and SET columns by their string value rather than by the string's relative position in the set. This differs from how ORDER BY compares them. This is expected to be rectified in a future MySQL release.

- MIN([DISTINCT] *expr*)

  Returns the minimum value of *expr*. MIN() may take a string argument; in such cases, it returns the minimum string value. See Section 8.3.1, “How MySQL Uses Indexes”. The DISTINCT keyword can be used to find the minimum of the distinct values of *expr*, however, this produces the same result as omitting DISTINCT.

  MIN() returns NULL if there were no matching rows.

  ```
  mysql> SELECT student_name, MIN(test_score), MAX(test_score)
      ->        FROM student
      ->        GROUP BY student_name;
  ```

For MIN(), MySQL currently compares ENUM and SET columns by their string value rather than by the string's relative position in the set. This differs from how ORDER BY compares them. This is expected to be rectified in a future MySQL release.

- STD(*expr*)

  Returns the population standard deviation of *expr*. This is an extension to standard SQL. The standard SQL function STDDEV_POP() can be used instead.

  This function returns NULL if there were no matching rows.

- STDDEV(*expr*)

  Returns the population standard deviation of *expr*. This function is provided for compatibility with Oracle. The standard SQL function STDDEV_POP() can be used instead.

  This function returns NULL if there were no matching rows.

- STDDEV_POP(*expr*)

  Returns the population standard deviation of *expr* (the square root of VAR_POP()). You can also use STD() or STDDEV(), which are equivalent but not standard SQL.

  STDDEV_POP() returns NULL if there were no matching rows.

- STDDEV_SAMP(*expr*)

  Returns the sample standard deviation of *expr* (the square root of VAR_SAMP().

  STDDEV_SAMP() returns NULL if there were no matching rows.

- SUM([DISTINCT] *expr*)

  Returns the sum of *expr*. If the return set has no rows, SUM() returns NULL. The DISTINCT keyword can be used to sum only the distinct values of *expr*.

  SUM() returns NULL if there were no matching rows.

- VAR_POP(*expr*)

  Returns the population standard variance of *expr*. It considers rows as the whole population, not as a sample, so it has the number of rows as the denominator. You can also use VARIANCE(), which is equivalent but is not standard SQL.

  VAR_POP() returns NULL if there were no matching rows.

- VAR_SAMP(*expr*)

  Returns the sample variance of *expr*. That is, the denominator is the number of rows minus one.

  VAR_SAMP() returns NULL if there were no matching rows.

- VARIANCE(*expr*)

  Returns the population standard variance of *expr*. This is an extension to standard SQL. The standard SQL function VAR_POP() can be used instead.

  VARIANCE() returns NULL if there were no matching rows.

12.17.2 GROUP BY Modifiers

The GROUP BY clause permits a WITH ROLLUP modifier that causes extra rows to be added to the summary output. These rows represent higher-level (or super-aggregate) summary operations. ROLLUP thus enables you to answer questions at multiple levels of analysis with a single query. It can be used, for example, to provide support for OLAP (Online Analytical Processing) operations.

Suppose that a table named sales has year, country, product, and profit columns for recording sales profitability:

```
CREATE TABLE sales
(
    year    INT NOT NULL,
    country VARCHAR(20) NOT NULL,
    product VARCHAR(32) NOT NULL,
    profit  INT
);
```

The table's contents can be summarized per year with a simple GROUP BY like this:

```
mysql> SELECT year, SUM(profit) FROM sales GROUP BY year;
+------+-------------+
| year | SUM(profit) |
+------+-------------+
| 2000 |        4525 |
| 2001 |        3010 |
+------+-------------+
```

This output shows the total profit for each year, but if you also want to determine the total profit summed over all years, you must add up the individual values yourself or run an additional query.

Or you can use ROLLUP, which provides both levels of analysis with a single query. Adding a WITH ROLLUP modifier to the GROUP BY clause causes the query to produce another row that shows the grand total over all year values:

```
mysql> SELECT year, SUM(profit) FROM sales GROUP BY year WITH ROLLUP;
+------+-------------+
| year | SUM(profit) |
+------+-------------+
| 2000 |        4525 |
| 2001 |        3010 |
| NULL |        7535 |
+------+-------------+
```

The grand total super-aggregate line is identified by the value NULL in the year column.

ROLLUP has a more complex effect when there are multiple GROUP BY columns. In this case, each time there is a “break” (change in value) in any but the last grouping column, the query produces an extra super-aggregate summary row.

For example, without ROLLUP, a summary on the sales table based on year, country, and product might look like this:

```
mysql> SELECT year, country, product, SUM(profit)
    -> FROM sales
    -> GROUP BY year, country, product;
+------+---------+------------+-------------+
| year | country | product    | SUM(profit) |
+------+---------+------------+-------------+
| 2000 | Finland | Computer   |        1500 |
| 2000 | Finland | Phone      |         100 |
| 2000 | India   | Calculator |         150 |
| 2000 | India   | Computer   |        1200 |
| 2000 | USA     | Calculator |          75 |
| 2000 | USA     | Computer   |        1500 |
| 2001 | Finland | Phone      |          10 |
| 2001 | USA     | Calculator |          50 |
| 2001 | USA     | Computer   |        2700 |
| 2001 | USA     | TV         |         250 |
+------+---------+------------+-------------+
```

The output indicates summary values only at the year/country/product level of analysis. When ROLLUP is added, the query produces several extra rows:

```
mysql> SELECT year, country, product, SUM(profit)
    -> FROM sales
    -> GROUP BY year, country, product WITH ROLLUP;
+------+---------+------------+-------------+
| year | country | product    | SUM(profit) |
+------+---------+------------+-------------+
| 2000 | Finland | Computer   |        1500 |
| 2000 | Finland | Phone      |         100 |
| 2000 | Finland | NULL       |        1600 |
| 2000 | India   | Calculator |         150 |
| 2000 | India   | Computer   |        1200 |
| 2000 | India   | NULL       |        1350 |
| 2000 | USA     | Calculator |          75 |
| 2000 | USA     | Computer   |        1500 |
| 2000 | USA     | NULL       |        1575 |
| 2000 | NULL    | NULL       |        4525 |
| 2001 | Finland | Phone      |          10 |
| 2001 | Finland | NULL       |          10 |
| 2001 | USA     | Calculator |          50 |
| 2001 | USA     | Computer   |        2700 |
| 2001 | USA     | TV         |         250 |
| 2001 | USA     | NULL       |        3000 |
| 2001 | NULL    | NULL       |        3010 |
| NULL | NULL    | NULL       |        7535 |
+------+---------+------------+-------------+
```

For this query, adding ROLLUP causes the output to include summary information at four levels of analysis, not just one. Here is how to interpret the ROLLUP output:

- Following each set of product rows for a given year and country, an extra summary row is produced showing the total for all products. These rows have the product column set to NULL.

- Following each set of rows for a given year, an extra summary row is produced showing the total for all countries and products. These rows have the country and products columns set to NULL.

- Finally, following all other rows, an extra summary row is produced showing the grand total for all years, countries, and products. This row has the year, country, and products columns set to NULL.

Other Considerations When using ROLLUP

The following items list some behaviors specific to the MySQL implementation of ROLLUP.

When you use ROLLUP, you cannot also use an ORDER BY clause to sort the results. In other words, ROLLUP and ORDER BY are mutually exclusive. However, you still have some control over sort order. GROUP BY in MySQL sorts results, and you can use explicit ASC and DESC keywords with columns named in the GROUP BY list to specify sort order for individual columns. (The higher-level summary rows added by ROLLUP still appear after the rows from which they are calculated, regardless of the sort order.)

LIMIT can be used to restrict the number of rows returned to the client. LIMIT is applied after ROLLUP, so the limit applies against the extra rows added by ROLLUP. For example:

```
mysql> SELECT year, country, product, SUM(profit)
    -> FROM sales
    -> GROUP BY year, country, product WITH ROLLUP
    -> LIMIT 5;
+------+---------+------------+-------------+
| year | country | product    | SUM(profit) |
+------+---------+------------+-------------+
| 2000 | Finland | Computer   |        1500 |
| 2000 | Finland | Phone      |         100 |
| 2000 | Finland | NULL       |        1600 |
| 2000 | India   | Calculator |         150 |
| 2000 | India   | Computer   |        1200 |
+------+---------+------------+-------------+
```

Using LIMIT with ROLLUP may produce results that are more difficult to interpret, because you have less context for understanding the super-aggregate rows.

The NULL indicators in each super-aggregate row are produced when the row is sent to the client. The server looks at the columns named in the GROUP BY clause following the leftmost one that has changed value. For any column in the result set with a name that is a lexical match to any of those names, its value is set to NULL. (If you specify grouping columns by column number, the server identifies which columns to set to NULL by number.)

Because the NULL values in the super-aggregate rows are placed into the result set at such a late stage in query processing, you cannot test them as NULL values within the query itself. For example, you cannot add HAVING product IS NULL to the query to eliminate from the output all but the super-aggregate rows.

On the other hand, the NULL values do appear as NULL on the client side and can be tested as such using any MySQL client programming interface.

MySQL permits a column that does not appear in the GROUP BY list to be named in the select list. In this case, the server is free to choose any value from this nonaggregated column in summary rows, and this includes the extra rows added by WITH ROLLUP. For example, in the following query, country is a nonaggregated column that does not appear in the GROUP BY list and values chosen for this column are indeterminate:

```
mysql> SELECT year, country, SUM(profit)
    -> FROM sales GROUP BY year WITH ROLLUP;
+------+---------+-------------+
| year | country | SUM(profit) |
+------+---------+-------------+
| 2000 | India   |        4525 |
| 2001 | USA     |        3010 |
| NULL | USA     |        7535 |
+------+---------+-------------+
```

This behavior occurs if the ONLY_FULL_GROUP_BY SQL mode is not enabled. If that mode is enabled, the server rejects the query as illegal because country is not listed in the GROUP BY clause. For more information about nonaggregated columns and GROUP BY, see Section 12.17.3, “MySQL Extensions to GROUP BY”.

12.17.3 MySQL Extensions to GROUP BY

In standard SQL, a query that includes a GROUP BY clause cannot refer to nonaggregated columns in the select list that are not named in the GROUP BY clause. For example, this query is illegal in standard SQL because the name column in the select list does not appear in the GROUP BY:

```
SELECT o.custid, c.name, MAX(o.payment)
  FROM orders AS o, customers AS c
  WHERE o.custid = c.custid
  GROUP BY o.custid;
```

For the query to be legal, the name column must be omitted from the select list or named in the GROUP BY clause.

MySQL extends the use of GROUP BY so that the select list can refer to nonaggregated columns not named in the GROUP BY clause. This means that the preceding query is legal in MySQL. You can use this feature to get better performance by avoiding unnecessary column sorting and grouping. However, this is useful primarily when all values in each nonaggregated column not named in the GROUP BY are the same for each group. The server is free to choose any value from each group, so unless they are the same, the values chosen are indeterminate. Furthermore, the selection of values from each group cannot be influenced by adding an ORDER BY clause. Sorting of the result set occurs after values have been chosen, and ORDER BY does not affect which values within each group the server chooses.

A similar MySQL extension applies to the HAVING clause. In standard SQL, a query that includes a GROUP BY clause cannot refer to nonaggregated columns in the HAVING clause that are not named in the GROUP BY clause. A MySQL extension permits references to such columns to simplify calculations. This extension assumes that the nongrouped columns will have the same group-wise values. Otherwise, the result is indeterminate.

To disable the MySQL GROUP BY extension, enable the ONLY_FULL_GROUP_BY SQL mode. This enables standard SQL behavior: Columns not named in the GROUP BY clause cannot be used in the select list or HAVING clause unless enclosed in an aggregate function.

ONLY_FULL_GROUP_BY also affects use of aliases in the HAVING clauses. For example, the following query returns name values that occur only once in table orders:

```
SELECT name, COUNT(name) FROM orders
  GROUP BY name
  HAVING COUNT(name) = 1;
```

MySQL extends this behavior to permit the use of an alias in the HAVING clause for the aggregated column:

```
SELECT name, COUNT(name) AS c FROM orders
  GROUP BY name
  HAVING c = 1;
```

Enabling ONLY_FULL_GROUP_BY disables this MySQL extension and a non-grouping field 'c' is used in HAVING clause error occurs because the column c in the HAVING clause is not enclosed in an aggregate function (instead, it *is* an aggregate function).

The select list extension also applies to ORDER BY. That is, you can refer to nonaggregated columns in the ORDER BY clause that do not appear in the GROUP BY clause. (However, as mentioned previously, ORDER BY does not affect which values are chosen from nonaggregated columns; it only sorts them after they have been chosen.) This extension does not apply if the ONLY_FULL_GROUP_BY SQL mode is enabled.

In some cases, you can use MIN() and MAX() to obtain a specific column value even if it is not unique. If the sort column contains integers no larger than 6 digits, the following query gives the value of column from the row containing the smallest sort value:

```
SUBSTR(MIN(CONCAT(LPAD(sort,6,'0'),column)),7)
```

See Section 3.6.4, “The Rows Holding the Group-wise Maximum of a Certain Column”.

If you are trying to follow standard SQL, you cannot use expressions in `GROUP BY` clauses. As a workaround, use an alias for the expression:

```
SELECT id, FLOOR(value/100) AS val
  FROM tbl_name
  GROUP BY id, val;
```

MySQL permits expressions in `GROUP BY` clauses, so the alias is unnecessary:

```
SELECT id, FLOOR(value/100)
  FROM tbl_name
  GROUP BY id, FLOOR(value/100);
```

12.18 Spatial Extensions

MySQL supports spatial extensions to enable the generation, storage, and analysis of geographic features. These features are available for `MyISAM`, `InnoDB`, `NDB`, and `ARCHIVE` tables.

For spatial columns, `MyISAM` supports both `SPATIAL` and non-`SPATIAL` indexes. Other storage engines support non-`SPATIAL` indexes, as described in Section 13.1.11, “`CREATE INDEX` Syntax”.

This chapter covers the following topics:

- The basis of these spatial extensions in the OpenGIS geometry model
- Data formats for representing spatial data
- How to use spatial data in MySQL
- Use of indexing for spatial data
- MySQL differences from the OpenGIS specification

Additional Resources

- The Open Geospatial Consortium publishes the *OpenGIS® Simple Features Specifications For SQL*, a document that proposes several conceptual ways for extending an SQL RDBMS to support spatial data. This specification is available from the OGC Web site at http://www.opengis.org/docs/99-049.pdf.
- If you have questions or concerns about the use of the spatial extensions to MySQL, you can discuss them in the GIS forum: http://forums.mysql.com/list.php?23.

12.18.1 Introduction to MySQL Spatial Support

MySQL implements spatial extensions following the specification of the Open Geospatial Consortium (OGC). This is an international consortium of more than 250 companies, agencies, and universities participating in the development of publicly available conceptual solutions that can be useful with all kinds of applications that manage spatial data. The OGC maintains a Web site at http://www.opengis.org/.

In 1997, the Open Geospatial Consortium published the *OpenGIS® Simple Features Specifications For SQL*, a document that proposes several conceptual ways for extending an SQL RDBMS to support spatial data. This specification is available from the OGC Web site at http://www.opengis.org/docs/99-049.pdf. It contains additional information relevant to this chapter.

MySQL implements a subset of the **SQL with Geometry Types** environment proposed by OGC. This term refers to an SQL environment that has been extended with a set of geometry types. A geometry-valued SQL column is implemented as a column that has a geometry type. The specification describe a set of SQL geometry types, as well as functions on those types to create and analyze geometry values.

A **geographic feature** is anything in the world that has a location. A feature can be:

- An entity. For example, a mountain, a pond, a city.
- A space. For example, town district, the tropics.
- A definable location. For example, a crossroad, as a particular place where two streets intersect.

Some documents use the term **geospatial feature** to refer to geographic features.

Geometry is another word that denotes a geographic feature. Originally the word **geometry** meant measurement of the earth. Another meaning comes from cartography, referring to the geometric features that cartographers use to map the world.

This chapter uses all of these terms synonymously: **geographic feature**, **geospatial feature**, **feature**, or **geometry**. Here, the term most commonly used is **geometry**, defined as *a point or an aggregate of points representing anything in the world that has a location*.

12.18.2 The OpenGIS Geometry Model

The set of geometry types proposed by OGC's **SQL with Geometry Types** environment is based on the **OpenGIS Geometry Model**. In this model, each geometric object has the following general properties:

- It is associated with a Spatial Reference System, which describes the coordinate space in which the object is defined.
- It belongs to some geometry class.

12.18.2.1 The Geometry Class Hierarchy

The geometry classes define a hierarchy as follows:

- `Geometry` (noninstantiable)
  - `Point` (instantiable)
  - `Curve` (noninstantiable)
    - `LineString` (instantiable)
      - `Line`
      - `LinearRing`
  - `Surface` (noninstantiable)
    - `Polygon` (instantiable)
  - `GeometryCollection` (instantiable)
    - `MultiPoint` (instantiable)
    - `MultiCurve` (noninstantiable)

    - `MultiLineString` (instantiable)

  - `MultiSurface` (noninstantiable)

    - `MultiPolygon` (instantiable)

It is not possible to create objects in noninstantiable classes. It is possible to create objects in instantiable classes. All classes have properties, and instantiable classes may also have assertions (rules that define valid class instances).

`Geometry` is the base class. It is an abstract class. The instantiable subclasses of `Geometry` are restricted to zero-, one-, and two-dimensional geometric objects that exist in two-dimensional coordinate space. All instantiable geometry classes are defined so that valid instances of a geometry class are topologically closed (that is, all defined geometries include their boundary).

The base `Geometry` class has subclasses for `Point`, `Curve`, `Surface`, and `GeometryCollection`:

- `Point` represents zero-dimensional objects.

- `Curve` represents one-dimensional objects, and has subclass `LineString`, with sub-subclasses `Line` and `LinearRing`.

- `Surface` is designed for two-dimensional objects and has subclass `Polygon`.

- `GeometryCollection` has specialized zero-, one-, and two-dimensional collection classes named `MultiPoint`, `MultiLineString`, and `MultiPolygon` for modeling geometries corresponding to collections of `Points`, `LineStrings`, and `Polygons`, respectively. `MultiCurve` and `MultiSurface` are introduced as abstract superclasses that generalize the collection interfaces to handle `Curves` and `Surfaces`.

`Geometry`, `Curve`, `Surface`, `MultiCurve`, and `MultiSurface` are defined as noninstantiable classes. They define a common set of methods for their subclasses and are included for extensibility.

`Point`, `LineString`, `Polygon`, `GeometryCollection`, `MultiPoint`, `MultiLineString`, and `MultiPolygon` are instantiable classes.

12.18.2.2 Class `Geometry`

`Geometry` is the root class of the hierarchy. It is a noninstantiable class but has a number of properties that are common to all geometry values created from any of the `Geometry` subclasses. These properties are described in the following list. Particular subclasses have their own specific properties, described later.

Geometry Properties

A geometry value has the following properties:

- Its **type**. Each geometry belongs to one of the instantiable classes in the hierarchy.

- Its **SRID**, or Spatial Reference Identifier. This value identifies the geometry's associated Spatial Reference System that describes the coordinate space in which the geometry object is defined.

  In MySQL, the SRID value is just an integer associated with the geometry value. All calculations are done assuming Euclidean (planar) geometry.

- Its **coordinates** in its Spatial Reference System, represented as double-precision (8-byte) numbers. All nonempty geometries include at least one pair of (X,Y) coordinates. Empty geometries contain no coordinates.

Coordinates are related to the SRID. For example, in different coordinate systems, the distance between two objects may differ even when objects have the same coordinates, because the distance on the **planar** coordinate system and the distance on the **geocentric** system (coordinates on the Earth's surface) are different things.

- Its **interior**, **boundary**, and **exterior**.

  Every geometry occupies some position in space. The exterior of a geometry is all space not occupied by the geometry. The interior is the space occupied by the geometry. The boundary is the interface between the geometry's interior and exterior.

- Its **MBR** (Minimum Bounding Rectangle), or Envelope. This is the bounding geometry, formed by the minimum and maximum (X,Y) coordinates:

  ```
  ((MINX MINY, MAXX MINY, MAXX MAXY, MINX MAXY, MINX MINY))
  ```

- Whether the value is **simple** or **nonsimple**. Geometry values of types (`LineString`, `MultiPoint`, `MultiLineString`) are either simple or nonsimple. Each type determines its own assertions for being simple or nonsimple.

- Whether the value is **closed** or **not closed**. Geometry values of types (`LineString`, `MultiString`) are either closed or not closed. Each type determines its own assertions for being closed or not closed.

- Whether the value is **empty** or **nonempty** A geometry is empty if it does not have any points. Exterior, interior, and boundary of an empty geometry are not defined (that is, they are represented by a `NULL` value). An empty geometry is defined to be always simple and has an area of 0.

- Its **dimension**. A geometry can have a dimension of –1, 0, 1, or 2:

  - –1 for an empty geometry.
  - 0 for a geometry with no length and no area.
  - 1 for a geometry with nonzero length and zero area.
  - 2 for a geometry with nonzero area.

  `Point` objects have a dimension of zero. `LineString` objects have a dimension of 1. `Polygon` objects have a dimension of 2. The dimensions of `MultiPoint`, `MultiLineString`, and `MultiPolygon` objects are the same as the dimensions of the elements they consist of.

12.18.2.3 Class `Point`

A `Point` is a geometry that represents a single location in coordinate space.

`Point` Examples

- Imagine a large-scale map of the world with many cities. A `Point` object could represent each city.
- On a city map, a `Point` object could represent a bus stop.

`Point` Properties

- X-coordinate value.
- Y-coordinate value.
- `Point` is defined as a zero-dimensional geometry.

- The boundary of a `Point` is the empty set.

12.18.2.4 Class `Curve`

A `Curve` is a one-dimensional geometry, usually represented by a sequence of points. Particular subclasses of `Curve` define the type of interpolation between points. `Curve` is a noninstantiable class.

`Curve` Properties

- A `Curve` has the coordinates of its points.
- A `Curve` is defined as a one-dimensional geometry.
- A `Curve` is simple if it does not pass through the same point twice.
- A `Curve` is closed if its start point is equal to its endpoint.
- The boundary of a closed `Curve` is empty.
- The boundary of a nonclosed `Curve` consists of its two endpoints.
- A `Curve` that is simple and closed is a `LinearRing`.

12.18.2.5 Class `LineString`

A `LineString` is a `Curve` with linear interpolation between points.

`LineString` Examples

- On a world map, `LineString` objects could represent rivers.
- In a city map, `LineString` objects could represent streets.

`LineString` Properties

- A `LineString` has coordinates of segments, defined by each consecutive pair of points.
- A `LineString` is a `Line` if it consists of exactly two points.
- A `LineString` is a `LinearRing` if it is both closed and simple.

12.18.2.6 Class `Surface`

A `Surface` is a two-dimensional geometry. It is a noninstantiable class. Its only instantiable subclass is `Polygon`.

`Surface` Properties

- A `Surface` is defined as a two-dimensional geometry.
- The OpenGIS specification defines a simple `Surface` as a geometry that consists of a single “patch” that is associated with a single exterior boundary and zero or more interior boundaries.
- The boundary of a simple `Surface` is the set of closed curves corresponding to its exterior and interior boundaries.

12.18.2.7 Class `Polygon`

A `Polygon` is a planar `Surface` representing a multisided geometry. It is defined by a single exterior boundary and zero or more interior boundaries, where each interior boundary defines a hole in the `Polygon`.

Polygon Examples

- On a region map, Polygon objects could represent forests, districts, and so on.

Polygon Assertions

- The boundary of a Polygon consists of a set of LinearRing objects (that is, LineString objects that are both simple and closed) that make up its exterior and interior boundaries.
- A Polygon has no rings that cross. The rings in the boundary of a Polygon may intersect at a Point, but only as a tangent.
- A Polygon has no lines, spikes, or punctures.
- A Polygon has an interior that is a connected point set.
- A Polygon may have holes. The exterior of a Polygon with holes is not connected. Each hole defines a connected component of the exterior.

The preceding assertions make a Polygon a simple geometry.

12.18.2.8 Class GeometryCollection

A GeometryCollection is a geometry that is a collection of one or more geometries of any class.

All the elements in a GeometryCollection must be in the same Spatial Reference System (that is, in the same coordinate system). There are no other constraints on the elements of a GeometryCollection, although the subclasses of GeometryCollection described in the following sections may restrict membership. Restrictions may be based on:

- Element type (for example, a MultiPoint may contain only Point elements)
- Dimension
- Constraints on the degree of spatial overlap between elements

12.18.2.9 Class MultiPoint

A MultiPoint is a geometry collection composed of Point elements. The points are not connected or ordered in any way.

MultiPoint Examples

- On a world map, a MultiPoint could represent a chain of small islands.
- On a city map, a MultiPoint could represent the outlets for a ticket office.

MultiPoint Properties

- A MultiPoint is a zero-dimensional geometry.
- A MultiPoint is simple if no two of its Point values are equal (have identical coordinate values).
- The boundary of a MultiPoint is the empty set.

12.18.2.10 Class MultiCurve

A MultiCurve is a geometry collection composed of Curve elements. MultiCurve is a noninstantiable class.

`MultiCurve` Properties

- A `MultiCurve` is a one-dimensional geometry.
- A `MultiCurve` is simple if and only if all of its elements are simple; the only intersections between any two elements occur at points that are on the boundaries of both elements.
- A `MultiCurve` boundary is obtained by applying the “mod 2 union rule” (also known as the “odd-even rule”): A point is in the boundary of a `MultiCurve` if it is in the boundaries of an odd number of `MultiCurve` elements.
- A `MultiCurve` is closed if all of its elements are closed.
- The boundary of a closed `MultiCurve` is always empty.

12.18.2.11 Class `MultiLineString`

A `MultiLineString` is a `MultiCurve` geometry collection composed of `LineString` elements.

`MultiLineString` Examples

- On a region map, a `MultiLineString` could represent a river system or a highway system.

12.18.2.12 Class `MultiSurface`

A `MultiSurface` is a geometry collection composed of surface elements. `MultiSurface` is a noninstantiable class. Its only instantiable subclass is `MultiPolygon`.

`MultiSurface` Assertions

- Two `MultiSurface` surfaces have no interiors that intersect.
- Two `MultiSurface` elements have boundaries that intersect at most at a finite number of points.

12.18.2.13 Class `MultiPolygon`

A `MultiPolygon` is a `MultiSurface` object composed of `Polygon` elements.

`MultiPolygon` Examples

- On a region map, a `MultiPolygon` could represent a system of lakes.

`MultiPolygon` Assertions

- A `MultiPolygon` has no two `Polygon` elements with interiors that intersect.
- A `MultiPolygon` has no two `Polygon` elements that cross (crossing is also forbidden by the previous assertion), or that touch at an infinite number of points.
- A `MultiPolygon` may not have cut lines, spikes, or punctures. A `MultiPolygon` is a regular, closed point set.
- A `MultiPolygon` that has more than one `Polygon` has an interior that is not connected. The number of connected components of the interior of a `MultiPolygon` is equal to the number of `Polygon` values in the `MultiPolygon`.

`MultiPolygon` Properties

- A `MultiPolygon` is a two-dimensional geometry.

- A MultiPolygon boundary is a set of closed curves (LineString values) corresponding to the boundaries of its Polygon elements.
- Each Curve in the boundary of the MultiPolygon is in the boundary of exactly one Polygon element.
- Every Curve in the boundary of an Polygon element is in the boundary of the MultiPolygon.

12.18.3 Supported Spatial Data Formats

This section describes the standard spatial data formats that are used to represent geometry objects in queries. They are:

- Well-Known Text (WKT) format
- Well-Known Binary (WKB) format

Internally, MySQL stores geometry values in a format that is not identical to either WKT or WKB format.

12.18.3.1 Well-Known Text (WKT) Format

The Well-Known Text (WKT) representation of Geometry is designed to exchange geometry data in ASCII form. For a Backus-Naur grammar that specifies the formal production rules for writing WKT values, see the OpenGIS specification document referenced in Section 12.18, “Spatial Extensions”.

Examples of WKT representations of geometry objects:

- A Point:

  ```
  POINT(15 20)
  ```

  Note that point coordinates are specified with no separating comma. This differs from the syntax for the SQL POINT() function, which requires a comma between the coordinates. Take care to use the syntax appropriate to the context of a given spatial operation. For example, the following statements both extract the X-coordinate from a Point object. The first produces the object directly using the POINT() function. The second uses a WKT representation converted to a Point with GeomFromText().

  ```
  mysql> SELECT X(POINT(15, 20));
  +------------------+
  | X(POINT(15, 20)) |
  +------------------+
  |               15 |
  +------------------+

  mysql> SELECT X(GeomFromText('POINT(15 20)'));
  +---------------------------------+
  | X(GeomFromText('POINT(15 20)')) |
  +---------------------------------+
  |                              15 |
  +---------------------------------+
  ```

- A LineString with four points:

  ```
  LINESTRING(0 0, 10 10, 20 25, 50 60)
  ```

  Note that point coordinate pairs are separated by commas.

- A Polygon with one exterior ring and one interior ring:

```
POLYGON((0 0,10 0,10 10,0 10,0 0),(5 5,7 5,7 7,5 7, 5 5))
```

- A MultiPoint with three Point values:

```
MULTIPOINT(0 0, 20 20, 60 60)
```

- A MultiLineString with two LineString values:

```
MULTILINESTRING((10 10, 20 20), (15 15, 30 15))
```

- A MultiPolygon with two Polygon values:

```
MULTIPOLYGON(((0 0,10 0,10 10,0 10,0 0)),((5 5,7 5,7 7,5 7, 5 5)))
```

- A GeometryCollection consisting of two Point values and one LineString:

```
GEOMETRYCOLLECTION(POINT(10 10), POINT(30 30), LINESTRING(15 15, 20 20))
```

12.18.3.2 Well-Known Binary (WKB) Format

The Well-Known Binary (WKB) representation for geometric values is defined by the OpenGIS specification. It is also defined in the ISO *SQL/MM Part 3: Spatial* standard.

WKB is used to exchange geometry data as binary streams represented by BLOB values containing geometric WKB information.

WKB uses 1-byte unsigned integers, 4-byte unsigned integers, and 8-byte double-precision numbers (IEEE 754 format). A byte is eight bits.

For example, a WKB value that corresponds to POINT(1 1) consists of this sequence of 21 bytes (each represented here by two hex digits):

```
0101000000000000000000F03F000000000000F03F
```

The sequence may be broken down into these components:

```
Byte order : 01
WKB type   : 01000000
X          : 000000000000F03F
Y          : 000000000000F03F
```

Component representation is as follows:

- The byte order may be either 1 or 0 to indicate little-endian or big-endian storage. The little-endian and big-endian byte orders are also known as Network Data Representation (NDR) and External Data Representation (XDR), respectively.

- The WKB type is a code that indicates the geometry type. Values from 1 through 7 indicate Point, LineString, Polygon, MultiPoint, MultiLineString, MultiPolygon, and GeometryCollection.

- A Point value has X and Y coordinates, each represented as a double-precision value.

WKB values for more complex geometry values are represented by more complex data structures, as detailed in the OpenGIS specification.

12.18.4 Creating a Spatially Enabled MySQL Database

This section describes the data types you can use for representing spatial data in MySQL, and the functions available for creating and retrieving spatial values.

12.18.4.1 MySQL Spatial Data Types

MySQL has data types that correspond to OpenGIS classes. Some of these types hold single geometry values:

- `GEOMETRY`
- `POINT`
- `LINESTRING`
- `POLYGON`

`GEOMETRY` can store geometry values of any type. The other single-value types (`POINT`, `LINESTRING`, and `POLYGON`) restrict their values to a particular geometry type.

The other data types hold collections of values:

- `MULTIPOINT`
- `MULTILINESTRING`
- `MULTIPOLYGON`
- `GEOMETRYCOLLECTION`

`GEOMETRYCOLLECTION` can store a collection of objects of any type. The other collection types (`MULTIPOINT`, `MULTILINESTRING`, `MULTIPOLYGON`, and `GEOMETRYCOLLECTION`) restrict collection members to those having a particular geometry type.

12.18.4.2 Creating Spatial Values

This section describes how to create spatial values using Well-Known Text and Well-Known Binary functions that are defined in the OpenGIS standard, and using MySQL-specific functions.

Creating Geometry Values Using WKT Functions

MySQL provides a number of functions that take as arguments a Well-Known Text representation and, optionally, a spatial reference system identifier (SRID). They return the corresponding geometry.

`GeomFromText()` accepts a WKT of any geometry type as its first argument. An implementation also provides type-specific construction functions for construction of geometry values of each geometry type.

- `GeomCollFromText(`*`wkt`*`[,`*`srid`*`])`, `GeometryCollectionFromText(`*`wkt`*`[,`*`srid`*`])`

  Constructs a `GEOMETRYCOLLECTION` value using its WKT representation and SRID.

- `GeomFromText(`*`wkt`*`[,`*`srid`*`])`, `GeometryFromText(`*`wkt`*`[,`*`srid`*`])`

  Constructs a geometry value of any type using its WKT representation and SRID.

- `LineFromText(`*`wkt`*`[,`*`srid`*`])`, `LineStringFromText(`*`wkt`*`[,`*`srid`*`])`

  Constructs a `LINESTRING` value using its WKT representation and SRID.

- `MLineFromText(wkt[,srid])`, `MultiLineStringFromText(wkt[,srid])`

  Constructs a `MULTILINESTRING` value using its WKT representation and SRID.

- `MPointFromText(wkt[,srid])`, `MultiPointFromText(wkt[,srid])`

  Constructs a `MULTIPOINT` value using its WKT representation and SRID.

- `MPolyFromText(wkt[,srid])`, `MultiPolygonFromText(wkt[,srid])`

  Constructs a `MULTIPOLYGON` value using its WKT representation and SRID.

- `PointFromText(wkt[,srid])`

  Constructs a `POINT` value using its WKT representation and SRID.

- `PolyFromText(wkt[,srid])`, `PolygonFromText(wkt[,srid])`

  Constructs a `POLYGON` value using its WKT representation and SRID.

The OpenGIS specification also defines the following optional functions, which MySQL does not implement. These functions construct `Polygon` or `MultiPolygon` values based on the WKT representation of a collection of rings or closed `LineString` values. These values may intersect.

- `BdMPolyFromText(wkt,srid)`

  Constructs a `MultiPolygon` value from a `MultiLineString` value in WKT format containing an arbitrary collection of closed `LineString` values.

- `BdPolyFromText(wkt,srid)`

  Constructs a `Polygon` value from a `MultiLineString` value in WKT format containing an arbitrary collection of closed `LineString` values.

Creating Geometry Values Using WKB Functions

MySQL provides a number of functions that take as arguments a `BLOB` containing a Well-Known Binary representation and, optionally, a spatial reference system identifier (SRID). They return the corresponding geometry.

These functions also accept geometry objects for compatibility with the return value of the functions in Creating Geometry Values Using MySQL-Specific Functions. Thus, those functions may be used to provide the first argument to the functions in this section.

- `GeomCollFromWKB(wkb[,srid])`, `GeometryCollectionFromWKB(wkb[,srid])`

  Constructs a `GEOMETRYCOLLECTION` value using its WKB representation and SRID.

- `GeomFromWKB(wkb[,srid])`, `GeometryFromWKB(wkb[,srid])`

  Constructs a geometry value of any type using its WKB representation and SRID.

- `LineFromWKB(wkb[,srid])`, `LineStringFromWKB(wkb[,srid])`

  Constructs a `LINESTRING` value using its WKB representation and SRID.

- `MLineFromWKB(wkb[,srid])`, `MultiLineStringFromWKB(wkb[,srid])`

  Constructs a `MULTILINESTRING` value using its WKB representation and SRID.

- MPointFromWKB(*wkb*[,*srid*]), MultiPointFromWKB(*wkb*[,*srid*])

  Constructs a MULTIPOINT value using its WKB representation and SRID.

- MPolyFromWKB(*wkb*[,*srid*]), MultiPolygonFromWKB(*wkb*[,*srid*])

  Constructs a MULTIPOLYGON value using its WKB representation and SRID.

- PointFromWKB(*wkb*[,*srid*])

  Constructs a POINT value using its WKB representation and SRID.

- PolyFromWKB(*wkb*[,*srid*]), PolygonFromWKB(*wkb*[,*srid*])

  Constructs a POLYGON value using its WKB representation and SRID.

The OpenGIS specification also describes optional functions for constructing Polygon or MultiPolygon values based on the WKB representation of a collection of rings or closed LineString values. These values may intersect. MySQL does not implement these functions:

- BdMPolyFromWKB(*wkb*,*srid*)

  Constructs a MultiPolygon value from a MultiLineString value in WKB format containing an arbitrary collection of closed LineString values.

- BdPolyFromWKB(*wkb*,*srid*)

  Constructs a Polygon value from a MultiLineString value in WKB format containing an arbitrary collection of closed LineString values.

Creating Geometry Values Using MySQL-Specific Functions

MySQL provides a set of useful nonstandard functions for creating geometry values. The functions described in this section are MySQL extensions to the OpenGIS specification.

These functions produce geometry objects from either WKB values or geometry objects as arguments. If any argument is not a proper WKB or geometry representation of the proper object type, the return value is NULL.

For example, you can insert the geometry return value from Point() directly into a Point column:

```
INSERT INTO t1 (pt_col) VALUES(Point(1,2));
```

- GeometryCollection(*g1*,*g2*,...)

  Constructs a GeometryCollection.

- LineString(*pt1*,*pt2*,...)

  Constructs a LineString value from a number of Point or WKB Point arguments. If the number of arguments is less than two, the return value is NULL.

- MultiLineString(*ls1*,*ls2*,...)

  Constructs a MultiLineString value using LineString or WKB LineString arguments.

- MultiPoint(*pt1*,*pt2*,...)

  Constructs a MultiPoint value using Point or WKB Point arguments.

- `MultiPolygon(poly1,poly2,...)`

  Constructs a `MultiPolygon` value from a set of `Polygon` or WKB `Polygon` arguments.

- `Point(x,y)`

  Constructs a `Point` using its coordinates.

- `Polygon(ls1,ls2,...)`

  Constructs a `Polygon` value from a number of `LineString` or WKB `LineString` arguments. If any argument does not represent a `LinearRing` (that is, not a closed and simple `LineString`), the return value is `NULL`.

12.18.4.3 Creating Spatial Columns

MySQL provides a standard way of creating spatial columns for geometry types, for example, with `CREATE TABLE` or `ALTER TABLE`. Currently, spatial columns are supported for `MyISAM`, `InnoDB`, `NDB`, and `ARCHIVE` tables. See also the annotations about spatial indexes under Section 12.18.6.1, “Creating Spatial Indexes”.

- Use the `CREATE TABLE` statement to create a table with a spatial column:

```
CREATE TABLE geom (g GEOMETRY);
```

- Use the `ALTER TABLE` statement to add or drop a spatial column to or from an existing table:

```
ALTER TABLE geom ADD pt POINT;
ALTER TABLE geom DROP pt;
```

12.18.4.4 Populating Spatial Columns

After you have created spatial columns, you can populate them with spatial data.

Values should be stored in internal geometry format, but you can convert them to that format from either Well-Known Text (WKT) or Well-Known Binary (WKB) format. The following examples demonstrate how to insert geometry values into a table by converting WKT values into internal geometry format:

- Perform the conversion directly in the `INSERT` statement:

```
INSERT INTO geom VALUES (GeomFromText('POINT(1 1)'));

SET @g = 'POINT(1 1)';
INSERT INTO geom VALUES (GeomFromText(@g));
```

- Perform the conversion prior to the `INSERT`:

```
SET @g = GeomFromText('POINT(1 1)');
INSERT INTO geom VALUES (@g);
```

The following examples insert more complex geometries into the table:

```
SET @g = 'LINESTRING(0 0,1 1,2 2)';
INSERT INTO geom VALUES (GeomFromText(@g));

SET @g = 'POLYGON((0 0,10 0,10 10,0 10,0 0),(5 5,7 5,7 7,5 7, 5 5))';
INSERT INTO geom VALUES (GeomFromText(@g));
```

```
SET @g =
'GEOMETRYCOLLECTION(POINT(1 1),LINESTRING(0 0,1 1,2 2,3 3,4 4))';
INSERT INTO geom VALUES (GeomFromText(@g));
```

The preceding examples all use `GeomFromText()` to create geometry values. You can also use type-specific functions:

```
SET @g = 'POINT(1 1)';
INSERT INTO geom VALUES (PointFromText(@g));

SET @g = 'LINESTRING(0 0,1 1,2 2)';
INSERT INTO geom VALUES (LineStringFromText(@g));

SET @g = 'POLYGON((0 0,10 0,10 10,0 10,0 0),(5 5,7 5,7 7,5 7, 5 5))';
INSERT INTO geom VALUES (PolygonFromText(@g));

SET @g =
'GEOMETRYCOLLECTION(POINT(1 1),LINESTRING(0 0,1 1,2 2,3 3,4 4))';
INSERT INTO geom VALUES (GeomCollFromText(@g));
```

Note that if a client application program wants to use WKB representations of geometry values, it is responsible for sending correctly formed WKB in queries to the server. However, there are several ways of satisfying this requirement. For example:

- Inserting a `POINT(1 1)` value with hex literal syntax:

  ```
  mysql> INSERT INTO geom VALUES
      -> (GeomFromWKB(0x0101000000000000000000F03F000000000000F03F));
  ```

- An ODBC application can send a WKB representation, binding it to a placeholder using an argument of `BLOB` type:

  ```
  INSERT INTO geom VALUES (GeomFromWKB(?))
  ```

  Other programming interfaces may support a similar placeholder mechanism.

- In a C program, you can escape a binary value using `mysql_real_escape_string()` and include the result in a query string that is sent to the server. See Section 21.8.7.55, “`mysql_real_escape_string()`”.

12.18.4.5 Fetching Spatial Data

Geometry values stored in a table can be fetched in internal format. You can also convert them into WKT or WKB format.

- Fetching spatial data in internal format:

  Fetching geometry values using internal format can be useful in table-to-table transfers:

  ```
  CREATE TABLE geom2 (g GEOMETRY) SELECT g FROM geom;
  ```

- Fetching spatial data in WKT format:

  The `AsText()` function converts a geometry from internal format into a WKT string.

  ```
  SELECT AsText(g) FROM geom;
  ```

- Fetching spatial data in WKB format:

  The AsBinary() function converts a geometry from internal format into a BLOB containing the WKB value.

  ```
  SELECT AsBinary(g) FROM geom;
  ```

12.18.5 Spatial Analysis Functions

After populating spatial columns with values, you are ready to query and analyze them. MySQL provides a set of functions to perform various operations on spatial data. These functions can be grouped into four major categories according to the type of operation they perform:

- Functions that convert geometries between various formats
- Functions that provide access to qualitative or quantitative properties of a geometry
- Functions that describe relations between two geometries
- Functions that create new geometries from existing ones

Spatial analysis functions can be used in many contexts, such as:

- Any interactive SQL program, such as mysql.
- Application programs written in any language that supports a MySQL client API

12.18.5.1 Geometry Format Conversion Functions

MySQL supports the following functions for converting geometry values between internal format and either WKT or WKB format:

- AsBinary(*g*), AsWKB(*g*)

  Converts a value in internal geometry format to its WKB representation and returns the binary result.

  ```
  SELECT AsBinary(g) FROM geom;
  ```

- AsText(*g*), AsWKT(*g*)

  Converts a value in internal geometry format to its WKT representation and returns the string result.

  ```
  mysql> SET @g = 'LineString(1 1,2 2,3 3)';
  mysql> SELECT AsText(GeomFromText(@g));
  +--------------------------+
  | AsText(GeomFromText(@g)) |
  +--------------------------+
  | LINESTRING(1 1,2 2,3 3)  |
  +--------------------------+
  ```

- GeomFromText(*wkt*[,*srid*])

  Converts a string value from its WKT representation into internal geometry format and returns the result. A number of type-specific functions are also supported, such as PointFromText() and LineFromText(). See Creating Geometry Values Using WKT Functions.

- GeomFromWKB(*wkb*[,*srid*])

Converts a binary value from its WKB representation into internal geometry format and returns the result. A number of type-specific functions are also supported, such as `PointFromWKB()` and `LineFromWKB()`. See Creating Geometry Values Using WKB Functions.

12.18.5.2 `Geometry` Property Functions

Each function that belongs to this group takes a geometry value as its argument and returns some quantitative or qualitative property of the geometry. Some functions restrict their argument type. Such functions return `NULL` if the argument is of an incorrect geometry type. For example, `Area()` returns `NULL` if the object type is neither `Polygon` nor `MultiPolygon`.

General Geometry Functions

The functions listed in this section do not restrict their argument and accept a geometry value of any type.

- `Dimension(`*`g`*`)`

  Returns the inherent dimension of the geometry value *`g`*. The result can be –1, 0, 1, or 2. The meaning of these values is given in Section 12.18.2.2, “Class `Geometry`”.

```
mysql> SELECT Dimension(GeomFromText('LineString(1 1,2 2)'));
+------------------------------------------------+
| Dimension(GeomFromText('LineString(1 1,2 2)')) |
+------------------------------------------------+
|                                              1 |
+------------------------------------------------+
```

- `Envelope(`*`g`*`)`

  Returns the Minimum Bounding Rectangle (MBR) for the geometry value *`g`*. The result is returned as a `Polygon` value.

  The polygon is defined by the corner points of the bounding box:

```
POLYGON((MINX MINY, MAXX MINY, MAXX MAXY, MINX MAXY, MINX MINY))
```

```
mysql> SELECT AsText(Envelope(GeomFromText('LineString(1 1,2 2)')));
+-------------------------------------------------------+
| AsText(Envelope(GeomFromText('LineString(1 1,2 2)'))) |
+-------------------------------------------------------+
| POLYGON((1 1,2 1,2 2,1 2,1 1))                        |
+-------------------------------------------------------+
```

- `GeometryType(`*`g`*`)`

  Returns as a binary string the name of the geometry type of which the geometry instance *`g`* is a member. The name corresponds to one of the instantiable `Geometry` subclasses.

```
mysql> SELECT GeometryType(GeomFromText('POINT(1 1)'));
+------------------------------------------+
| GeometryType(GeomFromText('POINT(1 1)')) |
+------------------------------------------+
| POINT                                    |
+------------------------------------------+
```

- `IsSimple(`*`g`*`)`

Returns 1 if the geometry value *g* has no anomalous geometric points, such as self-intersection or self-tangency. `IsSimple()` returns 0 if the argument is not simple, and `NULL` if it is `NULL`.

The description of each instantiable geometric class given earlier in the chapter includes the specific conditions that cause an instance of that class to be classified as not simple. (See Section 12.18.2.1, “The Geometry Class Hierarchy”.)

- `SRID(g)`

  Returns an integer indicating the Spatial Reference System ID for the geometry value *g*.

  In MySQL, the SRID value is just an integer associated with the geometry value. All calculations are done assuming Euclidean (planar) geometry.

```
mysql> SELECT SRID(GeomFromText('LineString(1 1,2 2)',101));
+-----------------------------------------------+
| SRID(GeomFromText('LineString(1 1,2 2)',101)) |
+-----------------------------------------------+
|                                           101 |
+-----------------------------------------------+
```

The OpenGIS specification also defines the following functions, which MySQL does not implement:

- `Boundary(g)`

  Returns a geometry that is the closure of the combinatorial boundary of the geometry value *g*.

- `IsEmpty(g)`

  This function is a placeholder that returns 0 for any valid geometry value, 1 for any invalid geometry value or `NULL`.

  MySQL does not support GIS `EMPTY` values such as `POINT EMPTY`.

`Point` Functions

A `Point` consists of X and Y coordinates, which may be obtained using the following functions:

- `X(p)`

  Returns the X-coordinate value for the `Point` object *p* as a double-precision number.

```
mysql> SELECT X(POINT(56.7, 53.34));
+-----------------------+
| X(POINT(56.7, 53.34)) |
+-----------------------+
|                  56.7 |
+-----------------------+
```

- `Y(p)`

  Returns the Y-coordinate value for the `Point` object *p* as a double-precision number.

```
mysql> SELECT Y(POINT(56.7, 53.34));
+-----------------------+
| Y(POINT(56.7, 53.34)) |
+-----------------------+
|                 53.34 |
+------------------------+
```

LineString Functions

A LineString consists of Point values. You can extract particular points of a LineString, count the number of points that it contains, or obtain its length.

- EndPoint(*ls*)

  Returns the Point that is the endpoint of the LineString value *ls*.

```
mysql> SET @ls = 'LineString(1 1,2 2,3 3)';
mysql> SELECT AsText(EndPoint(GeomFromText(@ls)));
+-------------------------------------+
| AsText(EndPoint(GeomFromText(@ls))) |
+-------------------------------------+
| POINT(3 3)                          |
+-------------------------------------+
```

- GLength(*ls*)

  Returns as a double-precision number the length of the LineString value *ls* in its associated spatial reference.

```
mysql> SET @ls = 'LineString(1 1,2 2,3 3)';
mysql> SELECT GLength(GeomFromText(@ls));
+----------------------------+
| GLength(GeomFromText(@ls)) |
+----------------------------+
|            2.8284271247462 |
+----------------------------+
```

  GLength() is a nonstandard name. It corresponds to the OpenGIS Length() function.

- IsClosed(*ls*)

  Returns 1 if the LineString value *ls* is closed (that is, its StartPoint() and EndPoint() values are the same) and is simple (does not pass through the same point more than once). Returns 0 if *ls* is not closed, and –1 if it is NULL.

```
mysql> SET @ls1 = 'LineString(1 1,2 2,3 3,2 2)';
Query OK, 0 rows affected (0.00 sec)

mysql> SET @ls2 = 'LineString(1 1,2 2,3 3,1 1)';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT IsClosed(GeomFromText(@ls1));
+------------------------------+
| IsClosed(GeomFromText(@ls1)) |
+------------------------------+
|                            0 |
+------------------------------+
1 row in set (0.00 sec)

mysql> SELECT IsClosed(GeomFromText(@ls2));
+------------------------------+
| IsClosed(GeomFromText(@ls2)) |
+------------------------------+
|                            1 |
+------------------------------+
1 row in set (0.00 sec)
```

- `NumPoints(ls)`

  Returns the number of `Point` objects in the `LineString` value *`ls`*.

  ```
  mysql> SET @ls = 'LineString(1 1,2 2,3 3)';
  mysql> SELECT NumPoints(GeomFromText(@ls));
  +------------------------------+
  | NumPoints(GeomFromText(@ls)) |
  +------------------------------+
  |                            3 |
  +------------------------------+
  ```

- `PointN(ls,N)`

  Returns the *`N`*-th `Point` in the `Linestring` value *`ls`*. Points are numbered beginning with 1.

  ```
  mysql> SET @ls = 'LineString(1 1,2 2,3 3)';
  mysql> SELECT AsText(PointN(GeomFromText(@ls),2));
  +-------------------------------------+
  | AsText(PointN(GeomFromText(@ls),2)) |
  +-------------------------------------+
  | POINT(2 2)                          |
  +-------------------------------------+
  ```

- `StartPoint(ls)`

  Returns the `Point` that is the start point of the `LineString` value *`ls`*.

  ```
  mysql> SET @ls = 'LineString(1 1,2 2,3 3)';
  mysql> SELECT AsText(StartPoint(GeomFromText(@ls)));
  +---------------------------------------+
  | AsText(StartPoint(GeomFromText(@ls))) |
  +---------------------------------------+
  | POINT(1 1)                            |
  +---------------------------------------+
  ```

`MultiLineString` Functions

These functions return properties of `MultiLineString` values.

- `GLength(mls)`

  Returns as a double-precision number the length of the `MultiLineString` value *`mls`*. The length of *`mls`* is equal to the sum of the lengths of its elements.

  ```
  mysql> SET @mls = 'MultiLineString((1 1,2 2,3 3),(4 4,5 5))';
  mysql> SELECT GLength(GeomFromText(@mls));
  +-----------------------------+
  | GLength(GeomFromText(@mls)) |
  +-----------------------------+
  |             4.2426406871193 |
  +-----------------------------+
  ```

  `GLength()` is a nonstandard name. It corresponds to the OpenGIS `Length()` function.

- `IsClosed(mls)`

  Returns 1 if the `MultiLineString` value *`mls`* is closed (that is, the `StartPoint()` and `EndPoint()` values are the same for each `LineString` in *`mls`*). Returns 0 if *`mls`* is not closed, and –1 if it is `NULL`.

```
mysql> SET @mls = 'MultiLineString((1 1,2 2,3 3),(4 4,5 5))';
mysql> SELECT IsClosed(GeomFromText(@mls));
+------------------------------+
| IsClosed(GeomFromText(@mls)) |
+------------------------------+
|                            0 |
+------------------------------+
```

`Polygon` Functions

These functions return properties of `Polygon` values.

- `Area(`*`poly`*`)`

  Returns as a double-precision number the area of the `Polygon` value *`poly`*, as measured in its spatial reference system.

  ```
  mysql> SET @poly = 'Polygon((0 0,0 3,3 0,0 0),(1 1,1 2,2 1,1 1))';
  mysql> SELECT Area(GeomFromText(@poly));
  +---------------------------+
  | Area(GeomFromText(@poly)) |
  +---------------------------+
  |                         4 |
  +---------------------------+
  ```

- `ExteriorRing(`*`poly`*`)`

  Returns the exterior ring of the `Polygon` value *`poly`* as a `LineString`.

  ```
  mysql> SET @poly =
      -> 'Polygon((0 0,0 3,3 3,3 0,0 0),(1 1,1 2,2 2,2 1,1 1))';
  mysql> SELECT AsText(ExteriorRing(GeomFromText(@poly)));
  +-------------------------------------------+
  | AsText(ExteriorRing(GeomFromText(@poly))) |
  +-------------------------------------------+
  | LINESTRING(0 0,0 3,3 3,3 0,0 0)           |
  +-------------------------------------------+
  ```

- `InteriorRingN(`*`poly`*`,`*`N`*`)`

  Returns the *`N`*-th interior ring for the `Polygon` value *`poly`* as a `LineString`. Rings are numbered beginning with 1.

  ```
  mysql> SET @poly =
      -> 'Polygon((0 0,0 3,3 3,3 0,0 0),(1 1,1 2,2 2,2 1,1 1))';
  mysql> SELECT AsText(InteriorRingN(GeomFromText(@poly),1));
  +----------------------------------------------+
  | AsText(InteriorRingN(GeomFromText(@poly),1)) |
  +----------------------------------------------+
  | LINESTRING(1 1,1 2,2 2,2 1,1 1)              |
  +----------------------------------------------+
  ```

- `NumInteriorRings(`*`poly`*`)`

  Returns the number of interior rings in the `Polygon` value *`poly`*.

  ```
  mysql> SET @poly =
      -> 'Polygon((0 0,0 3,3 3,3 0,0 0),(1 1,1 2,2 2,2 1,1 1))';
  ```

```
mysql> SELECT NumInteriorRings(GeomFromText(@poly));
+---------------------------------------+
| NumInteriorRings(GeomFromText(@poly)) |
+---------------------------------------+
|                                     1 |
+---------------------------------------+
```

`MultiPolygon` Functions

These functions return properties of `MultiPolygon` values.

- `Area(`*`mpoly`*`)`

  Returns as a double-precision number the area of the `MultiPolygon` value *`mpoly`*, as measured in its spatial reference system.

```
mysql> SET @mpoly =
    -> 'MultiPolygon(((0 0,0 3,3 3,3 0,0 0),(1 1,1 2,2 2,2 1,1 1)))';
mysql> SELECT Area(GeomFromText(@mpoly));
+----------------------------+
| Area(GeomFromText(@mpoly)) |
+----------------------------+
|                          8 |
+----------------------------+
```

- `Centroid(`*`mpoly`*`)`

  Returns the mathematical centroid for the `MultiPolygon` value *`mpoly`* as a `Point`. The result is not guaranteed to be on the `MultiPolygon`.

```
mysql> SET @poly =
    -> GeomFromText('POLYGON((0 0,10 0,10 10,0 10,0 0),(5 5,7 5,7 7,5 7,5 5))');
mysql> SELECT GeometryType(@poly),AsText(Centroid(@poly));
+---------------------+--------------------------------------------+
| GeometryType(@poly) | AsText(Centroid(@poly))                    |
+---------------------+--------------------------------------------+
| POLYGON             | POINT(4.958333333333333 4.958333333333333) |
+---------------------+--------------------------------------------+
```

The OpenGIS specification also defines the following function, which MySQL does not implement:

- `PointOnSurface(`*`mpoly`*`)`

  Returns a `Point` value that is guaranteed to be on the `MultiPolygon` value *`mpoly`*.

`GeometryCollection` Functions

These functions return properties of `GeometryCollection` values.

- `GeometryN(`*`gc`*`,`*`N`*`)`

  Returns the *`N`*-th geometry in the `GeometryCollection` value *`gc`*. Geometries are numbered beginning with 1.

```
mysql> SET @gc = 'GeometryCollection(Point(1 1),LineString(2 2, 3 3))';
mysql> SELECT AsText(GeometryN(GeomFromText(@gc),1));
+----------------------------------------+
| AsText(GeometryN(GeomFromText(@gc),1)) |
+----------------------------------------+
| POINT(1 1)                             |
+----------------------------------------+
```

- `NumGeometries(gc)`

  Returns the number of geometries in the `GeometryCollection` value *`gc`*.

```
mysql> SET @gc = 'GeometryCollection(Point(1 1),LineString(2 2, 3 3))';
mysql> SELECT NumGeometries(GeomFromText(@gc));
+----------------------------------+
| NumGeometries(GeomFromText(@gc)) |
+----------------------------------+
|                                2 |
+----------------------------------+
```

12.18.5.3 Functions That Create New Geometries from Existing Ones

The following sections describe functions that take geometry values as arguments and return new geometry values.

Geometry Functions That Produce New Geometries

Section 12.18.5.2, “`Geometry` Property Functions”, discusses several functions that construct new geometries from existing ones. See that section for descriptions of these functions:

- `Envelope(g)`
- `StartPoint(ls)`
- `EndPoint(ls)`
- `PointN(ls,N)`
- `ExteriorRing(poly)`
- `InteriorRingN(poly,N)`
- `GeometryN(gc,N)`

Spatial Operators

OpenGIS proposes a number of other functions that can produce geometries. They are designed to implement spatial operators.

- `Buffer(g,d)`

  Returns a geometry that represents all points whose distance from the geometry value *`g`* is less than or equal to a distance of *`d`*.

  `Buffer()` supports negative distances for polygons, multipolygons, and geometry collections containing polygons or multipolygons. For point, multipoint, linestring, multilinestring, and geometry collections not containing any polygons or multipolygons, `Buffer()` with a negative distance returns `NULL`.

The OpenGIS specification also defines the following functions, which MySQL does not implement:

- `ConvexHull(g)`

  Returns a geometry that represents the convex hull of the geometry value *`g`*.

- `Difference(g1,g2)`

  Returns a geometry that represents the point set difference of the geometry value *`g1`* with *`g2`*.

- `Intersection(g1,g2)`

  Returns a geometry that represents the point set intersection of the geometry values *g1* with *g2*.

- `SymDifference(g1,g2)`

  Returns a geometry that represents the point set symmetric difference of the geometry value *g1* with *g2*.

- `Union(g1,g2)`

  Returns a geometry that represents the point set union of the geometry values *g1* and *g2*.

12.18.5.4 Functions for Testing Spatial Relations Between Geometric Objects

The functions described in these sections take two geometries as input parameters and return a qualitative or quantitative relation between them.

Relations on Geometry Minimal Bounding Rectangles (MBRs)

MySQL provides several functions that test relations between minimal bounding rectangles of two geometries `g1` and `g2`. The return values 1 and 0 indicate true and false, respectively.

- `MBRContains(g1,g2)`

  Returns 1 or 0 to indicate whether the Minimum Bounding Rectangle of *g1* contains the Minimum Bounding Rectangle of *g2*. This tests the opposite relationship as `MBRWithin()`.

```
mysql> SET @g1 = GeomFromText('Polygon((0 0,0 3,3 3,3 0,0 0))');
mysql> SET @g2 = GeomFromText('Point(1 1)');
mysql> SELECT MBRContains(@g1,@g2), MBRContains(@g2,@g1);
----------------------+----------------------+
| MBRContains(@g1,@g2) | MBRContains(@g2,@g1) |
+----------------------+----------------------+
|                    1 |                    0 |
+----------------------+----------------------+
```

- `MBRDisjoint(g1,g2)`

  Returns 1 or 0 to indicate whether the Minimum Bounding Rectangles of the two geometries *g1* and *g2* are disjoint (do not intersect).

- `MBREqual(g1,g2)`

  Returns 1 or 0 to indicate whether the Minimum Bounding Rectangles of the two geometries *g1* and *g2* are the same.

- `MBRIntersects(g1,g2)`

  Returns 1 or 0 to indicate whether the Minimum Bounding Rectangles of the two geometries *g1* and *g2* intersect.

- `MBROverlaps(g1,g2)`

  Returns 1 or 0 to indicate whether the Minimum Bounding Rectangles of the two geometries *g1* and *g2* overlap. The term *spatially overlaps* is used if two geometries intersect and their intersection results in a geometry of the same dimension but not equal to either of the given geometries.

- `MBRTouches(g1,g2)`

Returns 1 or 0 to indicate whether the Minimum Bounding Rectangles of the two geometries *g1* and *g2* touch. Two geometries *spatially touch* if the interiors of the geometries do not intersect, but the boundary of one of the geometries intersects either the boundary or the interior of the other.

- `MBRWithin(g1,g2)`

  Returns 1 or 0 to indicate whether the Minimum Bounding Rectangle of *g1* is within the Minimum Bounding Rectangle of *g2*. This tests the opposite relationship as `MBRContains()`.

```
mysql> SET @g1 = GeomFromText('Polygon((0 0,0 3,3 3,3 0,0 0))');
mysql> SET @g2 = GeomFromText('Polygon((0 0,0 5,5 5,5 0,0 0))');
mysql> SELECT MBRWithin(@g1,@g2), MBRWithin(@g2,@g1);
+--------------------+--------------------+
| MBRWithin(@g1,@g2) | MBRWithin(@g2,@g1) |
+--------------------+--------------------+
|                  1 |                  0 |
+--------------------+--------------------+
```

Functions That Test Spatial Relationships Between Geometries

The OpenGIS specification defines the following functions. They test the relationship between two geometry values `g1` and `g2`.

The return values 1 and 0 indicate true and false, respectively.

> **Note**
>
> MySQL originally implemented these functions such that they used object bounding rectangles and returned the same result as the corresponding MBR-based functions. Corresponding versions are available that use precise object shapes. These versions are named with an `ST_` prefix. For example, `Contains()` uses object bounding rectangles, whereas `ST_Contains()` uses object shapes.
>
> There are also `ST_` aliases for existing spatial functions that were already exact. For example, `ST_IsEmpty()` is an alias for `IsEmpty()`

Functions That Use Object Shapes

- `ST_Contains(g1,g2)`

  Returns 1 or 0 to indicate whether *g1* completely contains *g2*. This tests the opposite relationship as `ST_Within()`.

- `ST_Crosses(g1,g2)`

  Returns 1 if *g1* spatially crosses *g2*. Returns `NULL` if `g1` is a `Polygon` or a `MultiPolygon`, or if *g2* is a `Point` or a `MultiPoint`. Otherwise, returns 0.

  The term *spatially crosses* denotes a spatial relation between two given geometries that has the following properties:

  - The two geometries intersect
  - Their intersection results in a geometry that has a dimension that is one less than the maximum dimension of the two given geometries
  - Their intersection is not equal to either of the two given geometries

- `ST_Disjoint(g1,g2)`

  Returns 1 or 0 to indicate whether *`g1`* is spatially disjoint from (does not intersect) *`g2`*.

- `ST_Equals(g1,g2)`

  Returns 1 or 0 to indicate whether *`g1`* is spatially equal to *`g2`*.

- `ST_Intersects(g1,g2)`

  Returns 1 or 0 to indicate whether *`g1`* spatially intersects *`g2`*.

- `ST_Overlaps(g1,g2)`

  Returns 1 or 0 to indicate whether *`g1`* spatially overlaps *`g2`*. The term *spatially overlaps* is used if two geometries intersect and their intersection results in a geometry of the same dimension but not equal to either of the given geometries.

- `ST_Touches(g1,g2)`

  Returns 1 or 0 to indicate whether *`g1`* spatially touches *`g2`*. Two geometries *spatially touch* if the interiors of the geometries do not intersect, but the boundary of one of the geometries intersects either the boundary or the interior of the other.

- `ST_Within(g1,g2)`

  Returns 1 or 0 to indicate whether *`g1`* is spatially within *`g2`*. This tests the opposite relationship as `ST_Contains()`.

Functions That Use Bounding Rectangles

- `Contains(g1,g2)`

  Returns 1 or 0 to indicate whether *`g1`* completely contains *`g2`*. This tests the opposite relationship as `Within()`.

- `Crosses(g1,g2)`

  Returns 1 if *`g1`* spatially crosses *`g2`*. Returns `NULL` if `g1` is a `Polygon` or a `MultiPolygon`, or if *`g2`* is a `Point` or a `MultiPoint`. Otherwise, returns 0.

  The term *spatially crosses* denotes a spatial relation between two given geometries that has the following properties:

  - The two geometries intersect

  - Their intersection results in a geometry that has a dimension that is one less than the maximum dimension of the two given geometries

  - Their intersection is not equal to either of the two given geometries

- `Disjoint(g1,g2)`

  Returns 1 or 0 to indicate whether *`g1`* is spatially disjoint from (does not intersect) *`g2`*.

- `Equals(g1,g2)`

  Returns 1 or 0 to indicate whether *`g1`* is spatially equal to *`g2`*.

- `Intersects(g1,g2)`

  Returns 1 or 0 to indicate whether *`g1`* spatially intersects *`g2`*.

- `Overlaps(g1,g2)`

  Returns 1 or 0 to indicate whether *`g1`* spatially overlaps *`g2`*. The term *spatially overlaps* is used if two geometries intersect and their intersection results in a geometry of the same dimension but not equal to either of the given geometries.

- `Touches(g1,g2)`

  Returns 1 or 0 to indicate whether *`g1`* spatially touches *`g2`*. Two geometries *spatially touch* if the interiors of the geometries do not intersect, but the boundary of one of the geometries intersects either the boundary or the interior of the other.

- `Within(g1,g2)`

  Returns 1 or 0 to indicate whether *`g1`* is spatially within *`g2`*. This tests the opposite relationship as `Contains()`.

12.18.6 Optimizing Spatial Analysis

For `MyISAM` tables, Search operations in nonspatial databases can be optimized using `SPATIAL` indexes. This is true for spatial databases as well. With the help of a great variety of multi-dimensional indexing methods that have previously been designed, it is possible to optimize spatial searches. The most typical of these are:

- Point queries that search for all objects that contain a given point

- Region queries that search for all objects that overlap a given region

MySQL uses **R-Trees with quadratic splitting** for `SPATIAL` indexes on spatial columns. A `SPATIAL` index is built using the MBR of a geometry. For most geometries, the MBR is a minimum rectangle that surrounds the geometries. For a horizontal or a vertical linestring, the MBR is a rectangle degenerated into the linestring. For a point, the MBR is a rectangle degenerated into the point.

It is also possible to create normal indexes on spatial columns. In a non-`SPATIAL` index, you must declare a prefix for any spatial column except for `POINT` columns.

`MyISAM` supports both `SPATIAL` and non-`SPATIAL` indexes. Other storage engines support non-`SPATIAL` indexes, as described in Section 13.1.11, “`CREATE INDEX` Syntax”.

12.18.6.1 Creating Spatial Indexes

For `MyISAM` tables, MySQL can create spatial indexes using syntax similar to that for creating regular indexes, but extended with the `SPATIAL` keyword. Currently, columns in spatial indexes must be declared `NOT NULL`. The following examples demonstrate how to create spatial indexes:

- With `CREATE TABLE`:

  ```
  CREATE TABLE geom (g GEOMETRY NOT NULL, SPATIAL INDEX(g)) ENGINE=MyISAM;
  ```

- With `ALTER TABLE`:

```
ALTER TABLE geom ADD SPATIAL INDEX(g);
```

- With CREATE INDEX:

```
CREATE SPATIAL INDEX sp_index ON geom (g);
```

For MyISAM tables, SPATIAL INDEX creates an R-tree index. For storage engines that support nonspatial indexing of spatial columns, the engine creates a B-tree index. A B-tree index on spatial values will be useful for exact-value lookups, but not for range scans.

For more information on indexing spatial columns, see Section 13.1.11, “CREATE INDEX Syntax”.

To drop spatial indexes, use ALTER TABLE or DROP INDEX:

- With ALTER TABLE:

```
ALTER TABLE geom DROP INDEX g;
```

- With DROP INDEX:

```
DROP INDEX sp_index ON geom;
```

Example: Suppose that a table geom contains more than 32,000 geometries, which are stored in the column g of type GEOMETRY. The table also has an AUTO_INCREMENT column fid for storing object ID values.

```
mysql> DESCRIBE geom;
+-------+----------+------+-----+---------+----------------+
| Field | Type     | Null | Key | Default | Extra          |
+-------+----------+------+-----+---------+----------------+
| fid   | int(11)  |      | PRI | NULL    | auto_increment |
| g     | geometry |      |     |         |                |
+-------+----------+------+-----+---------+----------------+
2 rows in set (0.00 sec)

mysql> SELECT COUNT(*) FROM geom;
+----------+
| count(*) |
+----------+
|    32376 |
+----------+
1 row in set (0.00 sec)
```

To add a spatial index on the column g, use this statement:

```
mysql> ALTER TABLE geom ADD SPATIAL INDEX(g);
Query OK, 32376 rows affected (4.05 sec)
Records: 32376  Duplicates: 0  Warnings: 0
```

12.18.6.2 Using a Spatial Index

The optimizer investigates whether available spatial indexes can be involved in the search for queries that use a function such as MBRContains() or MBRWithin() in the WHERE clause. The following query finds all objects that are in the given rectangle:

```
mysql> SET @poly =
    -> 'Polygon((30000 15000,
                               31000 15000,
                               31000 16000,
                               30000 16000,
                               30000 15000))';
mysql> SELECT fid,AsText(g) FROM geom WHERE
    -> MBRContains(GeomFromText(@poly),g);
+-----+---------------------------------------------------------------+
| fid | AsText(g)                                                     |
+-----+---------------------------------------------------------------+
|  21 | LINESTRING(30350.4 15828.8,30350.6 15845,30333.8 15845,30 ... |
|  22 | LINESTRING(30350.6 15871.4,30350.6 15887.8,30334 15887.8, ... |
|  23 | LINESTRING(30350.6 15914.2,30350.6 15930.4,30334 15930.4, ... |
|  24 | LINESTRING(30290.2 15823,30290.2 15839.4,30273.4 15839.4, ... |
|  25 | LINESTRING(30291.4 15866.2,30291.6 15882.4,30274.8 15882. ... |
|  26 | LINESTRING(30291.6 15918.2,30291.6 15934.4,30275 15934.4, ... |
| 249 | LINESTRING(30337.8 15938.6,30337.8 15946.8,30320.4 15946. ... |
|   1 | LINESTRING(30250.4 15129.2,30248.8 15138.4,30238.2 15136. ... |
|   2 | LINESTRING(30220.2 15122.8,30217.2 15137.8,30207.6 15136, ... |
|   3 | LINESTRING(30179 15114.4,30176.6 15129.4,30167 15128,3016 ... |
|   4 | LINESTRING(30155.2 15121.4,30140.4 15118.6,30142 15109,30 ... |
|   5 | LINESTRING(30192.4 15085,30177.6 15082.2,30179.2 15072.4, ... |
|   6 | LINESTRING(30244 15087,30229 15086.2,30229.4 15076.4,3024 ... |
|   7 | LINESTRING(30200.6 15059.4,30185.6 15058.6,30186 15048.8, ... |
|  10 | LINESTRING(30179.6 15017.8,30181 15002.8,30190.8 15003.6, ... |
|  11 | LINESTRING(30154.2 15000.4,30168.6 15004.8,30166 15014.2, ... |
|  13 | LINESTRING(30105 15065.8,30108.4 15050.8,30118 15053,3011 ... |
| 154 | LINESTRING(30276.2 15143.8,30261.4 15141,30263 15131.4,30 ... |
| 155 | LINESTRING(30269.8 15084,30269.4 15093.4,30258.6 15093,30 ... |
| 157 | LINESTRING(30128.2 15011,30113.2 15010.2,30113.6 15000.4, ... |
+-----+---------------------------------------------------------------+
20 rows in set (0.00 sec)
```

Use EXPLAIN to check the way this query is executed:

```
mysql> SET @poly =
    -> 'Polygon((30000 15000,
                               31000 15000,
                               31000 16000,
                               30000 16000,
                               30000 15000))';
mysql> EXPLAIN SELECT fid,AsText(g) FROM geom WHERE
    -> MBRContains(GeomFromText(@poly),g)\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: geom
         type: range
possible_keys: g
          key: g
      key_len: 32
          ref: NULL
         rows: 50
        Extra: Using where
1 row in set (0.00 sec)
```

Check what would happen without a spatial index:

```
mysql> SET @poly =
    -> 'Polygon((30000 15000,
                               31000 15000,
                               31000 16000,
                               30000 16000,
                               30000 15000))';
mysql> EXPLAIN SELECT fid,AsText(g) FROM g IGNORE INDEX (g) WHERE
```

```
    -> MBRContains(GeomFromText(@poly),g)\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: geom
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 32376
        Extra: Using where
1 row in set (0.00 sec)
```

Executing the `SELECT` statement without the spatial index yields the same result but causes the execution time to rise from 0.00 seconds to 0.46 seconds:

```

mysql> SET @poly =
    -> 'Polygon((30000 15000,
                               31000 15000,
                               31000 16000,
                               30000 16000,
                               30000 15000))';
mysql> SELECT fid,AsText(g) FROM geom IGNORE INDEX (g) WHERE
    -> MBRContains(GeomFromText(@poly),g);
+-----+-------------------------------------------------------------+
| fid | AsText(g)                                                   |
+-----+-------------------------------------------------------------+
|   1 | LINESTRING(30250.4 15129.2,30248.8 15138.4,30238.2 15136. ... |
|   2 | LINESTRING(30220.2 15122.8,30217.2 15137.8,30207.6 15136, ... |
|   3 | LINESTRING(30179 15114.4,30176.6 15129.4,30167 15128,3016 ... |
|   4 | LINESTRING(30155.2 15121.4,30140.4 15118.6,30142 15109,30 ... |
|   5 | LINESTRING(30192.4 15085,30177.6 15082.2,30179.2 15072.4, ... |
|   6 | LINESTRING(30244 15087,30229 15086.2,30229.4 15076.4,3024 ... |
|   7 | LINESTRING(30200.6 15059.4,30185.6 15058.6,30186 15048.8, ... |
|  10 | LINESTRING(30179.6 15017.8,30181 15002.8,30190.8 15003.6, ... |
|  11 | LINESTRING(30154.2 15000.4,30168.6 15004.8,30166 15014.2, ... |
|  13 | LINESTRING(30105 15065.8,30108.4 15050.8,30118 15053,3011 ... |
|  21 | LINESTRING(30350.4 15828.8,30350.6 15845,30333.8 15845,30 ... |
|  22 | LINESTRING(30350.6 15871.4,30350.6 15887.8,30334 15887.8, ... |
|  23 | LINESTRING(30350.6 15914.2,30350.6 15930.4,30334 15930.4, ... |
|  24 | LINESTRING(30290.2 15823,30290.2 15839.4,30273.4 15839.4, ... |
|  25 | LINESTRING(30291.4 15866.2,30291.6 15882.4,30274.8 15882. ... |
|  26 | LINESTRING(30291.6 15918.2,30291.6 15934.4,30275 15934.4, ... |
| 154 | LINESTRING(30276.2 15143.8,30261.4 15141,30263 15131.4,30 ... |
| 155 | LINESTRING(30269.8 15084,30269.4 15093.4,30258.6 15093,30 ... |
| 157 | LINESTRING(30128.2 15011,30113.2 15010.2,30113.6 15000.4, ... |
| 249 | LINESTRING(30337.8 15938.6,30337.8 15946.8,30320.4 15946. ... |
+-----+-------------------------------------------------------------+
20 rows in set (0.46 sec)
```

12.18.7 MySQL Conformance and Compatibility

MySQL does not yet implement the following GIS features:

- Additional Metadata Views

  OpenGIS specifications propose several additional metadata views. For example, a system view named `GEOMETRY_COLUMNS` contains a description of geometry columns, one row for each geometry column in the database.

- The OpenGIS function `Length()` on `LineString` and `MultiLineString` currently should be called in MySQL as `GLength()`

The problem is that there is an existing SQL function `Length()` that calculates the length of string values, and sometimes it is not possible to distinguish whether the function is called in a textual or spatial context. We need either to solve this somehow, or decide on another function name.

12.19 Precision Math

MySQL 5.7 provides support for precision math: numeric value handling that results in extremely accurate results and a high degree control over invalid values. Precision math is based on these two features:

- SQL modes that control how strict the server is about accepting or rejecting invalid data.
- The MySQL library for fixed-point arithmetic.

These features have several implications for numeric operations and provide a high degree of compliance with standard SQL:

- **Precise calculations**: For exact-value numbers, calculations do not introduce floating-point errors. Instead, exact precision is used. For example, MySQL treats a number such as `.0001` as an exact value rather than as an approximation, and summing it 10,000 times produces a result of exactly `1`, not a value that is merely “close” to 1.
- **Well-defined rounding behavior**: For exact-value numbers, the result of `ROUND()` depends on its argument, not on environmental factors such as how the underlying C library works.
- **Platform independence**: Operations on exact numeric values are the same across different platforms such as Windows and Unix.
- **Control over handling of invalid values**: Overflow and division by zero are detectable and can be treated as errors. For example, you can treat a value that is too large for a column as an error rather than having the value truncated to lie within the range of the column's data type. Similarly, you can treat division by zero as an error rather than as an operation that produces a result of `NULL`. The choice of which approach to take is determined by the setting of the server SQL mode.

The following discussion covers several aspects of how precision math works, including possible incompatibilities with older applications. At the end, some examples are given that demonstrate how MySQL 5.7 handles numeric operations precisely. For information about controlling the SQL mode, see Section 5.1.7, “Server SQL Modes”.

12.19.1 Types of Numeric Values

The scope of precision math for exact-value operations includes the exact-value data types (integer and `DECIMAL` types) and exact-value numeric literals. Approximate-value data types and numeric literals are handled as floating-point numbers.

Exact-value numeric literals have an integer part or fractional part, or both. They may be signed. Examples: `1`, `.2`, `3.4`, `-5`, `-6.78`, `+9.10`.

Approximate-value numeric literals are represented in scientific notation with a mantissa and exponent. Either or both parts may be signed. Examples: `1.2E3`, `1.2E-3`, `-1.2E3`, `-1.2E-3`.

Two numbers that look similar may be treated differently. For example, `2.34` is an exact-value (fixed-point) number, whereas `2.34E0` is an approximate-value (floating-point) number.

The `DECIMAL` data type is a fixed-point type and calculations are exact. In MySQL, the `DECIMAL` type has several synonyms: `NUMERIC`, `DEC`, `FIXED`. The integer types also are exact-value types.

The FLOAT and DOUBLE data types are floating-point types and calculations are approximate. In MySQL, types that are synonymous with FLOAT or DOUBLE are DOUBLE PRECISION and REAL.

12.19.2 DECIMAL Data Type Characteristics

This section discusses the characteristics of the DECIMAL data type (and its synonyms) in MySQL 5.7, with particular regard to the following topics:

- Maximum number of digits
- Storage format
- Storage requirements
- The nonstandard MySQL extension to the upper range of DECIMAL columns

Possible incompatibilities with applications that are written for older versions of MySQL (prior to 5.0.3) are noted throughout this section.

The declaration syntax for a DECIMAL column is DECIMAL(*M*,*D*). The ranges of values for the arguments in MySQL 5.7 are as follows:

- *M* is the maximum number of digits (the precision). It has a range of 1 to 65. (Older versions of MySQL permitted a range of 1 to 254.)
- *D* is the number of digits to the right of the decimal point (the scale). It has a range of 0 to 30 and must be no larger than *M*.

The maximum value of 65 for *M* means that calculations on DECIMAL values are accurate up to 65 digits. This limit of 65 digits of precision also applies to exact-value numeric literals, so the maximum range of such literals differs from before. (In older versions of MySQL, decimal values could have up to 254 digits. However, calculations were done using floating-point and thus were approximate, not exact.)

Values for DECIMAL columns in MySQL 5.7 are stored using a binary format that packs nine decimal digits into 4 bytes. The storage requirements for the integer and fractional parts of each value are determined separately. Each multiple of nine digits requires 4 bytes, and any remaining digits left over require some fraction of 4 bytes. The storage required for remaining digits is given by the following table.

Leftover Digits	Number of Bytes
0	0
1–2	1
3–4	2
5–6	3
7–9	4

For example, a DECIMAL(18,9) column has nine digits on either side of the decimal point, so the integer part and the fractional part each require 4 bytes. A DECIMAL(20,6) column has fourteen integer digits and six fractional digits. The integer digits require four bytes for nine of the digits and 3 bytes for the remaining five digits. The six fractional digits require 3 bytes.

Unlike some older versions of MySQL, DECIMAL columns in MySQL 5.7 do not store a leading + character or - character or leading 0 digits. If you insert +0003.1 into a DECIMAL(5,1) column, it is stored as 3.1. For negative numbers, a literal - character is not stored. Applications that rely on the older behavior must be modified to account for this change.

DECIMAL columns in MySQL 5.7 do not permit values larger than the range implied by the column definition. For example, a DECIMAL(3,0) column supports a range of -999 to 999. A DECIMAL(*M*,*D*) column permits at most *M* - *D* digits to the left of the decimal point. This is not compatible with applications relying on older versions of MySQL that permitted storing an extra digit in lieu of a + sign.

The SQL standard requires that the precision of NUMERIC(*M*,*D*) be *exactly* *M* digits. For DECIMAL(*M*,*D*), the standard requires a precision of at least *M* digits but permits more. In MySQL, DECIMAL(*M*,*D*) and NUMERIC(*M*,*D*) are the same, and both have a precision of exactly *M* digits.

For a full explanation of the internal format of DECIMAL values, see the file strings/decimal.c in a MySQL source distribution. The format is explained (with an example) in the decimal2bin() function.

For more detailed information about porting applications that rely on the old treatment of the DECIMAL data type, see the *MySQL 5.0 Reference Manual*.

12.19.3 Expression Handling

With precision math, exact-value numbers are used as given whenever possible. For example, numbers in comparisons are used exactly as given without a change in value. In strict SQL mode, for INSERT into a column with an exact data type (DECIMAL or integer), a number is inserted with its exact value if it is within the column range. When retrieved, the value should be the same as what was inserted. (If strict SQL mode is not enabled, truncation for INSERT is permissible.)

Handling of a numeric expression depends on what kind of values the expression contains:

- If any approximate values are present, the expression is approximate and is evaluated using floating-point arithmetic.

- If no approximate values are present, the expression contains only exact values. If any exact value contains a fractional part (a value following the decimal point), the expression is evaluated using DECIMAL exact arithmetic and has a precision of 65 digits. The term “exact” is subject to the limits of what can be represented in binary. For example, 1.0/3.0 can be approximated in decimal notation as .333..., but not written as an exact number, so (1.0/3.0)*3.0 does not evaluate to exactly 1.0.

- Otherwise, the expression contains only integer values. The expression is exact and is evaluated using integer arithmetic and has a precision the same as BIGINT (64 bits).

If a numeric expression contains any strings, they are converted to double-precision floating-point values and the expression is approximate.

Inserts into numeric columns are affected by the SQL mode, which is controlled by the sql_mode system variable. (See Section 5.1.7, “Server SQL Modes”.) The following discussion mentions strict mode (selected by the STRICT_ALL_TABLES or STRICT_TRANS_TABLES mode values) and ERROR_FOR_DIVISION_BY_ZERO. (As of MySQL 5.7.4, the effect of ERROR_FOR_DIVISION_BY_ZERO is included in strict mode.) To turn on all restrictions, you can simply use TRADITIONAL mode, which includes both strict mode values and ERROR_FOR_DIVISION_BY_ZERO:

```
mysql> SET sql_mode='TRADITIONAL';
```

If a number is inserted into an exact type column (DECIMAL or integer), it is inserted with its exact value if it is within the column range.

If the value has too many digits in the fractional part, rounding occurs and a warning is generated. Rounding is done as described in Section 12.19.4, “Rounding Behavior”.

If the value has too many digits in the integer part, it is too large and is handled as follows:

- If strict mode is not enabled, the value is truncated to the nearest legal value and a warning is generated.
- If strict mode is enabled, an overflow error occurs.

Underflow is not detected, so underflow handling is undefined.

For inserts of strings into numeric columns, conversion from string to number is handled as follows if the string has nonnumeric contents:

- A string that does not begin with a number cannot be used as a number and produces an error in strict mode, or a warning otherwise. This includes the empty string.
- A string that begins with a number can be converted, but the trailing nonnumeric portion is truncated. If the truncated portion contains anything other than spaces, this produces an error in strict mode, or a warning otherwise.

By default, division by zero produces a result of `NULL` and no warning. By setting the SQL mode appropriately, division by zero can be restricted and MySQL handles it differently.

As of MySQL 5.7.4, the effect of `ERROR_FOR_DIVISION_BY_ZERO` is included in strict mode. If strict mode is enabled, inserts and updates involving division by zero are prohibited, and an error occurs.

Before MYSQL 5.7.4, division by zero is controlled by the `ERROR_FOR_DIVISION_BY_ZERO` SQL mode in conjunction with strict mode. With `ERROR_FOR_DIVISION_BY_ZERO` enabled, MySQL handles division by zero as follows:

- If strict mode is not enabled, a warning occurs.
- If strict mode is enabled, inserts and updates involving division by zero are prohibited, and an error occurs.

In other words, inserts and updates involving expressions that perform division by zero can be treated as errors, but this requires `ERROR_FOR_DIVISION_BY_ZERO` in addition to strict mode.

Suppose that we have this statement:

```
INSERT INTO t SET i = 1/0;
```

This is what happens for combinations of strict and `ERROR_FOR_DIVISION_BY_ZERO` modes.

`sql_mode` **Value**	**Result**
`''` (Default)	No warning, no error; `i` is set to `NULL`.
strict	No warning, no error; `i` is set to `NULL`.
`ERROR_FOR_DIVISION_BY_ZERO`	Warning, no error; `i` is set to `NULL`.
strict,`ERROR_FOR_DIVISION_BY_ZERO`	Error condition; no row is inserted.

12.19.4 Rounding Behavior

This section discusses precision math rounding for the `ROUND()` function and for inserts into columns with exact-value types (`DECIMAL` and integer).

The `ROUND()` function rounds differently depending on whether its argument is exact or approximate:

- For exact-value numbers, ROUND() uses the “round half up” rule: A value with a fractional part of .5 or greater is rounded up to the next integer if positive or down to the next integer if negative. (In other words, it is rounded away from zero.) A value with a fractional part less than .5 is rounded down to the next integer if positive or up to the next integer if negative.
- For approximate-value numbers, the result depends on the C library. On many systems, this means that ROUND() uses the “round to nearest even” rule: A value with any fractional part is rounded to the nearest even integer.

The following example shows how rounding differs for exact and approximate values:

```
mysql> SELECT ROUND(2.5), ROUND(25E-1);
+------------+--------------+
| ROUND(2.5) | ROUND(25E-1) |
+------------+--------------+
| 3          |            2 |
+------------+--------------+
```

For inserts into a DECIMAL or integer column, the target is an exact data type, so rounding uses “round half away from zero,” regardless of whether the value to be inserted is exact or approximate:

```
mysql> CREATE TABLE t (d DECIMAL(10,0));
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO t VALUES(2.5),(2.5E0);
Query OK, 2 rows affected, 2 warnings (0.00 sec)
Records: 2  Duplicates: 0  Warnings: 2

mysql> SELECT d FROM t;
+------+
| d    |
+------+
| 3    |
| 3    |
+------+
```

12.19.5 Precision Math Examples

This section provides some examples that show precision math query results in MySQL 5.7. These examples demonstrate the principles described in Section 12.19.3, “Expression Handling”, and Section 12.19.4, “Rounding Behavior”.

Example 1. Numbers are used with their exact value as given when possible:

```
mysql> SELECT (.1 + .2) = .3;
+----------------+
| (.1 + .2) = .3 |
+----------------+
|              1 |
+----------------+
```

For floating-point values, results are inexact:

```
mysql> SELECT (.1E0 + .2E0) = .3E0;
+----------------------+
| (.1E0 + .2E0) = .3E0 |
+----------------------+
|                    0 |
+----------------------+
```

Another way to see the difference in exact and approximate value handling is to add a small number to a sum many times. Consider the following stored procedure, which adds `.0001` to a variable 1,000 times.

```
CREATE PROCEDURE p ()
BEGIN
  DECLARE i INT DEFAULT 0;
  DECLARE d DECIMAL(10,4) DEFAULT 0;
  DECLARE f FLOAT DEFAULT 0;
  WHILE i < 10000 DO
    SET d = d + .0001;
    SET f = f + .0001E0;
    SET i = i + 1;
  END WHILE;
  SELECT d, f;
END;
```

The sum for both `d` and `f` logically should be 1, but that is true only for the decimal calculation. The floating-point calculation introduces small errors:

```
+--------+------------------+
| d      | f                |
+--------+------------------+
| 1.0000 | 0.99999999999991 |
+--------+------------------+
```

Example 2. Multiplication is performed with the scale required by standard SQL. That is, for two numbers *`X1`* and *`X2`* that have scale *`S1`* and *`S2`*, the scale of the result is *`S1 + S2`*:

```
mysql> SELECT .01 * .01;
+-----------+
| .01 * .01 |
+-----------+
| 0.0001    |
+-----------+
```

Example 3. Rounding behavior for exact-value numbers is well-defined:

Rounding behavior (for example, with the `ROUND()` function) is independent of the implementation of the underlying C library, which means that results are consistent from platform to platform.

- Rounding for exact-value columns (`DECIMAL` and integer) and exact-valued numbers uses the “round half away from zero” rule. Values with a fractional part of .5 or greater are rounded away from zero to the nearest integer, as shown here:

  ```
  mysql> SELECT ROUND(2.5), ROUND(-2.5);
  +------------+-------------+
  | ROUND(2.5) | ROUND(-2.5) |
  +------------+-------------+
  | 3          | -3          |
  +------------+-------------+
  ```

- Rounding for floating-point values uses the C library, which on many systems uses the “round to nearest even” rule. Values with any fractional part on such systems are rounded to the nearest even integer:

  ```
  mysql> SELECT ROUND(2.5E0), ROUND(-2.5E0);
  +--------------+---------------+
  ```

```
| ROUND(2.5E0) | ROUND(-2.5E0) |
+--------------+---------------+
|            2 |            -2 |
+--------------+---------------+
```

Example 4. In strict mode, inserting a value that is out of range for a column causes an error, rather than truncation to a legal value.

When MySQL is not running in strict mode, truncation to a legal value occurs:

```
mysql> SET sql_mode='';
Query OK, 0 rows affected (0.00 sec)

mysql> CREATE TABLE t (i TINYINT);
Query OK, 0 rows affected (0.01 sec)

mysql> INSERT INTO t SET i = 128;
Query OK, 1 row affected, 1 warning (0.00 sec)

mysql> SELECT i FROM t;
+------+
| i    |
+------+
|  127 |
+------+
1 row in set (0.00 sec)
```

However, an error occurs if strict mode is in effect:

```
mysql> SET sql_mode='STRICT_ALL_TABLES';
Query OK, 0 rows affected (0.00 sec)

mysql> CREATE TABLE t (i TINYINT);
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO t SET i = 128;
ERROR 1264 (22003): Out of range value adjusted for column 'i' at row 1

mysql> SELECT i FROM t;
Empty set (0.00 sec)
```

Example 5: In strict mode and with `ERROR_FOR_DIVISION_BY_ZERO` set, division by zero causes an error, not a result of `NULL`.

In nonstrict mode, division by zero has a result of `NULL`:

```
mysql> SET sql_mode='';
Query OK, 0 rows affected (0.01 sec)

mysql> CREATE TABLE t (i TINYINT);
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO t SET i = 1 / 0;
Query OK, 1 row affected (0.00 sec)

mysql> SELECT i FROM t;
+------+
| i    |
+------+
| NULL |
+------+
1 row in set (0.03 sec)
```

However, division by zero is an error if the proper SQL modes are in effect:

```
mysql> SET sql_mode='STRICT_ALL_TABLES,ERROR_FOR_DIVISION_BY_ZERO';
Query OK, 0 rows affected (0.00 sec)

mysql> CREATE TABLE t (i TINYINT);
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO t SET i = 1 / 0;
ERROR 1365 (22012): Division by 0

mysql> SELECT i FROM t;
Empty set (0.01 sec)
```

Example 6. Exact-value literals are evaluated as exact values.

Prior to MySQL 5.0.3, exact-value and approximate-value literals both are evaluated as double-precision floating-point values:

```
mysql> SELECT VERSION();
+------------+
| VERSION()  |
+------------+
| 4.1.18-log |
+------------+
1 row in set (0.01 sec)

mysql> CREATE TABLE t SELECT 2.5 AS a, 25E-1 AS b;
Query OK, 1 row affected (0.07 sec)
Records: 1  Duplicates: 0  Warnings: 0

mysql> DESCRIBE t;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| a     | double(3,1) |      |     | 0.0     |       |
| b     | double      |      |     | 0       |       |
+-------+-------------+------+-----+---------+-------+
2 rows in set (0.04 sec)
```

As of MySQL 5.0.3, the approximate-value literal is evaluated using floating point, but the exact-value literal is handled as `DECIMAL`:

```
mysql> SELECT VERSION();
+-----------------+
| VERSION()       |
+-----------------+
| 5.1.6-alpha-log |
+-----------------+
1 row in set (0.11 sec)

mysql> CREATE TABLE t SELECT 2.5 AS a, 25E-1 AS b;
Query OK, 1 row affected (0.01 sec)
Records: 1  Duplicates: 0  Warnings: 0

mysql> DESCRIBE t;
+-------+-----------------------+------+-----+---------+-------+
| Field | Type                  | Null | Key | Default | Extra |
+-------+-----------------------+------+-----+---------+-------+
| a     | decimal(2,1) unsigned | NO   |     | 0.0     |       |
| b     | double                | NO   |     | 0       |       |
+-------+-----------------------+------+-----+---------+-------+
2 rows in set (0.01 sec)
```

Example 7. If the argument to an aggregate function is an exact numeric type, the result is also an exact numeric type, with a scale at least that of the argument.

Consider these statements:

```
mysql> CREATE TABLE t (i INT, d DECIMAL, f FLOAT);
mysql> INSERT INTO t VALUES(1,1,1);
mysql> CREATE TABLE y SELECT AVG(i), AVG(d), AVG(f) FROM t;
```

Before MySQL 5.0.3, the result is a double no matter the argument type:

```
mysql> DESCRIBE y;
+--------+--------------+------+-----+---------+-------+
| Field  | Type         | Null | Key | Default | Extra |
+--------+--------------+------+-----+---------+-------+
| AVG(i) | double(17,4) | YES  |     | NULL    |       |
| AVG(d) | double(17,4) | YES  |     | NULL    |       |
| AVG(f) | double       | YES  |     | NULL    |       |
+--------+--------------+------+-----+---------+-------+
```

As of MySQL 5.0.3, the result is a double only for the floating-point argument. For exact type arguments, the result is also an exact type:

```
mysql> DESCRIBE y;
+--------+---------------+------+-----+---------+-------+
| Field  | Type          | Null | Key | Default | Extra |
+--------+---------------+------+-----+---------+-------+
| AVG(i) | decimal(14,4) | YES  |     | NULL    |       |
| AVG(d) | decimal(14,4) | YES  |     | NULL    |       |
| AVG(f) | double        | YES  |     | NULL    |       |
+--------+---------------+------+-----+---------+-------+
```

The result is a double only for the floating-point argument. For exact type arguments, the result is also an exact type.

Chapter 13 SQL Statement Syntax

Table of Contents

This chapter describes the syntax for the SQL statements supported by MySQL.

13.1 Data Definition Statements

13.1.1 ALTER DATABASE Syntax

```
ALTER {DATABASE | SCHEMA} [db_name]
    alter_specification ...
ALTER {DATABASE | SCHEMA} db_name
    UPGRADE DATA DIRECTORY NAME

alter_specification:
    [DEFAULT] CHARACTER SET [=] charset_name
  | [DEFAULT] COLLATE [=] collation_name
```

ALTER DATABASE enables you to change the overall characteristics of a database. These characteristics are stored in the db.opt file in the database directory. To use ALTER DATABASE, you need the ALTER privilege on the database. ALTER SCHEMA is a synonym for ALTER DATABASE.

The database name can be omitted from the first syntax, in which case the statement applies to the default database.

National Language Characteristics

The CHARACTER SET clause changes the default database character set. The COLLATE clause changes the default database collation. Section 10.1, “Character Set Support”, discusses character set and collation names.

You can see what character sets and collations are available using, respectively, the SHOW CHARACTER SET and SHOW COLLATION statements. See Section 13.7.5.3, “SHOW CHARACTER SET Syntax”, and Section 13.7.5.4, “SHOW COLLATION Syntax”, for more information.

If you change the default character set or collation for a database, stored routines that use the database defaults must be dropped and recreated so that they use the new defaults. (In a stored routine, variables with character data types use the database defaults if the character set or collation are not specified explicitly. See Section 13.1.12, “CREATE PROCEDURE and CREATE FUNCTION Syntax”.)

Upgrading from Versions Older than MySQL 5.1

The syntax that includes the UPGRADE DATA DIRECTORY NAME clause updates the name of the directory associated with the database to use the encoding implemented in MySQL 5.1 for mapping database names to database directory names (see Section 9.2.3, “Mapping of Identifiers to File Names”). This clause is for use under these conditions:

- It is intended when upgrading MySQL to 5.1 or later from older versions.
- It is intended to update a database directory name to the current encoding format if the name contains special characters that need encoding.
- The statement is used by mysqlcheck (as invoked by mysql_upgrade).

For example, if a database in MySQL 5.0 has the name a-b-c, the name contains instances of the - (dash) character. In MySQL 5.0, the database directory is also named a-b-c, which is not necessarily safe for all file systems. In MySQL 5.1 and later, the same database name is encoded as a@002db@002dc to produce a file system-neutral directory name.

When a MySQL installation is upgraded to MySQL 5.1 or later from an older version,the server displays a name such as a-b-c (which is in the old format) as #mysql50#a-b-c, and you must refer to the name using the #mysql50# prefix. Use UPGRADE DATA DIRECTORY NAME in this case to explicitly tell the server to re-encode the database directory name to the current encoding format:

```
ALTER DATABASE `#mysql50#a-b-c` UPGRADE DATA DIRECTORY NAME;
```

After executing this statement, you can refer to the database as a-b-c without the special #mysql50# prefix.

13.1.2 ALTER EVENT Syntax

```
ALTER
    [DEFINER = { user | CURRENT_USER }]
    EVENT event_name
    [ON SCHEDULE schedule]
    [ON COMPLETION [NOT] PRESERVE]
    [RENAME TO new_event_name]
    [ENABLE | DISABLE | DISABLE ON SLAVE]
    [COMMENT 'comment']
    [DO event_body]
```

The ALTER EVENT statement changes one or more of the characteristics of an existing event without the need to drop and recreate it. The syntax for each of the DEFINER, ON SCHEDULE, ON COMPLETION, COMMENT, ENABLE / DISABLE, and DO clauses is exactly the same as when used with CREATE EVENT. (See Section 13.1.9, “CREATE EVENT Syntax”.)

Any user can alter an event defined on a database for which that user has the EVENT privilege. When a user executes a successful ALTER EVENT statement, that user becomes the definer for the affected event.

`ALTER EVENT` works only with an existing event:

```
mysql> ALTER EVENT no_such_event
     >     ON SCHEDULE
     >       EVERY '2:3' DAY_HOUR;
ERROR 1517 (HY000): Unknown event 'no_such_event'
```

In each of the following examples, assume that the event named `myevent` is defined as shown here:

```
CREATE EVENT myevent
    ON SCHEDULE
      EVERY 6 HOUR
    COMMENT 'A sample comment.'
    DO
      UPDATE myschema.mytable SET mycol = mycol + 1;
```

The following statement changes the schedule for `myevent` from once every six hours starting immediately to once every twelve hours, starting four hours from the time the statement is run:

```
ALTER EVENT myevent
    ON SCHEDULE
      EVERY 12 HOUR
    STARTS CURRENT_TIMESTAMP + INTERVAL 4 HOUR;
```

It is possible to change multiple characteristics of an event in a single statement. This example changes the SQL statement executed by `myevent` to one that deletes all records from `mytable`; it also changes the schedule for the event such that it executes once, one day after this `ALTER EVENT` statement is run.

```
ALTER EVENT myevent
    ON SCHEDULE
      AT CURRENT_TIMESTAMP + INTERVAL 1 DAY
    DO
      TRUNCATE TABLE myschema.mytable;
```

Specify the options in an `ALTER EVENT` statement only for those characteristics that you want to change; omitted options keep their existing values. This includes any default values for `CREATE EVENT` such as `ENABLE`.

To disable `myevent`, use this `ALTER EVENT` statement:

```
ALTER EVENT myevent
    DISABLE;
```

The `ON SCHEDULE` clause may use expressions involving built-in MySQL functions and user variables to obtain any of the *`timestamp`* or *`interval`* values which it contains. You cannot use stored routines or user-defined functions in such expressions, and you cannot use any table references; however, you can use `SELECT FROM DUAL`. This is true for both `ALTER EVENT` and `CREATE EVENT` statements. References to stored routines, user-defined functions, and tables in such cases are specifically not permitted, and fail with an error (see Bug #22830).

Although an `ALTER EVENT` statement that contains another `ALTER EVENT` statement in its `DO` clause appears to succeed, when the server attempts to execute the resulting scheduled event, the execution fails with an error.

To rename an event, use the `ALTER EVENT` statement's `RENAME TO` clause. This statement renames the event `myevent` to `yourevent`:

```
ALTER EVENT myevent
    RENAME TO yourevent;
```

You can also move an event to a different database using ALTER EVENT ... RENAME TO ... and *db_name.event_name* notation, as shown here:

```
ALTER EVENT olddb.myevent
    RENAME TO newdb.myevent;
```

To execute the previous statement, the user executing it must have the EVENT privilege on both the olddb and newdb databases.

Note

There is no RENAME EVENT statement.

The value DISABLE ON SLAVE is used on a replication slave instead of ENABLED or DISABLED to indicate an event that was created on the master and replicated to the slave, but that is not executed on the slave. Normally, DISABLE ON SLAVE is set automatically as required; however, there are some circumstances under which you may want or need to change it manually. See Section 16.4.1.11, "Replication of Invoked Features", for more information.

13.1.3 ALTER FUNCTION Syntax

```
ALTER FUNCTION func_name [characteristic ...]

characteristic:
    COMMENT 'string'
  | LANGUAGE SQL
  | { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }
  | SQL SECURITY { DEFINER | INVOKER }
```

This statement can be used to change the characteristics of a stored function. More than one change may be specified in an ALTER FUNCTION statement. However, you cannot change the parameters or body of a stored function using this statement; to make such changes, you must drop and re-create the function using DROP FUNCTION and CREATE FUNCTION.

You must have the ALTER ROUTINE privilege for the function. (That privilege is granted automatically to the function creator.) If binary logging is enabled, the ALTER FUNCTION statement might also require the SUPER privilege, as described in Section 18.7, "Binary Logging of Stored Programs".

13.1.4 ALTER PROCEDURE Syntax

```
ALTER PROCEDURE proc_name [characteristic ...]

characteristic:
    COMMENT 'string'
  | LANGUAGE SQL
  | { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }
  | SQL SECURITY { DEFINER | INVOKER }
```

This statement can be used to change the characteristics of a stored procedure. More than one change may be specified in an ALTER PROCEDURE statement. However, you cannot change the parameters or body of a stored procedure using this statement; to make such changes, you must drop and re-create the procedure using DROP PROCEDURE and CREATE PROCEDURE.

You must have the ALTER ROUTINE privilege for the procedure. By default, that privilege is granted automatically to the procedure creator. This behavior can be changed by disabling the automatic_sp_privileges system variable. See Section 18.2.2, “Stored Routines and MySQL Privileges”.

13.1.5 ALTER SERVER Syntax

```
ALTER SERVER  server_name
    OPTIONS (option [, option] ...)
```

Alters the server information for *server_name*, adjusting any of the options permitted in the CREATE SERVER statement. The corresponding fields in the mysql.servers table are updated accordingly. This statement requires the SUPER privilege.

For example, to update the USER option:

```
ALTER SERVER s OPTIONS (USER 'sally');
```

ALTER SERVER does not cause an automatic commit.

In MySQL 5.7, ALTER SERVER is not written to the binary log, regardless of the logging format that is in use.

In MySQL 5.7.1, gtid_next must be set to AUTOMATIC before issuing this statement. This restriction does not apply in MySQL 5.7.2 or later. (Bug #16062608, Bug #16715809, Bug #69045)

13.1.6 ALTER TABLE Syntax

```
ALTER [IGNORE] TABLE tbl_name
    [alter_specification [, alter_specification] ...]
    [partition_options]


algorithm_option:
    ALGORITHM [=] {DEFAULT|INPLACE|COPY}

lock_option:
    LOCK [=] {DEFAULT|NONE|SHARED|EXCLUSIVE}

alter_specification:
    table_options
  | ADD [COLUMN] col_name column_definition
        [FIRST | AFTER col_name ]
  | ADD [COLUMN] (col_name column_definition,...)
  | ADD {INDEX|KEY} [index_name]
        [index_type] (index_col_name,...) [index_option] ...
  | ADD [CONSTRAINT [symbol]] PRIMARY KEY
        [index_type] (index_col_name,...) [index_option] ...
  | ADD [CONSTRAINT [symbol]]
        UNIQUE [INDEX|KEY] [index_name]
        [index_type] (index_col_name,...) [index_option] ...
  | ADD FULLTEXT [INDEX|KEY] [index_name]
        (index_col_name,...) [index_option] ...
  | ADD SPATIAL [INDEX|KEY] [index_name]
        (index_col_name,...) [index_option] ...
  | ADD [CONSTRAINT [symbol]]
        FOREIGN KEY [index_name] (index_col_name,...)
        reference_definition
  | ALGORITHM [=] {DEFAULT|INPLACE|COPY}
  | ALTER [COLUMN] col_name {SET DEFAULT literal | DROP DEFAULT}
  | CHANGE [COLUMN] old_col_name new_col_name column_definition
        [FIRST|AFTER col_name]
  | LOCK [=] {DEFAULT|NONE|SHARED|EXCLUSIVE}
  | MODIFY [COLUMN] col_name column_definition
```

```
        [FIRST | AFTER col_name]
  | DROP [COLUMN] col_name
  | DROP PRIMARY KEY
  | DROP {INDEX|KEY} index_name
  | DROP FOREIGN KEY fk_symbol
  | DISABLE KEYS
  | ENABLE KEYS
  | RENAME [TO|AS] new_tbl_name
  | RENAME {INDEX|KEY} old_index_name TO new_index_name
  | ORDER BY col_name [, col_name] ...
  | CONVERT TO CHARACTER SET charset_name [COLLATE collation_name]
  | [DEFAULT] CHARACTER SET [=] charset_name [COLLATE [=] collation_name]
  | DISCARD TABLESPACE
  | IMPORT TABLESPACE
  | FORCE
  | ADD PARTITION (partition_definition)
  | DROP PARTITION partition_names
  | DISCARD PARTITION {partition_names | ALL} TABLESPACE
  | IMPORT PARTITION {partition_names | ALL} TABLESPACE
  | TRUNCATE PARTITION {partition_names | ALL}
  | COALESCE PARTITION number
  | REORGANIZE PARTITION partition_names INTO (partition_definitions)
  | EXCHANGE PARTITION partition_name WITH TABLE tbl_name
  | ANALYZE PARTITION {partition_names | ALL}
  | CHECK PARTITION {partition_names | ALL}
  | OPTIMIZE PARTITION {partition_names | ALL}
  | REBUILD PARTITION {partition_names | ALL}
  | REPAIR PARTITION {partition_names | ALL}
  | REMOVE PARTITIONING

index_col_name:
    col_name [(length)] [ASC | DESC]

index_type:
    USING {BTREE | HASH}

index_option:
    KEY_BLOCK_SIZE [=] value
  | index_type
  | WITH PARSER parser_name
  | COMMENT 'string'

table_options:
    table_option [[,] table_option] ...  (see CREATE TABLE options)

partition_options:
    (see CREATE TABLE options)
```

ALTER TABLE changes the structure of a table. For example, you can add or delete columns, create or destroy indexes, change the type of existing columns, or rename columns or the table itself. You can also change characteristics such as the storage engine used for the table or the table comment.

Partitioning-related clauses for ALTER TABLE can be used with partitioned tables for repartitioning, for adding, dropping, discarding, importing, merging, and splitting partitions, and for performing partitioning maintenance. It is possible for an ALTER TABLE statement to contain a PARTITION BY or REMOVE PARTITIONING clause in an addition to other alter specifications, but the PARTITION BY or REMOVE PARTITIONING clause must be specified last after any other specifications. The ADD PARTITION, DROP PARTITION, DISCARD PARTITION, IMPORT PARTITION, COALESCE PARTITION, REORGANIZE PARTITION, ANALYZE PARTITION, CHECK PARTITION, and REPAIR PARTITION options cannot be combined with other alter specifications in a single ALTER TABLE, since the options just listed act on individual partitions. For more information, see Section 13.1.6.1, "ALTER TABLE Partition Operations".

Following the table name, specify the alterations to be made. If none are given, ALTER TABLE does nothing.

The syntax for many of the permissible alterations is similar to clauses of the CREATE TABLE statement. See Section 13.1.14, “CREATE TABLE Syntax”, for more information.

Some operations may result in warnings if attempted on a table for which the storage engine does not support the operation. These warnings can be displayed with SHOW WARNINGS. See Section 13.7.5.39, “SHOW WARNINGS Syntax”.

For information on troubleshooting ALTER TABLE, see Section C.5.7.1, “Problems with ALTER TABLE”.

Storage, Performance, and Concurrency Considerations

In most cases, ALTER TABLE makes a temporary copy of the original table. MySQL waits for other operations that are modifying the table, then proceeds. It incorporates the alteration into the copy, deletes the original table, and renames the new one. While ALTER TABLE is executing, the original table is readable by other sessions (with the exception noted shortly). Updates and writes to the table that begin after the ALTER TABLE operation begins are stalled until the new table is ready, then are automatically redirected to the new table without any failed updates. The temporary copy of the original table is created in the database directory of the new table. This can differ from the database directory of the original table for ALTER TABLE operations that rename the table to a different database.

The exception referred to earlier is that ALTER TABLE blocks reads (not just writes) at the point where it is ready to install a new version of the table .frm file, discard the old file, and clear outdated table structures from the table and table definition caches. At this point, it must acquire an exclusive lock. To do so, it waits for current readers to finish, and blocks new reads (and writes).

For MyISAM tables, you can speed up index re-creation (the slowest part of the alteration process) by setting the myisam_sort_buffer_size system variable to a high value.

For some operations, an in-place ALTER TABLE is possible that does not require a temporary table:

- For ALTER TABLE *tbl_name* RENAME TO *new_tbl_name* without any other options, MySQL simply renames any files that correspond to the table *tbl_name* without making a copy. (You can also use the RENAME TABLE statement to rename tables. See Section 13.1.26, “RENAME TABLE Syntax”.) Any privileges granted specifically for the renamed table are not migrated to the new name. They must be changed manually.

- Alterations that modify only table metadata and not table data are immediate because the server only needs to alter the table .frm file, not touch table contents. The following changes are fast alterations that can be made this way:

  - Renaming a column.

  - Changing the default value of a column.

  - Changing the definition of an ENUM or SET column by adding new enumeration or set members to the *end* of the list of valid member values, as long as the storage size of the data type does not change. For example, adding a member to a SET column that has 8 members changes the required storage per value from 1 byte to 2 bytes; this will require a table copy. Adding members in the middle of the list causes renumbering of existing members, which requires a table copy.

- ALTER TABLE with DISCARD ... PARTITION ... TABLESPACE or IMPORT ... PARTITION ... TABLESPACE do not create any temporary tables or temporary partition files.

  ALTER TABLE with ADD PARTITION, DROP PARTITION, COALESCE PARTITION, REBUILD PARTITION, or REORGANIZE PARTITION does not create any temporary tables (except when used with NDB tables); however, these operations can and do create temporary partition files.

ADD or DROP operations for RANGE or LIST partitions are immediate operations or nearly so. ADD or COALESCE operations for HASH or KEY partitions copy data between all partitions, unless LINEAR HASH or LINEAR KEY was used; this is effectively the same as creating a new table, although the ADD or COALESCE operation is performed partition by partition. REORGANIZE operations copy only changed partitions and do not touch unchanged ones.

- Renaming an index.
- Adding or dropping an index, for InnoDB.

You can force an ALTER TABLE operation that would otherwise not require a table copy to use the temporary table method (as supported in MySQL 5.0) by setting the old_alter_table system variable to ON, or specifying ALGORITHM=COPY as one of the *alter_specification* clauses. If there is a conflict between the old_alter_table setting and an ALGORITHM clause with a value other than DEFAULT, the ALGORITHM clause takes precedence. (ALGORITHM = DEFAULT is the same a specifying no ALGORITHM clause at all.)

Specifying ALGORITHM=INPLACE makes the operation use the in-place technique for clauses and storage engines that support it, and fail with an error otherwise, thus avoiding a lengthy table copy if you try altering a table that uses a different storage engine than you expect. See Section 14.2.11, “InnoDB and Online DDL” for information about online DDL for InnoDB tables.

You can control the level of concurrent reading and writing of the table while it is being altered, using the LOCK clause. Specifying a non-default value for this clause lets you require a certain amount of concurrent access or exclusivity during the alter operation, and halts the operation if the requested degree of locking is not available. The parameters for the LOCK clause are:

-
  ```
  LOCK = DEFAULT
  ```

  Maximum level of concurrency for the given ALGORITHM clause (if any) and ALTER TABLE operation: Permit concurrent reads and writes if supported. If not, permit concurrent reads if supported. If not, enforce exclusive access.

-
  ```
  LOCK = NONE
  ```

  If supported, permit concurrent reads and writes. Otherwise, return an error message.

-
  ```
  LOCK = SHARED
  ```

  If supported, permit concurrent reads but block writes. Note that writes will be blocked even if concurrent writes are supported by the storage engine for the given ALGORITHM clause (if any) and ALTER TABLE operation. If concurrent reads are not supported, return an error message.

-
  ```
  LOCK = EXCLUSIVE
  ```

  Enforce exclusive access. This will be done even if concurrent reads/writes are supported by the storage engine for the given ALGORITHM clause (if any) and ALTER TABLE operation.

You can also use ALTER TABLE *tbl_name* FORCE to perform a “null” alter operation that rebuilds the table. Previously the FORCE option was recognized but ignored. As of MySQL 5.6.17, online DDL support is provided for the FORCE option. For more information, see Section 14.2.11.1, “Overview of Online DDL”.

Usage Notes

- To use ALTER TABLE, you need ALTER, CREATE, and INSERT privileges for the table. Renaming a table requires ALTER and DROP on the old table, ALTER, CREATE, and INSERT on the new table.
- IGNORE is a MySQL extension to standard SQL. It controls how ALTER TABLE works if there are duplicates on unique keys in the new table or if warnings occur when strict mode is enabled. If IGNORE is not specified, the copy is aborted and rolled back if duplicate-key errors occur. If IGNORE is specified, only one row is used of rows with duplicates on a unique key. The other conflicting rows are deleted. Incorrect values are truncated to the closest matching acceptable value.

  As of MySQL 5.7.2, the IGNORE clause is supported with the CHECK PARTITION and REPAIR PARTITION options (Bug #16900947). See Section 13.1.6.1, "ALTER TABLE Partition Operations".

  As of MySQL 5.7.4, the IGNORE clause for ALTER TABLE is removed and its use produces an error.
- *table_option* signifies a table option of the kind that can be used in the CREATE TABLE statement, such as ENGINE, AUTO_INCREMENT, AVG_ROW_LENGTH, or MAX_ROWS. For a list of all table options and descriptions of each, see Section 13.1.14, "CREATE TABLE Syntax". However, ALTER TABLE ignores the DATA DIRECTORY and INDEX DIRECTORY table options.

  For example, to convert a table to be an InnoDB table, use this statement:

  ```
  ALTER TABLE t1 ENGINE = InnoDB;
  ```

  See Section 14.2.6.4, "Converting Tables from MyISAM to InnoDB" for considerations when switching tables to the InnoDB storage engine.

  When you specify an ENGINE clause, ALTER TABLE rebuilds the table. This is true even if the table already has the specified storage engine.

  Running ALTER TABLE *tbl_name* ENGINE=INNODB on an existing InnoDB table performs a "null" ALTER TABLE operation, which can be used to defragment an InnoDB table, as described in Section 14.2.10.4, "Defragmenting a Table". Running ALTER TABLE *tbl_name* FORCE on an InnoDB table performs the same function.

  As of MySQL 5.7.4, both ALTER TABLE *tbl_name* ENGINE=INNODB and ALTER TABLE *tbl_name* FORCE use online DDL (ALGORITHM=COPY). For more information, see Section 14.2.11.1, "Overview of Online DDL".

  The outcome of attempting to change a table's storage engine is affected by whether the desired storage engine is available and the setting of the NO_ENGINE_SUBSTITUTION SQL mode, as described in Section 5.1.7, "Server SQL Modes".

  To prevent inadvertent loss of data, ALTER TABLE cannot be used to change the storage engine of a table to MERGE or BLACKHOLE.

  To change the value of the AUTO_INCREMENT counter to be used for new rows, do this:

  ```
  ALTER TABLE t2 AUTO_INCREMENT = value;
  ```

  You cannot reset the counter to a value less than or equal to the value that is currently in use. For both InnoDB and MyISAM, if the value is less than or equal to the maximum value currently in the AUTO_INCREMENT column, the value is reset to the current maximum AUTO_INCREMENT column value plus one.

- You can issue multiple ADD, ALTER, DROP, and CHANGE clauses in a single ALTER TABLE statement, separated by commas. This is a MySQL extension to standard SQL, which permits only one of each clause per ALTER TABLE statement. For example, to drop multiple columns in a single statement, do this:

  ```
  ALTER TABLE t2 DROP COLUMN c, DROP COLUMN d;
  ```

- CHANGE *col_name*, DROP *col_name*, and DROP INDEX are MySQL extensions to standard SQL.

- The word COLUMN is optional and can be omitted.

- *column_definition* clauses use the same syntax for ADD and CHANGE as for CREATE TABLE. See Section 13.1.14, "CREATE TABLE Syntax".

- You can rename a column using a CHANGE *old_col_name new_col_name column_definition* clause. To do so, specify the old and new column names and the definition that the column currently has. For example, to rename an INTEGER column from a to b, you can do this:

  ```
  ALTER TABLE t1 CHANGE a b INTEGER;
  ```

  To change a column's type but not the name, CHANGE syntax still requires an old and new column name, even if they are the same. For example:

  ```
  ALTER TABLE t1 CHANGE b b BIGINT NOT NULL;
  ```

  You can also use MODIFY to change a column's type without renaming it:

  ```
  ALTER TABLE t1 MODIFY b BIGINT NOT NULL;
  ```

  MODIFY is an extension to ALTER TABLE for Oracle compatibility.

  When you use CHANGE or MODIFY, *column_definition* must include the data type and all attributes that should apply to the new column, other than index attributes such as PRIMARY KEY or UNIQUE. Attributes present in the original definition but not specified for the new definition are not carried forward. Suppose that a column col1 is defined as INT UNSIGNED DEFAULT 1 COMMENT 'my column' and you modify the column as follows:

  ```
  ALTER TABLE t1 MODIFY col1 BIGINT;
  ```

  The resulting column will be defined as BIGINT, but will not include the attributes UNSIGNED DEFAULT 1 COMMENT 'my column'. To retain them, the statement should be:

  ```
  ALTER TABLE t1 MODIFY col1 BIGINT UNSIGNED DEFAULT 1 COMMENT 'my column';
  ```

- When you change a data type using CHANGE or MODIFY, MySQL tries to convert existing column values to the new type as well as possible.

  Warning

  This conversion may result in alteration of data. For example, if you shorten a string column, values may be truncated. To prevent the operation from succeeding if conversions to the new data type would result in loss of data, enable strict SQL mode before using ALTER TABLE (see Section 5.1.7, "Server SQL Modes").

- To add a column at a specific position within a table row, use FIRST or AFTER *col_name*. The default is to add the column last. You can also use FIRST and AFTER in CHANGE or MODIFY operations to reorder columns within a table.

- ALTER ... SET DEFAULT or ALTER ... DROP DEFAULT specify a new default value for a column or remove the old default value, respectively. If the old default is removed and the column can be NULL, the new default is NULL. If the column cannot be NULL, MySQL assigns a default value as described in Section 11.5, "Data Type Default Values".

- DROP INDEX removes an index. This is a MySQL extension to standard SQL. See Section 13.1.20, "DROP INDEX Syntax". If you are unsure of the index name, use SHOW INDEX FROM *tbl_name*.

- If columns are dropped from a table, the columns are also removed from any index of which they are a part. If all columns that make up an index are dropped, the index is dropped as well. If you use CHANGE or MODIFY to shorten a column for which an index exists on the column, and the resulting column length is less than the index length, MySQL shortens the index automatically.

- If a table contains only one column, the column cannot be dropped. If what you intend is to remove the table, use DROP TABLE instead.

- DROP PRIMARY KEY drops the primary key. If there is no primary key, an error occurs. For information about the performance characteristics of primary keys, especially for InnoDB tables, see Section 8.3.2, "Using Primary Keys".

  If you add a UNIQUE INDEX or PRIMARY KEY to a table, MySQL stores it before any nonunique index to permit detection of duplicate keys as early as possible.

- Some storage engines permit you to specify an index type when creating an index. The syntax for the *index_type* specifier is USING *type_name*. For details about USING, see Section 13.1.11, "CREATE INDEX Syntax". The preferred position is after the column list. Support for use of the option before the column list will be removed in a future MySQL release.

  index_option values specify additional options for an index. USING is one such option. For details about permissible *index_option* values, see Section 13.1.11, "CREATE INDEX Syntax".

- RENAME INDEX *old_index_name* TO *new_index_name* renames an index. This is a MySQL extension to standard SQL. The content of the table remains unchanged. *old_index_name* must be the name of an existing index in the table that is not dropped by the same ALTER TABLE statement. new_index_name is the new index name, which cannot duplicate the name of an index in the resulting table after changes have been applied. Neither index name can be PRIMARY.

- After an ALTER TABLE statement, it may be necessary to run ANALYZE TABLE to update index cardinality information. See Section 13.7.5.21, "SHOW INDEX Syntax".

- ORDER BY enables you to create the new table with the rows in a specific order. Note that the table does not remain in this order after inserts and deletes. This option is useful primarily when you know that you are mostly to query the rows in a certain order most of the time. By using this option after major changes to the table, you might be able to get higher performance. In some cases, it might make sorting easier for MySQL if the table is in order by the column that you want to order it by later.

  ORDER BY syntax permits one or more column names to be specified for sorting, each of which optionally can be followed by ASC or DESC to indicate ascending or descending sort order, respectively. The default is ascending order. Only column names are permitted as sort criteria; arbitrary expressions are not permitted. This clause should be given last after any other clauses.

ORDER BY does not make sense for InnoDB tables because InnoDB always orders table rows according to the clustered index.

Note

When used on a partitioned table, ALTER TABLE ... ORDER BY orders rows within each partition only.

- If you use ALTER TABLE on a MyISAM table, all nonunique indexes are created in a separate batch (as for REPAIR TABLE). This should make ALTER TABLE much faster when you have many indexes.

  For MyISAM tables, key updating can be controlled explicitly. Use ALTER TABLE ... DISABLE KEYS to tell MySQL to stop updating nonunique indexes. Then use ALTER TABLE ... ENABLE KEYS to re-create missing indexes. MyISAM does this with a special algorithm that is much faster than inserting keys one by one, so disabling keys before performing bulk insert operations should give a considerable speedup. Using ALTER TABLE ... DISABLE KEYS requires the INDEX privilege in addition to the privileges mentioned earlier.

  While the nonunique indexes are disabled, they are ignored for statements such as SELECT and EXPLAIN that otherwise would use them.

- In MySQL 5.7, the server prohibits changes to foreign key columns that have the potential to cause loss of referential integrity. It also prohibits changes to the data type of such columns that may be unsafe. For example, changing VARCHAR(20) to VARCHAR(30) is permitted, but changing it to VARCHAR(1024) is not because that alters the number of length bytes required to store individual values. A workaround is to use ALTER TABLE ... DROP FOREIGN KEY before changing the column definition and ALTER TABLE ... ADD FOREIGN KEY afterward.

- The FOREIGN KEY and REFERENCES clauses are supported by the InnoDB storage engine, which implements ADD [CONSTRAINT [*symbol*]] FOREIGN KEY [*index_name*] (...) REFERENCES ... (...). See Section 14.2.6.6, “InnoDB and FOREIGN KEY Constraints”. For other storage engines, the clauses are parsed but ignored. The CHECK clause is parsed but ignored by all storage engines. See Section 13.1.14, “CREATE TABLE Syntax”. The reason for accepting but ignoring syntax clauses is for compatibility, to make it easier to port code from other SQL servers, and to run applications that create tables with references. See Section 1.8.2, “MySQL Differences from Standard SQL”.

  For ALTER TABLE, unlike CREATE TABLE, ADD FOREIGN KEY ignores *index_name* if given and uses an automatically generated foreign key name. As a workaround, include the CONSTRAINT clause to specify the foreign key name:

```
ADD CONSTRAINT name FOREIGN KEY (....) ...
```

Important

The inline REFERENCES specifications where the references are defined as part of the column specification are silently ignored by InnoDB. InnoDB only accepts REFERENCES clauses defined as part of a separate FOREIGN KEY specification.

Note

Partitioned InnoDB tables do not support foreign keys. See Section 17.6.2, “Partitioning Limitations Relating to Storage Engines”, for more information.

- InnoDB supports the use of ALTER TABLE to drop foreign keys:

```
ALTER TABLE tbl_name DROP FOREIGN KEY fk_symbol;
```

For more information, see Section 14.2.6.6, “InnoDB and FOREIGN KEY Constraints”.

- Prior to MySQL 5.6.6, adding and dropping a foreign key in the same ALTER TABLE statement may be problematic in some cases and is therefore unsupported. Separate statements should be used for each operation. As of MySQL 5.6.6, adding and dropping a foreign key in the same ALTER TABLE statement is supported for ALTER TABLE ... ALGORITHM=INPLACE but remains unsupported for ALTER TABLE ... ALGORITHM=COPY.

- For an InnoDB table that is created with its own tablespace in an .ibd file, that file can be discarded and imported. To discard the .ibd file, use this statement:

```
ALTER TABLE tbl_name DISCARD TABLESPACE;
```

This deletes the current .ibd file, so be sure that you have a backup first. Attempting to modify the table contents while the tablespace file is discarded results in an error. You can perform the DDL operations listed in Section 14.2.11, “InnoDB and Online DDL” while the tablespace file is discarded.

To import the backup .ibd file back into the table, copy it into the database directory, and then issue this statement:

```
ALTER TABLE tbl_name IMPORT TABLESPACE;
```

The tablespace file need not necessarily have been created on the server into which it is imported later. In MySQL 5.7, importing a tablespace file from another server works if the both servers have GA (General Availablility) status and their versions are within the same series. Otherwise, the file must have been created on the server into which it is imported.

Note

The ALTER TABLE ... IMPORT TABLESPACE feature does not enforce foreign key constraints on imported data.

See Section 14.2.5.2, “InnoDB File-Per-Table Mode”.

- To change the table default character set and all character columns (CHAR, VARCHAR, TEXT) to a new character set, use a statement like this:

```
ALTER TABLE tbl_name CONVERT TO CHARACTER SET charset_name;
```

For a column that has a data type of VARCHAR or one of the TEXT types, CONVERT TO CHARACTER SET will change the data type as necessary to ensure that the new column is long enough to store as many characters as the original column. For example, a TEXT column has two length bytes, which store the byte-length of values in the column, up to a maximum of 65,535. For a latin1 TEXT column, each character requires a single byte, so the column can store up to 65,535 characters. If the column is converted to utf8, each character might require up to three bytes, for a maximum possible length of 3 × 65,535 = 196,605 bytes. That length will not fit in a TEXT column's length bytes, so MySQL will convert the data type to MEDIUMTEXT, which is the smallest string type for which the length bytes can record a value of 196,605. Similarly, a VARCHAR column might be converted to MEDIUMTEXT.

To avoid data type changes of the type just described, do not use CONVERT TO CHARACTER SET. Instead, use MODIFY to change individual columns. For example:

```
ALTER TABLE t MODIFY latin1_text_col TEXT CHARACTER SET utf8;
ALTER TABLE t MODIFY latin1_varchar_col VARCHAR(M) CHARACTER SET utf8;
```

If you specify `CONVERT TO CHARACTER SET binary`, the `CHAR`, `VARCHAR`, and `TEXT` columns are converted to their corresponding binary string types (`BINARY`, `VARBINARY`, `BLOB`). This means that the columns no longer will have a character set and a subsequent `CONVERT TO` operation will not apply to them.

If you specify `CONVERT TO CHARACTER SET` without a collation, the default collation for the character set is used. If this collation is inappropriate for the intended table use (for example, if it would change from a case-sensitive collation to a case-insensitive collation), specify a collation explicitly.

If *`charset_name`* is `DEFAULT`, the database character set is used.

> **Warning**
>
> The `CONVERT TO` operation converts column values between the character sets. This is *not* what you want if you have a column in one character set (like `latin1`) but the stored values actually use some other, incompatible character set (like `utf8`). In this case, you have to do the following for each such column:
>
> ```
> ALTER TABLE t1 CHANGE c1 c1 BLOB;
> ALTER TABLE t1 CHANGE c1 c1 TEXT CHARACTER SET utf8;
> ```
>
> The reason this works is that there is no conversion when you convert to or from `BLOB` columns.

To change only the *default* character set for a table, use this statement:

```
ALTER TABLE tbl_name DEFAULT CHARACTER SET charset_name;
```

The word `DEFAULT` is optional. The default character set is the character set that is used if you do not specify the character set for columns that you add to a table later (for example, with `ALTER TABLE ... ADD column`).

With the `mysql_info()` C API function, you can find out how many rows were copied by `ALTER TABLE`, and (when `IGNORE` is used) how many rows were deleted due to duplication of unique key values. See Section 21.8.7.36, “`mysql_info()`”.

13.1.6.1 ALTER TABLE Partition Operations

Partitioning-related clauses for `ALTER TABLE` can be used with partitioned tables for repartitioning, for adding, dropping, discarding, importing, merging, and splitting partitions, and for performing partitioning maintenance.

- Simply using a *`partition_options`* clause with `ALTER TABLE` on a partitioned table repartitions the table according to the partitioning scheme defined by the *`partition_options`*. This clause always begins with `PARTITION BY`, and follows the same syntax and other rules as apply to the *`partition_options`* clause for `CREATE TABLE` (see Section 13.1.14, “`CREATE TABLE` Syntax”, for more detailed information), and can also be used to partition an existing table that is not already partitioned. For example, consider a (nonpartitioned) table defined as shown here:

```
CREATE TABLE t1 (
```

```
    id INT,
    year_col INT
);
```

This table can be partitioned by HASH, using the id column as the partitioning key, into 8 partitions by means of this statement:

```
ALTER TABLE t1
    PARTITION BY HASH(id)
    PARTITIONS 8;
```

MySQL 5.7.1 and later supports an ALGORITHM option with [SUB]PARTITION BY [LINEAR] KEY. ALGORITHM=1 causes the server to use the same key-hashing functions as MySQL 5.1 when computing the placement of rows in partitions; ALGORITHM=2 means that the server employs the key-hashing functions implemented and used by default for new KEY partitioned tables in MySQL 5.5 and later. (Partitioned tables created with the key-hashing functions employed in MySQL 5.5 and later cannot be used by a MySQL 5.1 server.) Not specifying the option has the same effect as using ALGORITHM=2. This option is intended for use chiefly when upgrading or downgrading [LINEAR] KEY partitioned tables between MySQL 5.1 and later MySQL versions, or for creating tables partitioned by KEY or LINEAR KEY on a MySQL 5.5 or later server which can be used on a MySQL 5.1 server.

To upgrade a KEY partitioned table that was created in MySQL 5.1, first execute SHOW CREATE TABLE and note the the exact columns and number of partitions shown. Now execute an ALTER TABLE statement using exactly the same column list and number of partitions as in the CREATE TABLE statement, while adding ALGORITHM=2 immediately following the PARTITION BY keywords. (You should also include the LINEAR keyword if it was used for the original table definition.) An example from a session in the mysql client is shown here:

```
mysql> SHOW CREATE TABLE p\G
*************************** 1. row ***************************
       Table: p
Create Table: CREATE TABLE `p` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `cd` datetime NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1
/*!50100 PARTITION BY LINEAR KEY (id)
PARTITIONS 32 */
1 row in set (0.00 sec)

mysql> ALTER TABLE p PARTITION BY LINEAR KEY ALGORITHM=2 (id) PARTITIONS 32;
Query OK, 0 rows affected (5.34 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> SHOW CREATE TABLE p\G
*************************** 1. row ***************************
       Table: p
Create Table: CREATE TABLE `p` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `cd` datetime NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1
/*!50100 PARTITION BY LINEAR KEY (id)
PARTITIONS 32 */
1 row in set (0.00 sec)
```

Downgrading a table created using the default key-hashing used in MySQL 5.5 and later to enable its use by a MySQL 5.1 server is similar, except in this case you should use ALGORITHM=1 to force the table's partitions to be rebuilt using the MySQL 5.1 key-hashing functions. It is recommended that you not do this except when necessary for compatibility with a MySQL 5.1 server, as the improved KEY hashing functions used by default in MySQL 5.5 and later provide fixes for a number of issues found in the older implementation.

Note

A table upgraded by means of ALTER TABLE ... PARTITION BY ALGORITHM=2 [LINEAR] KEY ... can no longer be used by a MySQL 5.1 server. (Such a table would need to be downgraded with ALTER TABLE ... PARTITION BY ALGORITHM=1 [LINEAR] KEY ... before it could be used again by a MySQL 5.1 server.)

The table that results from using an ALTER TABLE ... PARTITION BY statement must follow the same rules as one created using CREATE TABLE ... PARTITION BY. This includes the rules governing the relationship between any unique keys (including any primary key) that the table might have, and the column or columns used in the partitioning expression, as discussed in Section 17.6.1, “Partitioning Keys, Primary Keys, and Unique Keys”. The CREATE TABLE ... PARTITION BY rules for specifying the number of partitions also apply to ALTER TABLE ... PARTITION BY.

The *partition_definition* clause for ALTER TABLE ADD PARTITION supports the same options as the clause of the same name for the CREATE TABLE statement. (See Section 13.1.14, “CREATE TABLE Syntax”, for the syntax and description.) Suppose that you have the partitioned table created as shown here:

```
CREATE TABLE t1 (
    id INT,
    year_col INT
)
PARTITION BY RANGE (year_col) (
    PARTITION p0 VALUES LESS THAN (1991),
    PARTITION p1 VALUES LESS THAN (1995),
    PARTITION p2 VALUES LESS THAN (1999)
);
```

You can add a new partition p3 to this table for storing values less than 2002 as follows:

```
ALTER TABLE t1 ADD PARTITION (PARTITION p3 VALUES LESS THAN (2002));
```

DROP PARTITION can be used to drop one or more RANGE or LIST partitions. This statement cannot be used with HASH or KEY partitions; instead, use COALESCE PARTITION (see below). Any data that was stored in the dropped partitions named in the *partition_names* list is discarded. For example, given the table t1 defined previously, you can drop the partitions named p0 and p1 as shown here:

```
ALTER TABLE t1 DROP PARTITION p0, p1;
```

ADD PARTITION and DROP PARTITION do not currently support IF [NOT] EXISTS.

In MySQL 5.7.4, DISCARD PARTITION ... TABLESPACE and IMPORT PARTITION ... TABLESPACE options extend the Transportable Tablespace feature to individual InnoDB table partitions. Each InnoDB table partition has its own tablespace file (.idb file). The Transportable Tablespace feature makes it easy to copy the tablespaces from a running MySQL server instance to another running instance, or to perform a restore on the same instance. Both options take a comma-separated list of one or more partition names. For example:

```
ALTER TABLE t1 DISCARD PARTITION p2, p3 TABLESPACE;
ALTER TABLE t1 IMPORT PARTITION p2, p3 TABLESPACE;
```

When running DISCARD PARTITION ... TABLESPACE and IMPORT PARTITION ... TABLESPACE on subpartitioned tables, both partition and subpartition table names are allowed. When a partition name is specified, subpartitions of that partition are included.

As of MySQL 5.7.4, the Transportable Tablespace feature also supports copying or restoring partitioned InnoDB tables (all partitions at once). For addition information about the Transportable Tablespace feature, see Section 14.2.5.5, “Copying Tablespaces to Another Server (Transportable Tablespaces)”. For usage examples, see Transportable Tablespace Examples.

Renames of partitioned table are supported. You can rename individual partitions indirectly using ALTER TABLE ... REORGANIZE PARTITION; however, this operation makes a copy of the partition's data..

In MySQL 5.7, it is possible to delete rows from selected partitions using the TRUNCATE PARTITION option. This option takes a comma-separated list of one or more partition names. For example, consider the table t1 as defined here:

```
CREATE TABLE t1 (
    id INT,
    year_col INT
)
PARTITION BY RANGE (year_col) (
    PARTITION p0 VALUES LESS THAN (1991),
    PARTITION p1 VALUES LESS THAN (1995),
    PARTITION p2 VALUES LESS THAN (1999),
    PARTITION p3 VALUES LESS THAN (2003),
    PARTITION p4 VALUES LESS THAN (2007)
);
```

To delete all rows from partition p0, you can use the following statement:

```
ALTER TABLE t1 TRUNCATE PARTITION p0;
```

The statement just shown has the same effect as the following DELETE statement:

```
DELETE FROM t1 WHERE year_col < 1991;
```

When truncating multiple partitions, the partitions do not have to be contiguous: This can greatly simplify delete operations on partitioned tables that would otherwise require very complex WHERE conditions if done with DELETE statements. For example, this statement deletes all rows from partitions p1 and p3:

```
ALTER TABLE t1 TRUNCATE PARTITION p1, p3;
```

An equivalent DELETE statement is shown here:

```
DELETE FROM t1 WHERE
    (year_col >= 1991 AND year_col < 1995)
    OR
    (year_col >= 2003 AND year_col < 2007);
```

You can also use the ALL keyword in place of the list of partition names; in this case, the statement acts on all partitions in the table.

`TRUNCATE PARTITION` merely deletes rows; it does not alter the definition of the table itself, or of any of its partitions.

Note

Prior to MySQL 5.7.2, `TRUNCATE PARTITION` did not work with subpartitions (Bug #14028340, Bug #65184).

You can verify that the rows were dropped by checking the `INFORMATION_SCHEMA.PARTITIONS` table, using a query such as this one:

```
SELECT PARTITION_NAME, TABLE_ROWS
    FROM INFORMATION_SCHEMA.PARTITIONS
    WHERE TABLE_NAME = 't1';
```

`TRUNCATE PARTITION` is supported only for partitioned tables that use the `MyISAM`, `InnoDB`, or `MEMORY` storage engine. It also works on `BLACKHOLE` tables (but has no effect). It is not supported for `ARCHIVE` tables.

`COALESCE PARTITION` can be used with a table that is partitioned by `HASH` or `KEY` to reduce the number of partitions by *number*. Suppose that you have created table `t2` using the following definition:

```
CREATE TABLE t2 (
    name VARCHAR (30),
    started DATE
)
PARTITION BY HASH( YEAR(started) )
PARTITIONS 6;
```

You can reduce the number of partitions used by `t2` from 6 to 4 using the following statement:

```
ALTER TABLE t2 COALESCE PARTITION 2;
```

The data contained in the last *number* partitions will be merged into the remaining partitions. In this case, partitions 4 and 5 will be merged into the first 4 partitions (the partitions numbered 0, 1, 2, and 3).

To change some but not all the partitions used by a partitioned table, you can use `REORGANIZE PARTITION`. This statement can be used in several ways:

- To merge a set of partitions into a single partition. This can be done by naming several partitions in the *partition_names* list and supplying a single definition for *partition_definition*.
- To split an existing partition into several partitions. You can accomplish this by naming a single partition for *partition_names* and providing multiple *partition_definitions*.
- To change the ranges for a subset of partitions defined using `VALUES LESS THAN` or the value lists for a subset of partitions defined using `VALUES IN`.

Note

For partitions that have not been explicitly named, MySQL automatically provides the default names `p0`, `p1`, `p2`, and so on. The same is true with regard to subpartitions.

For more detailed information about and examples of ALTER TABLE ... REORGANIZE PARTITION statements, see Section 17.3.1, "Management of RANGE and LIST Partitions".

- It is also possible in MySQL 5.7 to exchange a table partition or subpartition with a table using ALTER TABLE *pt* EXCHANGE PARTITION *p* WITH TABLE *nt*, where *pt* is the partitioned table and *p* is the partition or subpartition of *pt* to be exchanged with unpartitioned table *nt*, provided that the following statements are true:

  1. Table *nt* is not itself partitioned.
  2. Table *nt* is not a temporary table.
  3. The structures of tables *pt* and *nt* are otherwise identical.
  4. There are no rows in *nt* that lie outside the boundaries of the partition definition for *p*.
  5. Table nt contains no foreign key references, and no other table has any foreign keys that refer to nt.

  Executing ALTER TABLE ... EXCHANGE PARTITION does not invoke any triggers on either the partitioned table or the table to be exchanged.

  Any AUTO_INCREMENT columns in the table to be exchanged with a partition are reset.

  The IGNORE keyword has no effect when used with ALTER TABLE ... EXCHANGE PARTITION.

  For more information about and examples of ALTER TABLE ... EXCHANGE PARTITION, see Section 17.3.3, "Exchanging Partitions and Subpartitions with Tables".

- Several additional options provide partition maintenance and repair functionality analogous to that implemented for nonpartitioned tables by statements such as CHECK TABLE and REPAIR TABLE (which are also supported for partitioned tables; see Section 13.7.2, "Table Maintenance Statements" for more information). These include ANALYZE PARTITION, CHECK PARTITION, OPTIMIZE PARTITION, REBUILD PARTITION, and REPAIR PARTITION. Each of these options takes a *partition_names* clause consisting of one or more names of partitions, separated by commas. The partitions must already exist in the table to be altered. You can also use the ALL keyword in place of *partition_names*, in which case the statement acts on all partitions in the table. For more information and examples, see Section 17.3.4, "Maintenance of Partitions".

  Some MySQL storage engines, such as InnoDB, do not support per-partition optimization. For a partitioned table using such a storage engine, ALTER TABLE ... OPTIMIZE PARTITION causes the entire table to rebuilt and analyzed, and an appropriate warning to be issued. (Bug #11751825, Bug #42822)

  To work around this problem, use the statements ALTER TABLE ... REBUILD PARTITION and ALTER TABLE ... ANALYZE PARTITION instead.

  The ANALYZE PARTITION, CHECK PARTITION, OPTIMIZE PARTITION, and REPAIR PARTITION options are not permitted for tables which are not partitioned.

- REMOVE PARTITIONING enables you to remove a table's partitioning without otherwise affecting the table or its data. This option can be combined with other ALTER TABLE options such as those used to add, drop, or rename drop columns or indexes.

- Using the ENGINE option with ALTER TABLE changes the storage engine used by the table without affecting the partitioning.

In MySQL 5.7, when `ALTER TABLE ... EXCHANGE PARTITION` or `ALTER TABLE ... TRUNCATE PARTITION` is run against a partitioned table that uses `MyISAM` (or another storage engine that makes use of table-level locking), only those partitions that are actually read from are locked. (This does not apply to partitioned tables using a storage enginethat employs row-level locking, such as `InnoDB`.) See Section 17.6.4, “Partitioning and Locking”.

It is possible for an `ALTER TABLE` statement to contain a `PARTITION BY` or `REMOVE PARTITIONING` clause in an addition to other alter specifications, but the `PARTITION BY` or `REMOVE PARTITIONING` clause must be specified last after any other specifications.

The `ADD PARTITION`, `DROP PARTITION`, `COALESCE PARTITION`, `REORGANIZE PARTITION`, `ANALYZE PARTITION`, `CHECK PARTITION`, and `REPAIR PARTITION` options cannot be combined with other alter specifications in a single `ALTER TABLE`, since the options just listed act on individual partitions. For more information, see Section 13.1.6.1, “`ALTER TABLE` Partition Operations”.

Only a single instance of any one of the following options can be used in a given `ALTER TABLE` statement: `PARTITION BY`, `ADD PARTITION`, `DROP PARTITION`, `TRUNCATE PARTITION`, `EXCHANGE PARTITION`, `REORGANIZE PARTITION`, or `COALESCE PARTITION`, `ANALYZE PARTITION`, `CHECK PARTITION`, `OPTIMIZE PARTITION`, `REBUILD PARTITION`, `REMOVE PARTITIONING`.

For example, the following two statements are invalid:

```
ALTER TABLE t1 ANALYZE PARTITION p1, ANALYZE PARTITION p2;

ALTER TABLE t1 ANALYZE PARTITION p1, CHECK PARTITION p2;
```

In the first case, you can analyze partitions `p1` and `p2` of table `t1` concurrently using a single statement with a single `ANALYZE PARTITION` option that lists both of the partitions to be analyzed, like this:

```
ALTER TABLE t1 ANALYZE PARTITION p1, p2;
```

In the second case, it is not possible to perform `ANALYZE` and `CHECK` operations on different partitions of the same table concurrently. Instead, you must issue two separate statements, like this:

```
ALTER TABLE t1 ANALYZE PARTITION p1;
ALTER TABLE t1 CHECK PARTITION p2;
```

Prior to MySQL 5.7.2, `ANALYZE`, `CHECK`, `OPTIMIZE`, `REBUILD`, `REPAIR`, and `TRUNCATE` operations were not supported for subpartitions (Bug #14028340, Bug #65184).

`CHECK PARTITION` and `REPAIR PARTITION` operations fail when the partition to be checked or repaired contains any duplicate key errors. MySQL 5.7.2 and later provides alternative behavior that can be invoked using `ALTER IGNORE TABLE` with the corresponding options (Bug #16900947), which causes the statement to behave as follows:

- `ALTER IGNORE TABLE ... REPAIR PARTITION` removes from the partition all rows that cannot be moved due to the presence of duplicate keys.

- `ALTER IGNORE TABLE ... CHECK PARTITION` writes out the contents of all columns in the partitioning expression for each row in the partition in which a duplicate key violation is found.

For more information about these statements, see Section 17.3.4, “Maintenance of Partitions”.

13.1.6.2 `ALTER TABLE` Examples

Begin with a table `t1` that is created as shown here:

```
CREATE TABLE t1 (a INTEGER,b CHAR(10));
```

To rename the table from t1 to t2:

```
ALTER TABLE t1 RENAME t2;
```

To change column a from INTEGER to TINYINT NOT NULL (leaving the name the same), and to change column b from CHAR(10) to CHAR(20) as well as renaming it from b to c:

```
ALTER TABLE t2 MODIFY a TINYINT NOT NULL, CHANGE b c CHAR(20);
```

To add a new TIMESTAMP column named d:

```
ALTER TABLE t2 ADD d TIMESTAMP;
```

To add an index on column d and a UNIQUE index on column a:

```
ALTER TABLE t2 ADD INDEX (d), ADD UNIQUE (a);
```

To remove column c:

```
ALTER TABLE t2 DROP COLUMN c;
```

To add a new AUTO_INCREMENT integer column named c:

```
ALTER TABLE t2 ADD c INT UNSIGNED NOT NULL AUTO_INCREMENT,
  ADD PRIMARY KEY (c);
```

We indexed c (as a PRIMARY KEY) because AUTO_INCREMENT columns must be indexed, and we declare c as NOT NULL because primary key columns cannot be NULL.

When you add an AUTO_INCREMENT column, column values are filled in with sequence numbers automatically. For MyISAM tables, you can set the first sequence number by executing SET INSERT_ID=*value* before ALTER TABLE or by using the AUTO_INCREMENT=*value* table option. See Section 5.1.4, “Server System Variables”.

With MyISAM tables, if you do not change the AUTO_INCREMENT column, the sequence number is not affected. If you drop an AUTO_INCREMENT column and then add another AUTO_INCREMENT column, the numbers are resequenced beginning with 1.

When replication is used, adding an AUTO_INCREMENT column to a table might not produce the same ordering of the rows on the slave and the master. This occurs because the order in which the rows are numbered depends on the specific storage engine used for the table and the order in which the rows were inserted. If it is important to have the same order on the master and slave, the rows must be ordered before assigning an AUTO_INCREMENT number. Assuming that you want to add an AUTO_INCREMENT column to the table t1, the following statements produce a new table t2 identical to t1 but with an AUTO_INCREMENT column:

```
CREATE TABLE t2 (id INT AUTO_INCREMENT PRIMARY KEY)
SELECT * FROM t1 ORDER BY col1, col2;
```

This assumes that the table t1 has columns col1 and col2.

This set of statements will also produce a new table t2 identical to t1, with the addition of an AUTO_INCREMENT column:

```
CREATE TABLE t2 LIKE t1;
ALTER TABLE t2 ADD id INT AUTO_INCREMENT PRIMARY KEY;
INSERT INTO t2 SELECT * FROM t1 ORDER BY col1, col2;
```

⚠ **Important**

To guarantee the same ordering on both master and slave, *all* columns of t1 must be referenced in the ORDER BY clause.

Regardless of the method used to create and populate the copy having the AUTO_INCREMENT column, the final step is to drop the original table and then rename the copy:

```
DROP t1;
ALTER TABLE t2 RENAME t1;
```

13.1.7 ALTER VIEW Syntax

```
ALTER
    [ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}]
    [DEFINER = { user | CURRENT_USER }]
    [SQL SECURITY { DEFINER | INVOKER }]
    VIEW view_name [(column_list)]
    AS select_statement
    [WITH [CASCADED | LOCAL] CHECK OPTION]
```

This statement changes the definition of a view, which must exist. The syntax is similar to that for CREATE VIEW and the effect is the same as for CREATE OR REPLACE VIEW. See Section 13.1.16, “CREATE VIEW Syntax”. This statement requires the CREATE VIEW and DROP privileges for the view, and some privilege for each column referred to in the SELECT statement. ALTER VIEW is permitted only to the definer or users with the SUPER privilege.

13.1.8 CREATE DATABASE Syntax

```
CREATE {DATABASE | SCHEMA} [IF NOT EXISTS] db_name
    [create_specification] ...

create_specification:
    [DEFAULT] CHARACTER SET [=] charset_name
  | [DEFAULT] COLLATE [=] collation_name
```

CREATE DATABASE creates a database with the given name. To use this statement, you need the CREATE privilege for the database. CREATE SCHEMA is a synonym for CREATE DATABASE.

An error occurs if the database exists and you did not specify IF NOT EXISTS.

In MySQL 5.7, CREATE DATABASE is not permitted within a session that has an active LOCK TABLES statement.

create_specification options specify database characteristics. Database characteristics are stored in the db.opt file in the database directory. The CHARACTER SET clause specifies the default database character set. The COLLATE clause specifies the default database collation. Section 10.1, “Character Set Support”, discusses character set and collation names.

A database in MySQL is implemented as a directory containing files that correspond to tables in the database. Because there are no tables in a database when it is initially created, the CREATE DATABASE statement creates only a directory under the MySQL data directory and the db.opt file. Rules for permissible database names are given in Section 9.2, “Schema Object Names”. If a database name contains special characters, the name for the database directory contains encoded versions of those characters as described in Section 9.2.3, “Mapping of Identifiers to File Names”.

If you manually create a directory under the data directory (for example, with mkdir), the server considers it a database directory and it shows up in the output of SHOW DATABASES.

You can also use the mysqladmin program to create databases. See Section 4.5.2, “mysqladmin — Client for Administering a MySQL Server”.

13.1.9 CREATE EVENT Syntax

```
CREATE
    [DEFINER = { user | CURRENT_USER }]
    EVENT
    [IF NOT EXISTS]
    event_name
    ON SCHEDULE schedule
    [ON COMPLETION [NOT] PRESERVE]
    [ENABLE | DISABLE | DISABLE ON SLAVE]
    [COMMENT 'comment']
    DO event_body;

schedule:
    AT timestamp [+ INTERVAL interval] ...
  | EVERY interval
    [STARTS timestamp [+ INTERVAL interval] ...]
    [ENDS timestamp [+ INTERVAL interval] ...]

interval:
    quantity {YEAR | QUARTER | MONTH | DAY | HOUR | MINUTE |
              WEEK | SECOND | YEAR_MONTH | DAY_HOUR | DAY_MINUTE |
              DAY_SECOND | HOUR_MINUTE | HOUR_SECOND | MINUTE_SECOND}
```

This statement creates and schedules a new event. The event will not run unless the Event Scheduler is enabled. For information about checking Event Scheduler status and enabling it if necessary, see Section 18.4.2, “Event Scheduler Configuration”.

CREATE EVENT requires the EVENT privilege for the schema in which the event is to be created. It might also require the SUPER privilege, depending on the DEFINER value, as described later in this section.

The minimum requirements for a valid CREATE EVENT statement are as follows:

- The keywords CREATE EVENT plus an event name, which uniquely identifies the event in a database schema.
- An ON SCHEDULE clause, which determines when and how often the event executes.
- A DO clause, which contains the SQL statement to be executed by an event.

This is an example of a minimal CREATE EVENT statement:

```
CREATE EVENT myevent
    ON SCHEDULE AT CURRENT_TIMESTAMP + INTERVAL 1 HOUR
    DO
      UPDATE myschema.mytable SET mycol = mycol + 1;
```

The previous statement creates an event named `myevent`. This event executes once—one hour following its creation—by running an SQL statement that increments the value of the `myschema.mytable` table's `mycol` column by 1.

The *`event_name`* must be a valid MySQL identifier with a maximum length of 64 characters. Event names are not case sensitive, so you cannot have two events named `myevent` and `MyEvent` in the same schema. In general, the rules governing event names are the same as those for names of stored routines. See Section 9.2, “Schema Object Names”.

An event is associated with a schema. If no schema is indicated as part of *`event_name`*, the default (current) schema is assumed. To create an event in a specific schema, qualify the event name with a schema using *`schema_name`*`.`*`event_name`* syntax.

The `DEFINER` clause specifies the MySQL account to be used when checking access privileges at event execution time. If a *`user`* value is given, it should be a MySQL account specified as `'`*`user_name`*`'@'`*`host_name`*`'` (the same format used in the `GRANT` statement), `CURRENT_USER`, or `CURRENT_USER()`. The default `DEFINER` value is the user who executes the `CREATE EVENT` statement. This is the same as specifying `DEFINER = CURRENT_USER` explicitly.

If you specify the `DEFINER` clause, these rules determine the valid `DEFINER` user values:

- If you do not have the `SUPER` privilege, the only permitted *`user`* value is your own account, either specified literally or by using `CURRENT_USER`. You cannot set the definer to some other account.

- If you have the `SUPER` privilege, you can specify any syntactically valid account name. If the account does not actually exist, a warning is generated.

- Although it is possible to create an event with a nonexistent `DEFINER` account, an error occurs at event execution time if the account does not exist.

For more information about event security, see Section 18.6, “Access Control for Stored Programs and Views”.

Within an event, the `CURRENT_USER()` function returns the account used to check privileges at event execution time, which is the `DEFINER` user. For information about user auditing within events, see Section 6.3.14, “SQL-Based MySQL Account Activity Auditing”.

`IF NOT EXISTS` has the same meaning for `CREATE EVENT` as for `CREATE TABLE`: If an event named *`event_name`* already exists in the same schema, no action is taken, and no error results. (However, a warning is generated in such cases.)

The `ON SCHEDULE` clause determines when, how often, and for how long the *`event_body`* defined for the event repeats. This clause takes one of two forms:

- `AT` *`timestamp`* is used for a one-time event. It specifies that the event executes one time only at the date and time given by *`timestamp`*, which must include both the date and time, or must be an expression that resolves to a datetime value. You may use a value of either the `DATETIME` or `TIMESTAMP` type for this purpose. If the date is in the past, a warning occurs, as shown here:

  ```
  mysql> SELECT NOW();
  +---------------------+
  ```

```
| NOW()               |
+---------------------+
| 2006-02-10 23:59:01 |
+---------------------+
1 row in set (0.04 sec)

mysql> CREATE EVENT e_totals
    ->     ON SCHEDULE AT '2006-02-10 23:59:00'
    ->     DO INSERT INTO test.totals VALUES (NOW());
Query OK, 0 rows affected, 1 warning (0.00 sec)

mysql> SHOW WARNINGS\G
*************************** 1. row ***************************
  Level: Note
   Code: 1588
Message: Event execution time is in the past and ON COMPLETION NOT
         PRESERVE is set. The event was dropped immediately after
         creation.
```

`CREATE EVENT` statements which are themselves invalid—for whatever reason—fail with an error.

You may use `CURRENT_TIMESTAMP` to specify the current date and time. In such a case, the event acts as soon as it is created.

To create an event which occurs at some point in the future relative to the current date and time—such as that expressed by the phrase “three weeks from now”—you can use the optional clause `+ INTERVAL` *`interval`*. The *`interval`* portion consists of two parts, a quantity and a unit of time, and follows the same syntax rules that govern intervals used in the `DATE_ADD()` function (see Section 12.7, “Date and Time Functions”. The units keywords are also the same, except that you cannot use any units involving microseconds when defining an event. With some interval types, complex time units may be used. For example, “two minutes and ten seconds” can be expressed as `+ INTERVAL '2:10' MINUTE_SECOND`.

You can also combine intervals. For example, `AT CURRENT_TIMESTAMP + INTERVAL 3 WEEK + INTERVAL 2 DAY` is equivalent to “three weeks and two days from now”. Each portion of such a clause must begin with `+ INTERVAL`.

- To repeat actions at a regular interval, use an `EVERY` clause. The `EVERY` keyword is followed by an *`interval`* as described in the previous discussion of the `AT` keyword. (`+ INTERVAL` is *not* used with `EVERY`.) For example, `EVERY 6 WEEK` means “every six weeks”.

  Although `+ INTERVAL` clauses are not permitted in an `EVERY` clause, you can use the same complex time units permitted in a `+ INTERVAL`.

  An `EVERY` clause may contain an optional `STARTS` clause. `STARTS` is followed by a *`timestamp`* value that indicates when the action should begin repeating, and may also use `+ INTERVAL` *`interval`* to specify an amount of time “from now”. For example, `EVERY 3 MONTH STARTS CURRENT_TIMESTAMP + INTERVAL 1 WEEK` means “every three months, beginning one week from now”. Similarly, you can express “every two weeks, beginning six hours and fifteen minutes from now” as `EVERY 2 WEEK STARTS CURRENT_TIMESTAMP + INTERVAL '6:15' HOUR_MINUTE`. Not specifying `STARTS` is the same as using `STARTS CURRENT_TIMESTAMP`—that is, the action specified for the event begins repeating immediately upon creation of the event.

  An `EVERY` clause may contain an optional `ENDS` clause. The `ENDS` keyword is followed by a *`timestamp`* value that tells MySQL when the event should stop repeating. You may also use `+ INTERVAL` *`interval`* with `ENDS`; for instance, `EVERY 12 HOUR STARTS CURRENT_TIMESTAMP + INTERVAL 30 MINUTE ENDS CURRENT_TIMESTAMP + INTERVAL 4 WEEK` is equivalent to “every twelve hours, beginning thirty minutes from now, and ending four weeks from now”. Not using `ENDS` means that the event continues executing indefinitely.

`ENDS` supports the same syntax for complex time units as `STARTS` does.

You may use `STARTS`, `ENDS`, both, or neither in an `EVERY` clause.

If a repeating event does not terminate within its scheduling interval, the result may be multiple instances of the event executing simultaneously. If this is undesirable, you should institute a mechanism to prevent simultaneous instances. For example, you could use the `GET_LOCK()` function, or row or table locking.

The `ON SCHEDULE` clause may use expressions involving built-in MySQL functions and user variables to obtain any of the *`timestamp`* or *`interval`* values which it contains. You may not use stored functions or user-defined functions in such expressions, nor may you use any table references; however, you may use `SELECT FROM DUAL`. This is true for both `CREATE EVENT` and `ALTER EVENT` statements. References to stored functions, user-defined functions, and tables in such cases are specifically not permitted, and fail with an error (see Bug #22830).

Times in the `ON SCHEDULE` clause are interpreted using the current session `time_zone` value. This becomes the event time zone; that is, the time zone that is used for event scheduling and is in effect within the event as it executes. These times are converted to UTC and stored along with the event time zone in the `mysql.event` table. This enables event execution to proceed as defined regardless of any subsequent changes to the server time zone or daylight saving time effects. For additional information about representation of event times, see Section 18.4.4, “Event Metadata”. See also Section 13.7.5.17, “`SHOW EVENTS` Syntax”, and Section 19.7, “The `INFORMATION_SCHEMA EVENTS` Table”.

Normally, once an event has expired, it is immediately dropped. You can override this behavior by specifying `ON COMPLETION PRESERVE`. Using `ON COMPLETION NOT PRESERVE` merely makes the default nonpersistent behavior explicit.

You can create an event but prevent it from being active using the `DISABLE` keyword. Alternatively, you can use `ENABLE` to make explicit the default status, which is active. This is most useful in conjunction with `ALTER EVENT` (see Section 13.1.2, “`ALTER EVENT` Syntax”).

A third value may also appear in place of `ENABLED` or `DISABLED`; `DISABLE ON SLAVE` is set for the status of an event on a replication slave to indicate that the event was created on the master and replicated to the slave, but is not executed on the slave. See Section 16.4.1.11, “Replication of Invoked Features”.

You may supply a comment for an event using a `COMMENT` clause. *`comment`* may be any string of up to 64 characters that you wish to use for describing the event. The comment text, being a string literal, must be surrounded by quotation marks.

The `DO` clause specifies an action carried by the event, and consists of an SQL statement. Nearly any valid MySQL statement that can be used in a stored routine can also be used as the action statement for a scheduled event. (See Section E.1, “Restrictions on Stored Programs”.) For example, the following event `e_hourly` deletes all rows from the `sessions` table once per hour, where this table is part of the `site_activity` schema:

```
CREATE EVENT e_hourly
    ON SCHEDULE
      EVERY 1 HOUR
    COMMENT 'Clears out sessions table each hour.'
    DO
      DELETE FROM site_activity.sessions;
```

MySQL stores the `sql_mode` system variable setting in effect when an event is created or altered, and always executes the event with this setting in force, *regardless of the current server SQL mode when the event begins executing*.

A CREATE EVENT statement that contains an ALTER EVENT statement in its DO clause appears to succeed; however, when the server attempts to execute the resulting scheduled event, the execution fails with an error.

Note

Statements such as SELECT or SHOW that merely return a result set have no effect when used in an event; the output from these is not sent to the MySQL Monitor, nor is it stored anywhere. However, you can use statements such as SELECT ... INTO and INSERT INTO ... SELECT that store a result. (See the next example in this section for an instance of the latter.)

The schema to which an event belongs is the default schema for table references in the DO clause. Any references to tables in other schemas must be qualified with the proper schema name.

As with stored routines, you can use compound-statement syntax in the DO clause by using the BEGIN and END keywords, as shown here:

```
delimiter |

CREATE EVENT e_daily
    ON SCHEDULE
      EVERY 1 DAY
    COMMENT 'Saves total number of sessions then clears the table each day'
    DO
      BEGIN
        INSERT INTO site_activity.totals (time, total)
          SELECT CURRENT_TIMESTAMP, COUNT(*)
            FROM site_activity.sessions;
        DELETE FROM site_activity.sessions;
      END |

delimiter ;
```

This example uses the delimiter command to change the statement delimiter. See Section 18.1, “Defining Stored Programs”.

More complex compound statements, such as those used in stored routines, are possible in an event. This example uses local variables, an error handler, and a flow control construct:

```
delimiter |

CREATE EVENT e
    ON SCHEDULE
      EVERY 5 SECOND
    DO
      BEGIN
        DECLARE v INTEGER;
        DECLARE CONTINUE HANDLER FOR SQLEXCEPTION BEGIN END;

        SET v = 0;

        WHILE v < 5 DO
          INSERT INTO t1 VALUES (0);
          UPDATE t2 SET s1 = s1 + 1;
          SET v = v + 1;
        END WHILE;
    END |

delimiter ;
```

There is no way to pass parameters directly to or from events; however, it is possible to invoke a stored routine with parameters within an event:

```
CREATE EVENT e_call_myproc
    ON SCHEDULE
      AT CURRENT_TIMESTAMP + INTERVAL 1 DAY
    DO CALL myproc(5, 27);
```

If an event's definer has the SUPER privilege, the event can read and write global variables. As granting this privilege entails a potential for abuse, extreme care must be taken in doing so.

Generally, any statements that are valid in stored routines may be used for action statements executed by events. For more information about statements permissible within stored routines, see Section 18.2.1, "Stored Routine Syntax". You can create an event as part of a stored routine, but an event cannot be created by another event.

13.1.10 CREATE FUNCTION Syntax

The CREATE FUNCTION statement is used to create stored functions and user-defined functions (UDFs):

- For information about creating stored functions, see Section 13.1.12, "CREATE PROCEDURE and CREATE FUNCTION Syntax".
- For information about creating user-defined functions, see Section 13.7.3.1, "CREATE FUNCTION Syntax for User-Defined Functions".

13.1.11 CREATE INDEX Syntax

```
CREATE [UNIQUE|FULLTEXT|SPATIAL] INDEX index_name
    [index_type]
    ON tbl_name (index_col_name,...)
    [index_option]
    [algorithm_option | lock_option] ...

index_col_name:
    col_name [(length)] [ASC | DESC]

index_type:
    USING {BTREE | HASH}

index_option:
    KEY_BLOCK_SIZE [=] value
  | index_type
  | WITH PARSER parser_name
  | COMMENT 'string'

algorithm_option:
    ALGORITHM [=] {DEFAULT|INPLACE|COPY}

lock_option:
    LOCK [=] {DEFAULT|NONE|SHARED|EXCLUSIVE}
```

CREATE INDEX is mapped to an ALTER TABLE statement to create indexes. See Section 13.1.6, "ALTER TABLE Syntax". CREATE INDEX cannot be used to create a PRIMARY KEY; use ALTER TABLE instead. For more information about indexes, see Section 8.3.1, "How MySQL Uses Indexes".

Normally, you create all indexes on a table at the time the table itself is created with CREATE TABLE. See Section 13.1.14, "CREATE TABLE Syntax". This guideline is especially important for InnoDB tables, where the primary key determines the physical layout of rows in the data file. CREATE INDEX enables you to add indexes to existing tables.

A column list of the form (col1,col2,...) creates a multiple-column index. Index key values are formed by concatenating the values of the given columns.

Indexes can be created that use only the leading part of column values, using *col_name*(*length*) syntax to specify an index prefix length:

- Prefixes can be specified for CHAR, VARCHAR, BINARY, and VARBINARY columns.
- BLOB and TEXT columns also can be indexed, but a prefix length *must* be given.
- Prefix lengths are given in characters for nonbinary string types and in bytes for binary string types. That is, index entries consist of the first *length* characters of each column value for CHAR, VARCHAR, and TEXT columns, and the first *length* bytes of each column value for BINARY, VARBINARY, and BLOB columns.
- For spatial columns, prefix values cannot be given, as described later in this section.

The statement shown here creates an index using the first 10 characters of the name column:

```
CREATE INDEX part_of_name ON customer (name(10));
```

If names in the column usually differ in the first 10 characters, this index should not be much slower than an index created from the entire name column. Also, using column prefixes for indexes can make the index file much smaller, which could save a lot of disk space and might also speed up INSERT operations.

Prefix support and lengths of prefixes (where supported) are storage engine dependent. For example, a prefix can be up to 1000 bytes long for MyISAM tables, and 767 bytes for InnoDB tables.

Note

Prefix limits are measured in bytes, whereas the prefix length in CREATE INDEX statements is interpreted as number of characters for nonbinary data types (CHAR, VARCHAR, TEXT). Take this into account when specifying a prefix length for a column that uses a multi-byte character set.

A UNIQUE index creates a constraint such that all values in the index must be distinct. An error occurs if you try to add a new row with a key value that matches an existing row. For all engines, a UNIQUE index permits multiple NULL values for columns that can contain NULL. If you specify a prefix value for a column in a UNIQUE index, the column values must be unique within the prefix.

FULLTEXT indexes are supported only for InnoDB and MyISAM tables and can include only CHAR, VARCHAR, and TEXT columns. Indexing always happens over the entire column; column prefix indexing is not supported and any prefix length is ignored if specified. See Section 12.9, “Full-Text Search Functions”, for details of operation.

The MyISAM, InnoDB, NDB, and ARCHIVE storage engines support spatial columns such as (POINT and GEOMETRY. (Section 12.18, “Spatial Extensions”, describes the spatial data types.) However, support for spatial column indexing varies among engines. Spatial and nonspatial indexes are available according to the following rules.

Spatial indexes (created using SPATIAL INDEX) have these characteristics:

- Available only for MyISAM tables. Specifying SPATIAL INDEX for other storage engines results in an error.
- Indexed columns must be NOT NULL.
- In MySQL 5.7, column prefix lengths are prohibited. The full width of each column is indexed.

Characteristics of nonspatial indexes (created with INDEX, UNIQUE, or PRIMARY KEY):

- Permitted for any storage engine that supports spatial columns except ARCHIVE.
- Columns can be NULL unless the index is a primary key.
- For each spatial column in a non-SPATIAL index except POINT columns, a column prefix length must be specified. (This is the same requirement as for indexed BLOB columns.) The prefix length is given in bytes.
- The index type for a non-SPATIAL index depends on the storage engine. Currently, B-tree is used.

In MySQL 5.7:

- You can add an index on a column that can have NULL values only if you are using the InnoDB, MyISAM, or MEMORY storage engine.
- You can add an index on a BLOB or TEXT column only if you are using the InnoDB or MyISAM storage engine.
- When the innodb_stats_persistent setting is enabled, run the ANALYZE TABLE statement for an InnoDB table after creating an index on that table.

An *index_col_name* specification can end with ASC or DESC. These keywords are permitted for future extensions for specifying ascending or descending index value storage. Currently, they are parsed but ignored; index values are always stored in ascending order.

Following the index column list, index options can be given. An *index_option* value can be any of the following:

- KEY_BLOCK_SIZE [=] *value*

  For compressed InnoDB tables, optionally specifies the size in bytes to use for pages. The value is treated as a hint; a different size could be used if necessary. A value of 0 represents the default compressed page size. See Section 14.2.7, "InnoDB Compressed Tables" for usage details.

  Note

  Oracle recommends enabling innodb_strict_mode when using the KEY_BLOCK_SIZE clause for InnoDB tables.

- *index_type*

  Some storage engines permit you to specify an index type when creating an index. The permissible index type values supported by different storage engines are shown in the following table. Where multiple index types are listed, the first one is the default when no index type specifier is given.

Storage Engine	Permissible Index Types
InnoDB	BTREE
MyISAM	BTREE
MEMORY/HEAP	HASH, BTREE
NDB	HASH, BTREE (see note in text)

  Example:

```
CREATE TABLE lookup (id INT) ENGINE = MEMORY;
CREATE INDEX id_index ON lookup (id) USING BTREE;
```

The *index_type* clause cannot be used together with SPATIAL INDEX.

If you specify an index type that is not valid for a given storage engine, but there is another index type available that the engine can use without affecting query results, the engine uses the available type. The parser recognizes RTREE as a type name, but currently this cannot be specified for any storage engine.

Use of this option before the ON *tbl_name* clause is deprecated; support for use of the option in this position will be removed in a future MySQL release. If an *index_type* option is given in both the earlier and later positions, the final option applies.

TYPE *type_name* is recognized as a synonym for USING *type_name*. However, USING is the preferred form.

- WITH PARSER *parser_name*

  This option can be used only with FULLTEXT indexes. It associates a parser plugin with the index if full-text indexing and searching operations need special handling. Prior to MySQL 5.7.3, only MyISAM supported full-text parser plugins. As of MySQL 5.7.3, both InnoDB and MyISAM support full-text parser plugins. See Section 22.2.3.2, “Full-Text Parser Plugins” and Section 22.2.4.4, “Writing Full-Text Parser Plugins” for more information.

- COMMENT '*string*'

  Index definitions can include an optional comment of up to 1024 characters.

ALGORITHM and LOCK clauses may be given. These influence the table copying method and level of concurrency for reading and writing the table while its indexes are being modified. They have the same meaning as for the ALTER TABLE statement. For more information, see Section 13.1.6, “ALTER TABLE Syntax”

13.1.12 CREATE PROCEDURE and CREATE FUNCTION Syntax

```
CREATE
    [DEFINER = { user | CURRENT_USER }]
    PROCEDURE sp_name ([proc_parameter[,...]])
    [characteristic ...] routine_body

CREATE
    [DEFINER = { user | CURRENT_USER }]
    FUNCTION sp_name ([func_parameter[,...]])
    RETURNS type
    [characteristic ...] routine_body

proc_parameter:
    [ IN | OUT | INOUT ] param_name type

func_parameter:
    param_name type

type:
    Any valid MySQL data type

characteristic:
    COMMENT 'string'
  | LANGUAGE SQL
  | [NOT] DETERMINISTIC
  | { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }
  | SQL SECURITY { DEFINER | INVOKER }
```

```
routine_body:
    Valid SQL routine statement
```

These statements create stored routines. By default, a routine is associated with the default database. To associate the routine explicitly with a given database, specify the name as *db_name.sp_name* when you create it.

The CREATE FUNCTION statement is also used in MySQL to support UDFs (user-defined functions). See Section 22.3, "Adding New Functions to MySQL". A UDF can be regarded as an external stored function. Stored functions share their namespace with UDFs. See Section 9.2.4, "Function Name Parsing and Resolution", for the rules describing how the server interprets references to different kinds of functions.

To invoke a stored procedure, use the CALL statement (see Section 13.2.1, "CALL Syntax"). To invoke a stored function, refer to it in an expression. The function returns a value during expression evaluation.

CREATE PROCEDURE and CREATE FUNCTION require the CREATE ROUTINE privilege. They might also require the SUPER privilege, depending on the DEFINER value, as described later in this section. If binary logging is enabled, CREATE FUNCTION might require the SUPER privilege, as described in Section 18.7, "Binary Logging of Stored Programs".

By default, MySQL automatically grants the ALTER ROUTINE and EXECUTE privileges to the routine creator. This behavior can be changed by disabling the automatic_sp_privileges system variable. See Section 18.2.2, "Stored Routines and MySQL Privileges".

The DEFINER and SQL SECURITY clauses specify the security context to be used when checking access privileges at routine execution time, as described later in this section.

If the routine name is the same as the name of a built-in SQL function, a syntax error occurs unless you use a space between the name and the following parenthesis when defining the routine or invoking it later. For this reason, avoid using the names of existing SQL functions for your own stored routines.

The IGNORE_SPACE SQL mode applies to built-in functions, not to stored routines. It is always permissible to have spaces after a stored routine name, regardless of whether IGNORE_SPACE is enabled.

The parameter list enclosed within parentheses must always be present. If there are no parameters, an empty parameter list of () should be used. Parameter names are not case sensitive.

Each parameter is an IN parameter by default. To specify otherwise for a parameter, use the keyword OUT or INOUT before the parameter name.

Note

Specifying a parameter as IN, OUT, or INOUT is valid only for a PROCEDURE. For a FUNCTION, parameters are always regarded as IN parameters.

An IN parameter passes a value into a procedure. The procedure might modify the value, but the modification is not visible to the caller when the procedure returns. An OUT parameter passes a value from the procedure back to the caller. Its initial value is NULL within the procedure, and its value is visible to the caller when the procedure returns. An INOUT parameter is initialized by the caller, can be modified by the procedure, and any change made by the procedure is visible to the caller when the procedure returns.

For each OUT or INOUT parameter, pass a user-defined variable in the CALL statement that invokes the procedure so that you can obtain its value when the procedure returns. If you are calling the procedure from within another stored procedure or function, you can also pass a routine parameter or local routine variable as an IN or INOUT parameter.

Routine parameters cannot be referenced in statements prepared within the routine; see Section E.1, "Restrictions on Stored Programs".

The following example shows a simple stored procedure that uses an OUT parameter:

```
mysql> delimiter //

mysql> CREATE PROCEDURE simpleproc (OUT param1 INT)
    -> BEGIN
    ->   SELECT COUNT(*) INTO param1 FROM t;
    -> END//
Query OK, 0 rows affected (0.00 sec)

mysql> delimiter ;

mysql> CALL simpleproc(@a);
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @a;
+------+
| @a   |
+------+
| 3    |
+------+
1 row in set (0.00 sec)
```

The example uses the mysql client delimiter command to change the statement delimiter from ; to // while the procedure is being defined. This enables the ; delimiter used in the procedure body to be passed through to the server rather than being interpreted by mysql itself. See Section 18.1, "Defining Stored Programs".

The RETURNS clause may be specified only for a FUNCTION, for which it is mandatory. It indicates the return type of the function, and the function body must contain a RETURN *value* statement. If the RETURN statement returns a value of a different type, the value is coerced to the proper type. For example, if a function specifies an ENUM or SET value in the RETURNS clause, but the RETURN statement returns an integer, the value returned from the function is the string for the corresponding ENUM member of set of SET members.

The following example function takes a parameter, performs an operation using an SQL function, and returns the result. In this case, it is unnecessary to use delimiter because the function definition contains no internal ; statement delimiters:

```
mysql> CREATE FUNCTION hello (s CHAR(20))
mysql> RETURNS CHAR(50) DETERMINISTIC
    -> RETURN CONCAT('Hello, ',s,'!');
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT hello('world');
+----------------+
| hello('world') |
+----------------+
| Hello, world!  |
+----------------+
1 row in set (0.00 sec)
```

Parameter types and function return types can be declared to use any valid data type. The COLLATE attribute can be used if preceded by the CHARACTER SET attribute.

The *routine_body* consists of a valid SQL routine statement. This can be a simple statement such as SELECT or INSERT, or a compound statement written using BEGIN and END. Compound statements can contain declarations, loops, and other control structure statements. The syntax for these statements is described in Section 13.6, "MySQL Compound-Statement Syntax".

MySQL permits routines to contain DDL statements, such as CREATE and DROP. MySQL also permits stored procedures (but not stored functions) to contain SQL transaction statements such as COMMIT. Stored functions may not contain statements that perform explicit or implicit commit or rollback. Support for these statements is not required by the SQL standard, which states that each DBMS vendor may decide whether to permit them.

Statements that return a result set can be used within a stored procedure but not within a stored function. This prohibition includes SELECT statements that do not have an INTO *var_list* clause and other statements such as SHOW, EXPLAIN, and CHECK TABLE. For statements that can be determined at function definition time to return a result set, a Not allowed to return a result set from a function error occurs (ER_SP_NO_RETSET). For statements that can be determined only at runtime to return a result set, a PROCEDURE %s can't return a result set in the given context error occurs (ER_SP_BADSELECT).

USE statements within stored routines are not permitted. When a routine is invoked, an implicit USE *db_name* is performed (and undone when the routine terminates). The causes the routine to have the given default database while it executes. References to objects in databases other than the routine default database should be qualified with the appropriate database name.

For additional information about statements that are not permitted in stored routines, see Section E.1, “Restrictions on Stored Programs”.

For information about invoking stored procedures from within programs written in a language that has a MySQL interface, see Section 13.2.1, “CALL Syntax”.

MySQL stores the sql_mode system variable setting in effect when a routine is created or altered, and always executes the routine with this setting in force, *regardless of the current server SQL mode when the routine begins executing*.

The switch from the SQL mode of the invoker to that of the routine occurs after evaluation of arguments and assignment of the resulting values to routine parameters. If you define a routine in strict SQL mode but invoke it in nonstrict mode, assignment of arguments to routine parameters does not take place in strict mode. If you require that expressions passed to a routine be assigned in strict SQL mode, you should invoke the routine with strict mode in effect.

The COMMENT characteristic is a MySQL extension, and may be used to describe the stored routine. This information is displayed by the SHOW CREATE PROCEDURE and SHOW CREATE FUNCTION statements.

The LANGUAGE characteristic indicates the language in which the routine is written. The server ignores this characteristic; only SQL routines are supported.

A routine is considered “deterministic” if it always produces the same result for the same input parameters, and “not deterministic” otherwise. If neither DETERMINISTIC nor NOT DETERMINISTIC is given in the routine definition, the default is NOT DETERMINISTIC. To declare that a function is deterministic, you must specify DETERMINISTIC explicitly.

Assessment of the nature of a routine is based on the “honesty” of the creator: MySQL does not check that a routine declared DETERMINISTIC is free of statements that produce nondeterministic results. However, misdeclaring a routine might affect results or affect performance. Declaring a nondeterministic routine as DETERMINISTIC might lead to unexpected results by causing the optimizer to make incorrect execution plan choices. Declaring a deterministic routine as NONDETERMINISTIC might diminish performance by causing available optimizations not to be used.

If binary logging is enabled, the DETERMINISTIC characteristic affects which routine definitions MySQL accepts. See Section 18.7, “Binary Logging of Stored Programs”.

A routine that contains the `NOW()` function (or its synonyms) or `RAND()` is nondeterministic, but it might still be replication-safe. For `NOW()`, the binary log includes the timestamp and replicates correctly. `RAND()` also replicates correctly as long as it is called only a single time during the execution of a routine. (You can consider the routine execution timestamp and random number seed as implicit inputs that are identical on the master and slave.)

Several characteristics provide information about the nature of data use by the routine. In MySQL, these characteristics are advisory only. The server does not use them to constrain what kinds of statements a routine will be permitted to execute.

- `CONTAINS SQL` indicates that the routine does not contain statements that read or write data. This is the default if none of these characteristics is given explicitly. Examples of such statements are `SET @x = 1` or `DO RELEASE_LOCK('abc')`, which execute but neither read nor write data.
- `NO SQL` indicates that the routine contains no SQL statements.
- `READS SQL DATA` indicates that the routine contains statements that read data (for example, `SELECT`), but not statements that write data.
- `MODIFIES SQL DATA` indicates that the routine contains statements that may write data (for example, `INSERT` or `DELETE`).

The `SQL SECURITY` characteristic can be `DEFINER` or `INVOKER` to specify the security context; that is, whether the routine executes using the privileges of the account named in the routine `DEFINER` clause or the user who invokes it. This account must have permission to access the database with which the routine is associated. The default value is `DEFINER`. The user who invokes the routine must have the `EXECUTE` privilege for it, as must the `DEFINER` account if the routine executes in definer security context.

The `DEFINER` clause specifies the MySQL account to be used when checking access privileges at routine execution time for routines that have the `SQL SECURITY DEFINER` characteristic.

If a *`user`* value is given for the `DEFINER` clause, it should be a MySQL account specified as `'user_name'@'host_name'` (the same format used in the `GRANT` statement), `CURRENT_USER`, or `CURRENT_USER()`. The default `DEFINER` value is the user who executes the `CREATE PROCEDURE` or `CREATE FUNCTION` or statement. This is the same as specifying `DEFINER = CURRENT_USER` explicitly.

If you specify the `DEFINER` clause, these rules determine the valid `DEFINER` user values:

- If you do not have the `SUPER` privilege, the only permitted *`user`* value is your own account, either specified literally or by using `CURRENT_USER`. You cannot set the definer to some other account.
- If you have the `SUPER` privilege, you can specify any syntactically valid account name. If the account does not actually exist, a warning is generated.
- Although it is possible to create a routine with a nonexistent `DEFINER` account, an error occurs at routine execution time if the `SQL SECURITY` value is `DEFINER` but the definer account does not exist.

For more information about stored routine security, see Section 18.6, “Access Control for Stored Programs and Views”.

Within a stored routine that is defined with the `SQL SECURITY DEFINER` characteristic, `CURRENT_USER` returns the routine's `DEFINER` value. For information about user auditing within stored routines, see Section 6.3.14, “SQL-Based MySQL Account Activity Auditing”.

Consider the following procedure, which displays a count of the number of MySQL accounts listed in the `mysql.user` table:

```
CREATE DEFINER = 'admin'@'localhost' PROCEDURE account_count()
BEGIN
  SELECT 'Number of accounts:', COUNT(*) FROM mysql.user;
END;
```

The procedure is assigned a DEFINER account of 'admin'@'localhost' no matter which user defines it. It executes with the privileges of that account no matter which user invokes it (because the default security characteristic is DEFINER). The procedure succeeds or fails depending on whether invoker has the EXECUTE privilege for it and 'admin'@'localhost' has the SELECT privilege for the mysql.user table.

Now suppose that the procedure is defined with the SQL SECURITY INVOKER characteristic:

```
CREATE DEFINER = 'admin'@'localhost' PROCEDURE account_count()
SQL SECURITY INVOKER
BEGIN
  SELECT 'Number of accounts:', COUNT(*) FROM mysql.user;
END;
```

The procedure still has a DEFINER of 'admin'@'localhost', but in this case, it executes with the privileges of the invoking user. Thus, the procedure succeeds or fails depending on whether the invoker has the EXECUTE privilege for it and the SELECT privilege for the mysql.user table.

The server handles the data type of a routine parameter, local routine variable created with DECLARE, or function return value as follows:

- Assignments are checked for data type mismatches and overflow. Conversion and overflow problems result in warnings, or errors in strict SQL mode.
- Only scalar values can be assigned. For example, a statement such as SET x = (SELECT 1, 2) is invalid.
- For character data types, if there is a CHARACTER SET attribute in the declaration, the specified character set and its default collation is used. If the COLLATE attribute is also present, that collation is used rather than the default collation. If there is no CHARACTER SET attribute, the database character set and collation in effect at routine creation time are used. (The database character set and collation are given by the value of the character_set_database and collation_database system variables.)

  If you change the database default character set or collation, stored routines that use the database defaults must be dropped and recreated so that they use the new defaults.

13.1.13 CREATE SERVER Syntax

```
CREATE SERVER server_name
    FOREIGN DATA WRAPPER wrapper_name
    OPTIONS (option [, option] ...)

option:
  { HOST character-literal
  | DATABASE character-literal
  | USER character-literal
  | PASSWORD character-literal
  | SOCKET character-literal
  | OWNER character-literal
  | PORT numeric-literal }
```

This statement creates the definition of a server for use with the FEDERATED storage engine. The CREATE SERVER statement creates a new row in the servers table in the mysql database. This statement requires the SUPER privilege.

The *server_name* should be a unique reference to the server. Server definitions are global within the scope of the server, it is not possible to qualify the server definition to a specific database. *server_name* has a maximum length of 64 characters (names longer than 64 characters are silently truncated), and is case insensitive. You may specify the name as a quoted string.

The *wrapper_name* should be mysql, and may be quoted with single quotation marks. Other values for *wrapper_name* are not currently supported.

For each *option* you must specify either a character literal or numeric literal. Character literals are UTF-8, support a maximum length of 64 characters and default to a blank (empty) string. String literals are silently truncated to 64 characters. Numeric literals must be a number between 0 and 9999, default value is 0.

> **Note**
>
> The OWNER option is currently not applied, and has no effect on the ownership or operation of the server connection that is created.

The CREATE SERVER statement creates an entry in the mysql.servers table that can later be used with the CREATE TABLE statement when creating a FEDERATED table. The options that you specify will be used to populate the columns in the mysql.servers table. The table columns are Server_name, Host, Db, Username, Password, Port and Socket.

For example:

```
CREATE SERVER s
FOREIGN DATA WRAPPER mysql
OPTIONS (USER 'Remote', HOST '192.168.1.106', DATABASE 'test');
```

Be sure to specify all options necessary to establish a connection to the server. The user name, host name, and database name are mandatory. Other options might be required as well, such as password.

The data stored in the table can be used when creating a connection to a FEDERATED table:

```
CREATE TABLE t (s1 INT) ENGINE=FEDERATED CONNECTION='s';
```

For more information, see Section 14.9, “The FEDERATED Storage Engine”.

CREATE SERVER causes an automatic commit.

In MySQL 5.7, CREATE SERVER is not written to the binary log, regardless of the logging format that is in use.

In MySQL 5.7.1, gtid_next must be set to AUTOMATIC before issuing this statement. This restriction does not apply in MySQL 5.7.2 or later. (Bug #16062608, Bug #16715809, Bug #69045)

13.1.14 CREATE TABLE Syntax

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name
    (create_definition,...)
    [table_options]
    [partition_options]
```

Or:

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name
    [(create_definition,...)]
    [table_options]
    [partition_options]
    select_statement
```

Or:

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name
    { LIKE old_tbl_name | (LIKE old_tbl_name) }
```

```
create_definition:
    col_name column_definition
  | [CONSTRAINT [symbol]] PRIMARY KEY [index_type] (index_col_name,...)
      [index_option] ...
  | {INDEX|KEY} [index_name] [index_type] (index_col_name,...)
      [index_option] ...
  | [CONSTRAINT [symbol]] UNIQUE [INDEX|KEY]
      [index_name] [index_type] (index_col_name,...)
      [index_option] ...
  | {FULLTEXT|SPATIAL} [INDEX|KEY] [index_name] (index_col_name,...)
      [index_option] ...
  | [CONSTRAINT [symbol]] FOREIGN KEY
      [index_name] (index_col_name,...) reference_definition
  | CHECK (expr)

column_definition:
    data_type [NOT NULL | NULL] [DEFAULT default_value]
      [AUTO_INCREMENT] [UNIQUE [KEY] | [PRIMARY] KEY]
      [COMMENT 'string']
      [COLUMN_FORMAT {FIXED|DYNAMIC|DEFAULT}]
      [reference_definition]

data_type:
    BIT[(length)]
  | TINYINT[(length)] [UNSIGNED] [ZEROFILL]
  | SMALLINT[(length)] [UNSIGNED] [ZEROFILL]
  | MEDIUMINT[(length)] [UNSIGNED] [ZEROFILL]
  | INT[(length)] [UNSIGNED] [ZEROFILL]
  | INTEGER[(length)] [UNSIGNED] [ZEROFILL]
  | BIGINT[(length)] [UNSIGNED] [ZEROFILL]
  | REAL[(length,decimals)] [UNSIGNED] [ZEROFILL]
  | DOUBLE[(length,decimals)] [UNSIGNED] [ZEROFILL]
  | FLOAT[(length,decimals)] [UNSIGNED] [ZEROFILL]
  | DECIMAL[(length[,decimals])] [UNSIGNED] [ZEROFILL]
  | NUMERIC[(length[,decimals])] [UNSIGNED] [ZEROFILL]
  | DATE
  | TIME
  | TIMESTAMP
  | DATETIME
  | YEAR
  | CHAR[(length)]
      [CHARACTER SET charset_name] [COLLATE collation_name]
  | VARCHAR(length)
      [CHARACTER SET charset_name] [COLLATE collation_name]
  | BINARY[(length)]
  | VARBINARY(length)
  | TINYBLOB
  | BLOB
  | MEDIUMBLOB
  | LONGBLOB
  | TINYTEXT [BINARY]
      [CHARACTER SET charset_name] [COLLATE collation_name]
  | TEXT [BINARY]
      [CHARACTER SET charset_name] [COLLATE collation_name]
  | MEDIUMTEXT [BINARY]
      [CHARACTER SET charset_name] [COLLATE collation_name]
  | LONGTEXT [BINARY]
      [CHARACTER SET charset_name] [COLLATE collation_name]
  | ENUM(value1,value2,value3,...)
      [CHARACTER SET charset_name] [COLLATE collation_name]
  | SET(value1,value2,value3,...)
      [CHARACTER SET charset_name] [COLLATE collation_name]
  | spatial_type

index_col_name:
    col_name [(length)] [ASC | DESC]

index_type:
    USING {BTREE | HASH}

index_option:
    KEY_BLOCK_SIZE [=] value
  | index_type
  | WITH PARSER parser_name
  | COMMENT 'string'

reference_definition:
    REFERENCES tbl_name (index_col_name,...)
      [MATCH FULL | MATCH PARTIAL | MATCH SIMPLE]
      [ON DELETE reference_option]
      [ON UPDATE reference_option]

reference_option:
    RESTRICT | CASCADE | SET NULL | NO ACTION

table_options:
    table_option [[,] table_option] ...

table_option:
    ENGINE [=] engine_name
  | AUTO_INCREMENT [=] value
  | AVG_ROW_LENGTH [=] value
  | [DEFAULT] CHARACTER SET [=] charset_name
  | CHECKSUM [=] {0 | 1}
  | [DEFAULT] COLLATE [=] collation_name
  | COMMENT [=] 'string'
  | CONNECTION [=] 'connect_string'
  | DATA DIRECTORY [=] 'absolute path to directory'
  | DELAY_KEY_WRITE [=] {0 | 1}
  | INDEX DIRECTORY [=] 'absolute path to directory'
  | INSERT_METHOD [=] { NO | FIRST | LAST }
  | KEY_BLOCK_SIZE [=] value
  | MAX_ROWS [=] value
  | MIN_ROWS [=] value
  | PACK_KEYS [=] {0 | 1 | DEFAULT}
  | PASSWORD [=] 'string'
  | ROW_FORMAT [=] {DEFAULT|DYNAMIC|FIXED|COMPRESSED|REDUNDANT|COMPACT}
  | STATS_AUTO_RECALC [=] {DEFAULT|0|1}
  | STATS_PERSISTENT [=] {DEFAULT|0|1}
  | STATS_SAMPLE_PAGES [=] value
  | UNION [=] (tbl_name[,tbl_name]...)

partition_options:
    PARTITION BY
        { [LINEAR] HASH(expr)
        | [LINEAR] KEY [ALGORITHM={1|2}] (column_list)
        | RANGE{(expr) | COLUMNS(column_list)}
        | LIST{(expr) | COLUMNS(column_list)} }
    [PARTITIONS num]
    [SUBPARTITION BY
        { [LINEAR] HASH(expr)
        | [LINEAR] KEY [ALGORITHM={1|2}] (column_list) }
      [SUBPARTITIONS num]
    ]
    [(partition_definition [, partition_definition] ...)]

partition_definition:
    PARTITION partition_name
        [VALUES
            {LESS THAN {(expr | value_list) | MAXVALUE}
            |
            IN (value_list)}]
        [[STORAGE] ENGINE [=] engine_name]
        [COMMENT [=] 'comment_text' ]
        [DATA DIRECTORY [=] 'data_dir']
        [INDEX DIRECTORY [=] 'index_dir']
        [MAX_ROWS [=] max_number_of_rows]
        [MIN_ROWS [=] min_number_of_rows]
        [(subpartition_definition [, subpartition_definition] ...)]

subpartition_definition:
    SUBPARTITION logical_name
        [[STORAGE] ENGINE [=] engine_name]
        [COMMENT [=] 'comment_text' ]
        [DATA DIRECTORY [=] 'data_dir']
        [INDEX DIRECTORY [=] 'index_dir']
        [MAX_ROWS [=] max_number_of_rows]
        [MIN_ROWS [=] min_number_of_rows]

select_statement:
    [IGNORE | REPLACE] [AS] SELECT ...   (Some valid select statement)
```

CREATE TABLE creates a table with the given name. You must have the CREATE privilege for the table.

Rules for permissible table names are given in Section 9.2, “Schema Object Names”. By default, the table is created in the default database, using the InnoDB storage engine. An error occurs if the table exists, if there is no default database, or if the database does not exist.

The table name can be specified as *db_name.tbl_name* to create the table in a specific database. This works regardless of whether there is a default database, assuming that the database exists. If you use quoted identifiers, quote the database and table names separately. For example, write `` `mydb`.`mytbl` ``, not `` `mydb.mytbl` ``.

Temporary Tables

You can use the TEMPORARY keyword when creating a table. A TEMPORARY table is visible only to the current connection, and is dropped automatically when the connection is closed. This means that two different connections can use the same temporary table name without conflicting with each other or with an existing non-TEMPORARY table of the same name. (The existing table is hidden until the temporary table is dropped.) To create temporary tables, you must have the CREATE TEMPORARY TABLES privilege.

Note

CREATE TABLE does not automatically commit the current active transaction if you use the TEMPORARY keyword.

Existing Table with Same Name

The keywords IF NOT EXISTS prevent an error from occurring if the table exists. However, there is no verification that the existing table has a structure identical to that indicated by the CREATE TABLE statement.

Physical Representation

MySQL represents each table by an .frm table format (definition) file in the database directory. The storage engine for the table might create other files as well.

For InnoDB tables, the file storage is controlled by the innodb_file_per_table configuration option. When this option is turned off, all InnoDB tables and indexes are stored in the system tablespace, represented by one or more .ibd files. For each InnoDB table created when this option is turned on, the table data and all associated indexes are stored in a .ibd file located inside the database directory.

For MyISAM tables, the storage engine creates data and index files. Thus, for each MyISAM table *tbl_name*, there are three disk files.

File	Purpose
tbl_name.frm	Table format (definition) file
tbl_name.MYD	Data file
tbl_name.MYI	Index file

Chapter 14, *Storage Engines*, describes what files each storage engine creates to represent tables. If a table name contains special characters, the names for the table files contain encoded versions of those characters as described in Section 9.2.3, “Mapping of Identifiers to File Names”.

Data Types and Attributes for Columns

data_type represents the data type in a column definition. *spatial_type* represents a spatial data type. The data type syntax shown is representative only. For a full description of the syntax available for specifying column data types, as well as information about the properties of each type, see Chapter 11, *Data Types*, and Section 12.18, “Spatial Extensions”.

Some attributes do not apply to all data types. AUTO_INCREMENT applies only to integer and floating-point types. DEFAULT does not apply to the BLOB or TEXT types.

- If neither NULL nor NOT NULL is specified, the column is treated as though NULL had been specified.

- An integer or floating-point column can have the additional attribute AUTO_INCREMENT. When you insert a value of NULL (recommended) or 0 into an indexed AUTO_INCREMENT column, the column is set to the next sequence value. Typically this is *value*+1, where *value* is the largest value for the column currently in the table. AUTO_INCREMENT sequences begin with 1.

  To retrieve an AUTO_INCREMENT value after inserting a row, use the LAST_INSERT_ID() SQL function or the mysql_insert_id() C API function. See Section 12.14, “Information Functions”, and Section 21.8.7.38, “mysql_insert_id()”.

  If the NO_AUTO_VALUE_ON_ZERO SQL mode is enabled, you can store 0 in AUTO_INCREMENT columns as 0 without generating a new sequence value. See Section 5.1.7, “Server SQL Modes”.

  > **Note**
  >
  > There can be only one AUTO_INCREMENT column per table, it must be indexed, and it cannot have a DEFAULT value. An AUTO_INCREMENT column works properly only if it contains only positive values. Inserting a negative number is regarded as inserting a very large positive number. This is done to avoid precision problems when numbers “wrap” over from positive to negative and also to ensure that you do not accidentally get an AUTO_INCREMENT column that contains 0.

  For MyISAM tables, you can specify an AUTO_INCREMENT secondary column in a multiple-column key. See Section 3.6.9, “Using AUTO_INCREMENT”.

To make MySQL compatible with some ODBC applications, you can find the AUTO_INCREMENT value for the last inserted row with the following query:

```
SELECT * FROM tbl_name WHERE auto_col IS NULL
```

For information about InnoDB and AUTO_INCREMENT, see Section 14.2.6.5, “AUTO_INCREMENT Handling in InnoDB”. For information about AUTO_INCREMENT and MySQL Replication, see Section 16.4.1.1, “Replication and AUTO_INCREMENT”.

- Character data types (CHAR, VARCHAR, TEXT) can include CHARACTER SET and COLLATE attributes to specify the character set and collation for the column. For details, see Section 10.1, “Character Set Support”. CHARSET is a synonym for CHARACTER SET. Example:

```
CREATE TABLE t (c CHAR(20) CHARACTER SET utf8 COLLATE utf8_bin);
```

MySQL 5.7 interprets length specifications in character column definitions in characters. (Versions before MySQL 4.1 interpreted them in bytes.) Lengths for BINARY and VARBINARY are in bytes.

- The DEFAULT clause specifies a default value for a column. With one exception, the default value must be a constant; it cannot be a function or an expression. This means, for example, that you cannot set the default for a date column to be the value of a function such as NOW() or CURRENT_DATE. The exception is that you can specify CURRENT_TIMESTAMP as the default for a TIMESTAMP or DATETIME column. See Section 11.3.5, “Automatic Initialization and Updating for TIMESTAMP and DATETIME”.

If a column definition includes no explicit DEFAULT value, MySQL determines the default value as described in Section 11.5, “Data Type Default Values”.

BLOB and TEXT columns cannot be assigned a default value.

Before MySQL 5.7.4, if the NO_ZERO_DATE or NO_ZERO_IN_DATE SQL mode is enabled and a date-valued default is not correct according to that mode, CREATE TABLE produces a warning if strict SQL mode is not enabled and an error if strict mode is enabled. For example, with NO_ZERO_IN_DATE enabled, c1 DATE DEFAULT '2010-00-00' produces a warning.

As of MySQL 5.7.4, if strict SQL mode is enabled and a date-valued default is not correct according to that mode, CREATE TABLE produces an error.

- A comment for a column can be specified with the COMMENT option, up to 1024 characters long. The comment is displayed by the SHOW CREATE TABLE and SHOW FULL COLUMNS statements.

- KEY is normally a synonym for INDEX. The key attribute PRIMARY KEY can also be specified as just KEY when given in a column definition. This was implemented for compatibility with other database systems.

- A UNIQUE index creates a constraint such that all values in the index must be distinct. An error occurs if you try to add a new row with a key value that matches an existing row. For all engines, a UNIQUE index permits multiple NULL values for columns that can contain NULL.

- A PRIMARY KEY is a unique index where all key columns must be defined as NOT NULL. If they are not explicitly declared as NOT NULL, MySQL declares them so implicitly (and silently). A table can have only one PRIMARY KEY. The name of a PRIMARY KEY is always PRIMARY, which thus cannot be used as the name for any other kind of index.

If you do not have a PRIMARY KEY and an application asks for the PRIMARY KEY in your tables, MySQL returns the first UNIQUE index that has no NULL columns as the PRIMARY KEY.

In `InnoDB` tables, keep the `PRIMARY KEY` short to minimize storage overhead for secondary indexes. Each secondary index entry contains a copy of the primary key columns for the corresponding row. (See Section 14.2.2.14, “`InnoDB` Table and Index Structures”.)

- In the created table, a `PRIMARY KEY` is placed first, followed by all `UNIQUE` indexes, and then the nonunique indexes. This helps the MySQL optimizer to prioritize which index to use and also more quickly to detect duplicated `UNIQUE` keys.

- A `PRIMARY KEY` can be a multiple-column index. However, you cannot create a multiple-column index using the `PRIMARY KEY` key attribute in a column specification. Doing so only marks that single column as primary. You must use a separate `PRIMARY KEY(`*`index_col_name, ...`*`)` clause.

- If a `PRIMARY KEY` or `UNIQUE` index consists of only one column that has an integer type, you can also refer to the column as `_rowid` in `SELECT` statements.

- In MySQL, the name of a `PRIMARY KEY` is `PRIMARY`. For other indexes, if you do not assign a name, the index is assigned the same name as the first indexed column, with an optional suffix (`_2`, `_3`, `...`) to make it unique. You can see index names for a table using `SHOW INDEX FROM` *`tbl_name`*. See Section 13.7.5.21, “`SHOW INDEX` Syntax”.

- Some storage engines permit you to specify an index type when creating an index. The syntax for the *`index_type`* specifier is `USING` *`type_name`*.

  Example:

  ```
  CREATE TABLE lookup
    (id INT, INDEX USING BTREE (id))
    ENGINE = MEMORY;
  ```

  The preferred position for `USING` is after the index column list. It can be given before the column list, but support for use of the option in that position is deprecated and will be removed in a future MySQL release.

  `index_option` values specify additional options for an index. `USING` is one such option. The `WITH PARSER` option can only be used with `FULLTEXT` indexes. It associates a parser plugin with the index if full-text indexing and searching operations need special handling. Prior to MySQL 5.7.3, only `MyISAM` supported full-text parser plugins. As of MySQL 5.7.3, both `InnoDB` and `MyISAM` support full-text parser plugins. If you have a `MyISAM` table with an associated full-text parser plugin, you can convert the table to `InnoDB` using `ALTER TABLE`.

  For more information about permissible *`index_option`* values, see Section 13.1.11, “`CREATE INDEX` Syntax”. For more information about indexes, see Section 8.3.1, “How MySQL Uses Indexes”.

- In MySQL 5.7, only the `InnoDB`, `MyISAM`, and `MEMORY` storage engines support indexes on columns that can have `NULL` values. In other cases, you must declare indexed columns as `NOT NULL` or an error results.

- For `CHAR`, `VARCHAR`, `BINARY`, and `VARBINARY` columns, indexes can be created that use only the leading part of column values, using *`col_name`*`(`*`length`*`)` syntax to specify an index prefix length. `BLOB` and `TEXT` columns also can be indexed, but a prefix length *must* be given. Prefix lengths are given in characters for nonbinary string types and in bytes for binary string types. That is, index entries consist of the first *`length`* characters of each column value for `CHAR`, `VARCHAR`, and `TEXT` columns, and the first *`length`* bytes of each column value for `BINARY`, `VARBINARY`, and `BLOB` columns. Indexing only a prefix of column values like this can make the index file much smaller. See Section 8.3.4, “Column Indexes”.

Only the InnoDB and MyISAM storage engines support indexing on BLOB and TEXT columns. For example:

```
CREATE TABLE test (blob_col BLOB, INDEX(blob_col(10)));
```

Prefixes can be up to 1000 bytes long (767 bytes for InnoDB tables). Note that prefix limits are measured in bytes, whereas the prefix length in CREATE TABLE statements is interpreted as number of characters for nonbinary data types (CHAR, VARCHAR, TEXT). Take this into account when specifying a prefix length for a column that uses a multi-byte character set.

- An *index_col_name* specification can end with ASC or DESC. These keywords are permitted for future extensions for specifying ascending or descending index value storage. Currently, they are parsed but ignored; index values are always stored in ascending order.

- When you use ORDER BY or GROUP BY on a column in a SELECT, the server sorts values using only the initial number of bytes indicated by the max_sort_length system variable.

- You can create special FULLTEXT indexes, which are used for full-text searches. Only the InnoDB and MyISAM storage engines support FULLTEXT indexes. They can be created only from CHAR, VARCHAR, and TEXT columns. Indexing always happens over the entire column; column prefix indexing is not supported and any prefix length is ignored if specified. See Section 12.9, “Full-Text Search Functions”, for details of operation. A WITH PARSER clause can be specified as an *index_option* value to associate a parser plugin with the index if full-text indexing and searching operations need special handling. This clause is valid only for FULLTEXT indexes. Prior to MySQL 5.7.3, only MyISAM supported full-text parser plugins. As of MySQL 5.7.3, both InnoDB and MyISAM support full-text parser plugins. See Section 22.2.3.2, “Full-Text Parser Plugins” and Section 22.2.4.4, “Writing Full-Text Parser Plugins” for more information.

- You can create SPATIAL indexes on spatial data types. Spatial types are supported only for MyISAM tables and indexed columns must be declared as NOT NULL. See Section 12.18, “Spatial Extensions”.

- In MySQL 5.7, index definitions can include an optional comment of up to 1024 characters.

- InnoDB tables support checking of foreign key constraints. The columns of the referenced table must always be explicitly named. Both ON DELETE and ON UPDATE actions on foreign keys. For more detailed information and examples, see Section 13.1.14.2, “Using FOREIGN KEY Constraints”. For information specific to foreign keys in InnoDB, see Section 14.2.6.6, “InnoDB and FOREIGN KEY Constraints”.

  For other storage engines, MySQL Server parses and ignores the FOREIGN KEY and REFERENCES syntax in CREATE TABLE statements. The CHECK clause is parsed but ignored by all storage engines. See Section 1.8.2.4, “Foreign Key Differences”.

⚠ **Important**

For users familiar with the ANSI/ISO SQL Standard, please note that no storage engine, including InnoDB, recognizes or enforces the MATCH clause used in referential integrity constraint definitions. Use of an explicit MATCH clause will not have the specified effect, and also causes ON DELETE and ON UPDATE clauses to be ignored. For these reasons, specifying MATCH should be avoided.

The MATCH clause in the SQL standard controls how NULL values in a composite (multiple-column) foreign key are handled when comparing to a primary key. InnoDB essentially implements the semantics defined by MATCH SIMPLE, which permit a foreign key to be all or partially NULL. In that case, the (child table) row containing such a foreign key is permitted to be inserted, and does not match any row in the referenced (parent) table. It is possible to implement other semantics using triggers.

Additionally, MySQL requires that the referenced columns be indexed for performance. However, InnoDB does not enforce any requirement that the referenced columns be declared UNIQUE or NOT NULL. The handling of foreign key references to nonunique keys or keys that contain NULL values is not well defined for operations such as UPDATE or DELETE CASCADE. You are advised to use foreign keys that reference only keys that are both UNIQUE (or PRIMARY) and NOT NULL.

MySQL does not recognize or support "inline REFERENCES specifications" (as defined in the SQL standard) where the references are defined as part of the column specification. MySQL accepts REFERENCES clauses only when specified as part of a separate FOREIGN KEY specification.

Note

Partitioned tables employing the InnoDB storage engine do not support foreign keys. See Section 17.6, "Restrictions and Limitations on Partitioning", for more information.

- There is a hard limit of 4096 columns per table, but the effective maximum may be less for a given table and depends on the factors discussed in Section E.10.4, "Limits on Table Column Count and Row Size".

Storage Engines

The ENGINE table option specifies the storage engine for the table, using one of the names shown in the following table. The engine name can be unquoted or quoted. The quoted name 'DEFAULT' is recognized but ignored.

Storage Engine	Description
InnoDB	Transaction-safe tables with row locking and foreign keys. The default storage engine for new tables. See Section 14.2, "The InnoDB Storage Engine", and in particular Section 14.2.1.1, "InnoDB as the Default MySQL Storage Engine" if you have MySQL experience but are new to InnoDB.
MyISAM	The binary portable storage engine that is primarily used for read-only or read-mostly workloads. See Section 14.3, "The MyISAM Storage Engine".
MEMORY	The data for this storage engine is stored only in memory. See Section 14.4, "The MEMORY Storage Engine".
CSV	Tables that store rows in comma-separated values format. See Section 14.5, "The CSV Storage Engine".
ARCHIVE	The archiving storage engine. See Section 14.6, "The ARCHIVE Storage Engine".
EXAMPLE	An example engine. See Section 14.10, "The EXAMPLE Storage Engine".
FEDERATED	Storage engine that accesses remote tables. See Section 14.9, "The FEDERATED Storage Engine".
HEAP	This is a synonym for MEMORY.
MERGE	A collection of MyISAM tables used as one table. Also known as MRG_MyISAM. See Section 14.8, "The MERGE Storage Engine".

If a storage engine is specified that is not available, MySQL uses the default engine instead. Normally, this is MyISAM. For example, if a table definition includes the ENGINE=INNODB option but the MySQL server does not support INNODB tables, the table is created as a MyISAM table. This makes it possible to have a replication setup where you have transactional tables on the master but tables created on the slave are nontransactional (to get more speed). In MySQL 5.7, a warning occurs if the storage engine specification is not honored.

Engine substitution can be controlled by the setting of the NO_ENGINE_SUBSTITUTION SQL mode, as described in Section 5.1.7, “Server SQL Modes”.

Note

The older TYPE option that was synonymous with ENGINE was removed in MySQL 5.5. *When upgrading to MySQL 5.5 or later, you must convert existing applications that rely on TYPE to use ENGINE instead*.

Optimizing Performance

The other table options are used to optimize the behavior of the table. In most cases, you do not have to specify any of them. These options apply to all storage engines unless otherwise indicated. Options that do not apply to a given storage engine may be accepted and remembered as part of the table definition. Such options then apply if you later use ALTER TABLE to convert the table to use a different storage engine.

- AUTO_INCREMENT

  The initial AUTO_INCREMENT value for the table. In MySQL 5.7, this works for MyISAM, MEMORY, InnoDB, and ARCHIVE tables. To set the first auto-increment value for engines that do not support the AUTO_INCREMENT table option, insert a “dummy” row with a value one less than the desired value after creating the table, and then delete the dummy row.

  For engines that support the AUTO_INCREMENT table option in CREATE TABLE statements, you can also use ALTER TABLE *tbl_name* AUTO_INCREMENT = *N* to reset the AUTO_INCREMENT value. The value cannot be set lower than the maximum value currently in the column.

- AVG_ROW_LENGTH

  An approximation of the average row length for your table. You need to set this only for large tables with variable-size rows.

  When you create a MyISAM table, MySQL uses the product of the MAX_ROWS and AVG_ROW_LENGTH options to decide how big the resulting table is. If you don't specify either option, the maximum size for MyISAM data and index files is 256TB by default. (If your operating system does not support files that large, table sizes are constrained by the file size limit.) If you want to keep down the pointer sizes to make the index smaller and faster and you don't really need big files, you can decrease the default pointer size by setting the myisam_data_pointer_size system variable. (See Section 5.1.4, “Server System Variables”.) If you want all your tables to be able to grow above the default limit and are willing to have your tables slightly slower and larger than necessary, you can increase the default pointer size by setting this variable. Setting the value to 7 permits table sizes up to 65,536TB.

- [DEFAULT] CHARACTER SET

  Specify a default character set for the table. CHARSET is a synonym for CHARACTER SET. If the character set name is DEFAULT, the database character set is used.

- CHECKSUM

Set this to 1 if you want MySQL to maintain a live checksum for all rows (that is, a checksum that MySQL updates automatically as the table changes). This makes the table a little slower to update, but also makes it easier to find corrupted tables. The CHECKSUM TABLE statement reports the checksum. (MyISAM only.)

- [DEFAULT] COLLATE

  Specify a default collation for the table.

- COMMENT

  A comment for the table, up to 2048 characters long.

- CONNECTION

  The connection string for a FEDERATED table.

  Note

  Older versions of MySQL used a COMMENT option for the connection string.

- DATA DIRECTORY, INDEX DIRECTORY

  By using DATA DIRECTORY='*directory*', you can specify where the InnoDB storage engine puts the .ibd tablespace file for a new table. This clause only applies when the innodb_file_per_table configuration option is enabled. The directory must be the full path name to the directory, not a relative path. See Section 14.2.5.4, “Specifying the Location of a Tablespace” for additional information.

  When creating MyISAM tables, you can use the DATA DIRECTORY='*directory*' clause, the INDEX DIRECTORY='*directory*' clause, or both. They specify where to put a MyISAM table's data file and index file respectively.

  Important

  Table-level DATA DIRECTORY and INDEX DIRECTORY options are ignored for partitioned tables. (Bug #32091)

  These options work only when you are not using the --skip-symbolic-links option. Your operating system must also have a working, thread-safe realpath() call. See Using Symbolic Links for MyISAM Tables on Unix, for more complete information.

  If a MyISAM table is created with no DATA DIRECTORY option, the .MYD file is created in the database directory. By default, if MyISAM finds an existing .MYD file in this case, it overwrites it. The same applies to .MYI files for tables created with no INDEX DIRECTORY option. To suppress this behavior, start the server with the --keep_files_on_create option, in which case MyISAM will not overwrite existing files and returns an error instead.

  If a MyISAM table is created with a DATA DIRECTORY or INDEX DIRECTORY option and an existing .MYD or .MYI file is found, MyISAM always returns an error. It will not overwrite a file in the specified directory.

  Important

  You cannot use path names that contain the MySQL data directory with DATA DIRECTORY or INDEX DIRECTORY. This includes partitioned tables and individual table partitions. (See Bug #32167.)

- DELAY_KEY_WRITE

  Set this to 1 if you want to delay key updates for the table until the table is closed. See the description of the delay_key_write system variable in Section 5.1.4, "Server System Variables". (MyISAM only.)

- INSERT_METHOD

  If you want to insert data into a MERGE table, you must specify with INSERT_METHOD the table into which the row should be inserted. INSERT_METHOD is an option useful for MERGE tables only. Use a value of FIRST or LAST to have inserts go to the first or last table, or a value of NO to prevent inserts. See Section 14.8, "The MERGE Storage Engine".

- KEY_BLOCK_SIZE

  For compressed InnoDB tables, optionally specifies the size in kilobytes to use for pages. The value is treated as a hint; a different size could be used if necessary. A value of 0 represents that the default compressed page size. See Section 14.2.7, "InnoDB Compressed Tables" for usage details.

  Individual index definitions can specify a KEY_BLOCK_SIZE value of their own to override the table value.

  Note

  Oracle recommends enabling innodb_strict_mode when using the KEY_BLOCK_SIZE clause for InnoDB tables.

- MAX_ROWS

  The maximum number of rows you plan to store in the table. This is not a hard limit, but rather a hint to the storage engine that the table must be able to store at least this many rows.

  The maximum MAX_ROWS value is 4294967295; larger values are truncated to this limit.

- MIN_ROWS

  The minimum number of rows you plan to store in the table. The MEMORY storage engine uses this option as a hint about memory use.

- PACK_KEYS

  PACK_KEYS takes effect only with MyISAM tables. Set this option to 1 if you want to have smaller indexes. This usually makes updates slower and reads faster. Setting the option to 0 disables all packing of keys. Setting it to DEFAULT tells the storage engine to pack only long CHAR, VARCHAR, BINARY, or VARBINARY columns.

  If you do not use PACK_KEYS, the default is to pack strings, but not numbers. If you use PACK_KEYS=1, numbers are packed as well.

  When packing binary number keys, MySQL uses prefix compression:

  - Every key needs one extra byte to indicate how many bytes of the previous key are the same for the next key.

  - The pointer to the row is stored in high-byte-first order directly after the key, to improve compression.

  This means that if you have many equal keys on two consecutive rows, all following "same" keys usually only take two bytes (including the pointer to the row). Compare this to the ordinary case where the following keys takes storage_size_for_key + pointer_size (where the pointer size is usually

4). Conversely, you get a significant benefit from prefix compression only if you have many numbers that are the same. If all keys are totally different, you use one byte more per key, if the key is not a key that can have NULL values. (In this case, the packed key length is stored in the same byte that is used to mark if a key is NULL.)

- PASSWORD

  This option is unused. If you have a need to scramble your .frm files and make them unusable to any other MySQL server, please contact our sales department.

- ROW_FORMAT

  Defines the physical format in which the rows are stored. The choices differ depending on the storage engine used for the table.

  For InnoDB tables:

  - Rows are stored in compact format (ROW_FORMAT=COMPACT) by default.

  - The noncompact format used in older versions of MySQL can still be requested by specifying ROW_FORMAT=REDUNDANT.

  - To enable compression for InnoDB tables, specify ROW_FORMAT=COMPRESSED and follow the procedures in Section 14.2.7, “InnoDB Compressed Tables”.

  - For more efficient InnoDB storage of data types, especially BLOB types, specify ROW_FORMAT=DYNAMIC and follow the procedures in Section 14.2.9.3, “DYNAMIC and COMPRESSED Row Formats”. Both the COMPRESSED and DYNAMIC row formats require creating the table with the configuration settings innodb_file_per_table=1 and innodb_file_format=barracuda.

  - When you specify a non-default ROW_FORMAT clause, consider also enabling the innodb_strict_mode configuration option.

  - For additional information about InnoDB row formats, see Section 14.2.9, “InnoDB Row Storage and Row Formats”.

  For MyISAM tables, the option value can be FIXED or DYNAMIC for static or variable-length row format. myisampack sets the type to COMPRESSED. See Section 14.3.3, “MyISAM Table Storage Formats”.

  Note

  When executing a CREATE TABLE statement, if you specify a row format that is not supported by the storage engine that is used for the table, the table is created using that storage engine's default row format. The information reported in this column in response to SHOW TABLE STATUS is the actual row format used. This may differ from the value in the Create_options column because the original CREATE TABLE definition is retained during creation.

- STATS_AUTO_RECALC

  Specifies whether to automatically recalculate persistent statistics for an InnoDB table. The value DEFAULT causes the persistent statistics setting for the table to be determined by the innodb_stats_auto_recalc configuration option. The value 1 causes statistics to be recalculated when 10% of the data in the table has changed. The value 0 prevents automatic recalculation for this table; with this setting, issue an ANALYZE TABLE statement to recalculate the statistics after making substantial changes to the table. For more information about the persistent statistics feature, see Persistent Optimizer Statistics for InnoDB Tables.

- STATS_PERSISTENT

  Specifies whether to enable persistent statistics for an InnoDB table. The value DEFAULT causes the persistent statistics setting for the table to be determined by the innodb_stats_persistent configuration option. The value 1 enables persistent statistics for the table, while the value 0 turns off this feature. After enabling persistent statistics through a CREATE TABLE or ALTER TABLE statement, issue an ANALYZE TABLE statement to calculate the statistics, after loading representative data into the table. For more information about the persistent statistics feature, see Persistent Optimizer Statistics for InnoDB Tables.

- STATS_SAMPLE_PAGES

  The number of index pages to sample when estimating cardinality and other statistics for an indexed column, such as those calculated by ANALYZE TABLE. For more information, see Section 14.2.12.5, “Controlling Optimizer Statistics Estimation”.

- UNION

  UNION is used when you want to access a collection of identical MyISAM tables as one. This works only with MERGE tables. See Section 14.8, “The MERGE Storage Engine”.

  You must have SELECT, UPDATE, and DELETE privileges for the tables you map to a MERGE table.

  Note

  Formerly, all tables used had to be in the same database as the MERGE table itself. This restriction no longer applies.

Partitioning

partition_options can be used to control partitioning of the table created with CREATE TABLE.

Important

Not all options shown in the syntax for *partition_options* at the beginning of this section are available for all partitioning types. Please see the listings for the following individual types for information specific to each type, and see Chapter 17, *Partitioning*, for more complete information about the workings of and uses for partitioning in MySQL, as well as additional examples of table creation and other statements relating to MySQL partitioning.

If used, a *partition_options* clause begins with PARTITION BY. This clause contains the function that is used to determine the partition; the function returns an integer value ranging from 1 to *num*, where *num* is the number of partitions. (The maximum number of user-defined partitions which a table may contain is 1024; the number of subpartitions—discussed later in this section—is included in this maximum.) The choices that are available for this function in MySQL 5.7 are shown in the following list:

- HASH(*expr*): Hashes one or more columns to create a key for placing and locating rows. *expr* is an expression using one or more table columns. This can be any valid MySQL expression (including MySQL functions) that yields a single integer value. For example, these are both valid CREATE TABLE statements using PARTITION BY HASH:

```
CREATE TABLE t1 (col1 INT, col2 CHAR(5))
    PARTITION BY HASH(col1);

CREATE TABLE t1 (col1 INT, col2 CHAR(5), col3 DATETIME)
    PARTITION BY HASH ( YEAR(col3) );
```

You may not use either VALUES LESS THAN or VALUES IN clauses with PARTITION BY HASH.

PARTITION BY HASH uses the remainder of *expr* divided by the number of partitions (that is, the modulus). For examples and additional information, see Section 17.2.4, “HASH Partitioning”.

The LINEAR keyword entails a somewhat different algorithm. In this case, the number of the partition in which a row is stored is calculated as the result of one or more logical AND operations. For discussion and examples of linear hashing, see Section 17.2.4.1, “LINEAR HASH Partitioning”.

- KEY(*column_list*): This is similar to HASH, except that MySQL supplies the hashing function so as to guarantee an even data distribution. The *column_list* argument is simply a list of 1 or more table columns (maximum: 16). This example shows a simple table partitioned by key, with 4 partitions:

```
CREATE TABLE tk (col1 INT, col2 CHAR(5), col3 DATE)
    PARTITION BY KEY(col3)
    PARTITIONS 4;
```

For tables that are partitioned by key, you can employ linear partitioning by using the LINEAR keyword. This has the same effect as with tables that are partitioned by HASH. That is, the partition number is found using the & operator rather than the modulus (see Section 17.2.4.1, “LINEAR HASH Partitioning”, and Section 17.2.5, “KEY Partitioning”, for details). This example uses linear partitioning by key to distribute data between 5 partitions:

```
CREATE TABLE tk (col1 INT, col2 CHAR(5), col3 DATE)
    PARTITION BY LINEAR KEY(col3)
    PARTITIONS 5;
```

The ALGORITHM={1|2} option is supported with [SUB]PARTITION BY [LINEAR] KEY beginning with MySQL 5.7.1. ALGORITHM=1 causes the server to use the same key-hashing functions as MySQL 5.1; ALGORITHM=2 means that the server employs the key-hashing functions implemented and used by default for new KEY partitioned tables in MySQL 5.5 and later. (Partitioned tables created with the key-hashing functions employed in MySQL 5.5 and later cannot be used by a MySQL 5.1 server.) Not specifying the option has the same effect as using ALGORITHM=2. This option is intended for use chiefly when upgrading or downgrading [LINEAR] KEY partitioned tables between MySQL 5.1 and later MySQL versions, or for creating tables partitioned by KEY or LINEAR KEY on a MySQL 5.5 or later server which can be used on a MySQL 5.1 server. For more information, see Section 13.1.6.1, “ALTER TABLE Partition Operations”.

mysqldump in MySQL 5.7 (and later) writes this option encased in versioned comments, like this:

```
CREATE TABLE t1 (a INT)
/*!50100 PARTITION BY KEY */ /*!50611 ALGORITHM = 1 */ /*!50100 ()
      PARTITIONS 3 */
```

This causes MySQL 5.6.10 and earlier servers to ignore the option, which would otherwise cause a syntax error in those versions. If you plan to load a dump made on a MySQL 5.7 server where you use tables that are partitioned or subpartitioned by KEY into a MySQL 5.6 server previous to version 5.6.11, be sure to consult Upgrading from MySQL 5.5 to 5.6, before proceeding. (The information found there also applies if you are loading a dump containing KEY partitioned or subpartitioned tables made from a MySQL 5.7—actually 5.6.11 or later—server into a MySQL 5.5.30 or earlier server.)

Also in MySQL 5.6.11 and later, ALGORITHM=1 is shown when necessary in the output of SHOW CREATE TABLE using versioned comments in the same manner as mysqldump. ALGORITHM=2 is always omitted from SHOW CREATE TABLE output, even if this option was specified when creating the original table.

You may not use either VALUES LESS THAN or VALUES IN clauses with PARTITION BY KEY.

- RANGE(*expr*): In this case, *expr* shows a range of values using a set of VALUES LESS THAN operators. When using range partitioning, you must define at least one partition using VALUES LESS THAN. You cannot use VALUES IN with range partitioning.

  Note

  For tables partitioned by RANGE, VALUES LESS THAN must be used with either an integer literal value or an expression that evaluates to a single integer value. In MySQL 5.7, you can overcome this limitation in a table that is defined using PARTITION BY RANGE COLUMNS, as described later in this section.

  Suppose that you have a table that you wish to partition on a column containing year values, according to the following scheme.

Partition Number:	Years Range:
0	1990 and earlier
1	1991 to 1994
2	1995 to 1998
3	1999 to 2002
4	2003 to 2005
5	2006 and later

  A table implementing such a partitioning scheme can be realized by the CREATE TABLE statement shown here:

```
CREATE TABLE t1 (
    year_col  INT,
    some_data INT
)
PARTITION BY RANGE (year_col) (
    PARTITION p0 VALUES LESS THAN (1991),
    PARTITION p1 VALUES LESS THAN (1995),
    PARTITION p2 VALUES LESS THAN (1999),
    PARTITION p3 VALUES LESS THAN (2002),
    PARTITION p4 VALUES LESS THAN (2006),
    PARTITION p5 VALUES LESS THAN MAXVALUE
);
```

  PARTITION ... VALUES LESS THAN ... statements work in a consecutive fashion. VALUES LESS THAN MAXVALUE works to specify “leftover” values that are greater than the maximum value otherwise specified.

  Note that VALUES LESS THAN clauses work sequentially in a manner similar to that of the case portions of a switch ... case block (as found in many programming languages such as C, Java, and PHP). That is, the clauses must be arranged in such a way that the upper limit specified in each successive VALUES LESS THAN is greater than that of the previous one, with the one referencing MAXVALUE coming last of all in the list.

- RANGE COLUMNS(*column_list*): This variant on RANGE facilitates partition pruning for queries using range conditions on multiple columns (that is, having conditions such as WHERE a = 1 AND b < 10 or

`WHERE a = 1 AND b = 10 AND c < 10`). It enables you to specify value ranges in multiple columns by using a list of columns in the `COLUMNS` clause and a set of column values in each `PARTITION ... VALUES LESS THAN (`*`value_list`*`)` partition definition clause. (In the simplest case, this set consists of a single column.) The maximum number of columns that can be referenced in the *`column_list`* and *`value_list`* is 16.

The *`column_list`* used in the `COLUMNS` clause may contain only names of columns; each column in the list must be one of the following MySQL data types: the integer types; the string types; and time or date column types. Columns using `BLOB`, `TEXT`, `SET`, `ENUM`, `BIT`, or spatial data types are not permitted; columns that use floating-point number types are also not permitted. You also may not use functions or arithmetic expressions in the `COLUMNS` clause.

The `VALUES LESS THAN` clause used in a partition definition must specify a literal value for each column that appears in the `COLUMNS()` clause; that is, the list of values used for each `VALUES LESS THAN` clause must contain the same number of values as there are columns listed in the `COLUMNS` clause. An attempt to use more or fewer values in a `VALUES LESS THAN` clause than there are in the `COLUMNS` clause causes the statement to fail with the error `Inconsistency in usage of column lists for partitioning...`. You cannot use `NULL` for any value appearing in `VALUES LESS THAN`. It is possible to use `MAXVALUE` more than once for a given column other than the first, as shown in this example:

```
CREATE TABLE rc (
    a INT NOT NULL,
    b INT NOT NULL
)
PARTITION BY RANGE COLUMNS(a,b) (
    PARTITION p0 VALUES LESS THAN (10,5),
    PARTITION p1 VALUES LESS THAN (20,10),
    PARTITION p2 VALUES LESS THAN (MAXVALUE,15),
    PARTITION p3 VALUES LESS THAN (MAXVALUE,MAXVALUE)
);
```

Each value used in a `VALUES LESS THAN` value list must match the type of the corresponding column exactly; no conversion is made. For example, you cannot use the string `'1'` for a value that matches a column that uses an integer type (you must use the numeral `1` instead), nor can you use the numeral `1` for a value that matches a column that uses a string type (in such a case, you must use a quoted string: `'1'`).

For more information, see Section 17.2.1, “RANGE Partitioning”, and Section 17.4, “Partition Pruning”.

- `LIST(`*`expr`*`)`: This is useful when assigning partitions based on a table column with a restricted set of possible values, such as a state or country code. In such a case, all rows pertaining to a certain state or country can be assigned to a single partition, or a partition can be reserved for a certain set of states or countries. It is similar to `RANGE`, except that only `VALUES IN` may be used to specify permissible values for each partition.

  `VALUES IN` is used with a list of values to be matched. For instance, you could create a partitioning scheme such as the following:

  ```
  CREATE TABLE client_firms (
      id   INT,
      name VARCHAR(35)
  )
  PARTITION BY LIST (id) (
      PARTITION r0 VALUES IN (1, 5, 9, 13, 17, 21),
      PARTITION r1 VALUES IN (2, 6, 10, 14, 18, 22),
      PARTITION r2 VALUES IN (3, 7, 11, 15, 19, 23),
  ```

```
    PARTITION r3 VALUES IN (4, 8, 12, 16, 20, 24)
);
```

When using list partitioning, you must define at least one partition using `VALUES IN`. You cannot use `VALUES LESS THAN` with `PARTITION BY LIST`.

> **Note**
>
> For tables partitioned by `LIST`, the value list used with `VALUES IN` must consist of integer values only. In MySQL 5.7, you can overcome this limitation using partitioning by `LIST COLUMNS`, which is described later in this section.

- `LIST COLUMNS(column_list)`: This variant on `LIST` facilitates partition pruning for queries using comparison conditions on multiple columns (that is, having conditions such as `WHERE a = 5 AND b = 5` or `WHERE a = 1 AND b = 10 AND c = 5`). It enables you to specify values in multiple columns by using a list of columns in the `COLUMNS` clause and a set of column values in each `PARTITION ... VALUES IN (value_list)` partition definition clause.

  The rules governing regarding data types for the column list used in `LIST COLUMNS(column_list)` and the value list used in `VALUES IN(value_list)` are the same as those for the column list used in `RANGE COLUMNS(column_list)` and the value list used in `VALUES LESS THAN(value_list)`, respectively, except that in the `VALUES IN` clause, `MAXVALUE` is not permitted, and you may use `NULL`.

  There is one important difference between the list of values used for `VALUES IN` with `PARTITION BY LIST COLUMNS` as opposed to when it is used with `PARTITION BY LIST`. When used with `PARTITION BY LIST COLUMNS`, each element in the `VALUES IN` clause must be a *set* of column values; the number of values in each set must be the same as the number of columns used in the `COLUMNS` clause, and the data types of these values must match those of the columns (and occur in the same order). In the simplest case, the set consists of a single column. The maximum number of columns that can be used in the `column_list` and in the elements making up the `value_list` is 16.

  The table defined by the following `CREATE TABLE` statement provides an example of a table using `LIST COLUMNS` partitioning:

```
CREATE TABLE lc (
    a INT NULL,
    b INT NULL
)
PARTITION BY LIST COLUMNS(a,b) (
    PARTITION p0 VALUES IN( (0,0), (NULL,NULL) ),
    PARTITION p1 VALUES IN( (0,1), (0,2), (0,3), (1,1), (1,2) ),
    PARTITION p2 VALUES IN( (1,0), (2,0), (2,1), (3,0), (3,1) ),
    PARTITION p3 VALUES IN( (1,3), (2,2), (2,3), (3,2), (3,3) )
);
```

- The number of partitions may optionally be specified with a `PARTITIONS num` clause, where `num` is the number of partitions. If both this clause *and* any `PARTITION` clauses are used, `num` must be equal to the total number of any partitions that are declared using `PARTITION` clauses.

> **Note**
>
> Whether or not you use a `PARTITIONS` clause in creating a table that is partitioned by `RANGE` or `LIST`, you must still include at least one `PARTITION VALUES` clause in the table definition (see below).

- A partition may optionally be divided into a number of subpartitions. This can be indicated by using the optional `SUBPARTITION BY` clause. Subpartitioning may be done by `HASH` or `KEY`. Either of these may be LINEAR. These work in the same way as previously described for the equivalent partitioning types. (It is not possible to subpartition by LIST or RANGE.)

The number of subpartitions can be indicated using the SUBPARTITIONS keyword followed by an integer value.

- Rigorous checking of the value used in PARTITIONS or SUBPARTITIONS clauses is applied and this value must adhere to the following rules:

  - The value must be a positive, nonzero integer.

  - No leading zeros are permitted.

  - The value must be an integer literal, and cannot not be an expression. For example, PARTITIONS 0.2E+01 is not permitted, even though 0.2E+01 evaluates to 2. (Bug #15890)

Note

The expression (*expr*) used in a PARTITION BY clause cannot refer to any columns not in the table being created; such references are specifically not permitted and cause the statement to fail with an error. (Bug #29444)

Each partition may be individually defined using a *partition_definition* clause. The individual parts making up this clause are as follows:

- PARTITION *partition_name*: This specifies a logical name for the partition.

- A VALUES clause: For range partitioning, each partition must include a VALUES LESS THAN clause; for list partitioning, you must specify a VALUES IN clause for each partition. This is used to determine which rows are to be stored in this partition. See the discussions of partitioning types in Chapter 17, *Partitioning*, for syntax examples.

- An optional COMMENT clause may be used to specify a string that describes the partition. Example:

```
COMMENT = 'Data for the years previous to 1999'
```

  The maximum length for a partition comment is 1024 characters.

- DATA DIRECTORY and INDEX DIRECTORY may be used to indicate the directory where, respectively, the data and indexes for this partition are to be stored. Both the *data_dir* and the *index_dir* must be absolute system path names. Example:

```
CREATE TABLE th (id INT, name VARCHAR(30), adate DATE)
PARTITION BY LIST(YEAR(adate))
(
  PARTITION p1999 VALUES IN (1995, 1999, 2003)
    DATA DIRECTORY = '/var/appdata/95/data'
    INDEX DIRECTORY = '/var/appdata/95/idx',
  PARTITION p2000 VALUES IN (1996, 2000, 2004)
    DATA DIRECTORY = '/var/appdata/96/data'
    INDEX DIRECTORY = '/var/appdata/96/idx',
  PARTITION p2001 VALUES IN (1997, 2001, 2005)
    DATA DIRECTORY = '/var/appdata/97/data'
    INDEX DIRECTORY = '/var/appdata/97/idx',
  PARTITION p2002 VALUES IN (1998, 2002, 2006)
    DATA DIRECTORY = '/var/appdata/98/data'
    INDEX DIRECTORY = '/var/appdata/98/idx'
);
```

DATA DIRECTORY and INDEX DIRECTORY behave in the same way as in the CREATE TABLE statement's *table_option* clause as used for MyISAM tables.

One data directory and one index directory may be specified per partition. If left unspecified, the data and indexes are stored by default in the table's database directory.

On Windows, the DATA DIRECTORY and INDEX DIRECTORY options are not supported for individual partitions or subpartitions of MyISAM tables, and the INDEX DIRECTORY option is not supported for individual partitions or subpartitions of InnoDB tables. These options are ignored on Windows, except that a warning is generated. (Bug #30459)

> **Note**
>
> The DATA DIRECTORY and INDEX DIRECTORY options are ignored for creating partitioned tables if NO_DIR_IN_CREATE is in effect. (Bug #24633)

- MAX_ROWS and MIN_ROWS may be used to specify, respectively, the maximum and minimum number of rows to be stored in the partition. The values for *max_number_of_rows* and *min_number_of_rows* must be positive integers. As with the table-level options with the same names, these act only as “suggestions” to the server and are not hard limits.

- The partitioning handler accepts a [STORAGE] ENGINE option for both PARTITION and SUBPARTITION. Currently, the only way in which this can be used is to set all partitions or all subpartitions to the same storage engine, and an attempt to set different storage engines for partitions or subpartitions in the same table will give rise to the error ERROR 1469 (HY000): The mix of handlers in the partitions is not permitted in this version of MySQL. We expect to lift this restriction on partitioning in a future MySQL release.

- The partition definition may optionally contain one or more *subpartition_definition* clauses. Each of these consists at a minimum of the SUBPARTITION *name*, where *name* is an identifier for the subpartition. Except for the replacement of the PARTITION keyword with SUBPARTITION, the syntax for a subpartition definition is identical to that for a partition definition.

  Subpartitioning must be done by HASH or KEY, and can be done only on RANGE or LIST partitions. See Section 17.2.6, “Subpartitioning”.

Partitions can be modified, merged, added to tables, and dropped from tables. For basic information about the MySQL statements to accomplish these tasks, see Section 13.1.6, “ALTER TABLE Syntax”. For more detailed descriptions and examples, see Section 17.3, “Partition Management”.

> **Important**
>
> The original CREATE TABLE statement, including all specifications and table options are stored by MySQL when the table is created. The information is retained so that if you change storage engines, collations or other settings using an ALTER TABLE statement, the original table options specified are retained. This enables you to change between InnoDB and MyISAM table types even though the row formats supported by the two engines are different.
>
> Because the text of the original statement is retained, but due to the way that certain values and options may be silently reconfigured (such as the ROW_FORMAT), the active table definition (accessible through DESCRIBE or with SHOW TABLE STATUS) and the table creation string (accessible through SHOW CREATE TABLE) will report different values.

Cloning or Copying a Table

You can create one table from another by adding a SELECT statement at the end of the CREATE TABLE statement:

```
CREATE TABLE new_tbl SELECT * FROM orig_tbl;
```

For more information, see Section 13.1.14.1, “CREATE TABLE ... SELECT Syntax”.

Use LIKE to create an empty table based on the definition of another table, including any column attributes and indexes defined in the original table:

```
CREATE TABLE new_tbl LIKE orig_tbl;
```

The copy is created using the same version of the table storage format as the original table. The SELECT privilege is required on the original table.

LIKE works only for base tables, not for views.

Important

You cannot execute CREATE TABLE or CREATE TABLE ... LIKE while a LOCK TABLES statement is in effect.

CREATE TABLE ... LIKE makes the same checks as CREATE TABLE and does not just copy the .frm file. This means that if the current SQL mode is different from the mode in effect when the original table was created, the table definition might be considered invalid for the new mode and the statement will fail.

CREATE TABLE ... LIKE does not preserve any DATA DIRECTORY or INDEX DIRECTORY table options that were specified for the original table, or any foreign key definitions.

If the original table is a TEMPORARY table, CREATE TABLE ... LIKE does not preserve TEMPORARY. To create a TEMPORARY destination table, use CREATE TEMPORARY TABLE ... LIKE.

13.1.14.1 CREATE TABLE ... SELECT Syntax

You can create one table from another by adding a SELECT statement at the end of the CREATE TABLE statement:

```
CREATE TABLE new_tbl [AS] SELECT * FROM orig_tbl;
```

MySQL creates new columns for all elements in the SELECT. For example:

```
mysql> CREATE TABLE test (a INT NOT NULL AUTO_INCREMENT,
    ->        PRIMARY KEY (a), KEY(b))
    ->        ENGINE=MyISAM SELECT b,c FROM test2;
```

This creates a MyISAM table with three columns, a, b, and c. The ENGINE option is part of the CREATE TABLE statement, and should not be used following the SELECT; this would result in a syntax error. The same is true for other CREATE TABLE options such as CHARSET.

Notice that the columns from the SELECT statement are appended to the right side of the table, not overlapped onto it. Take the following example:

```
mysql> SELECT * FROM foo;
+---+
| n |
+---+
| 1 |
+---+

mysql> CREATE TABLE bar (m INT) SELECT n FROM foo;
Query OK, 1 row affected (0.02 sec)
Records: 1  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM bar;
+------+---+
| m    | n |
+------+---+
| NULL | 1 |
+------+---+
1 row in set (0.00 sec)
```

For each row in table foo, a row is inserted in bar with the values from foo and default values for the new columns.

In a table resulting from CREATE TABLE ... SELECT, columns named only in the CREATE TABLE part come first. Columns named in both parts or only in the SELECT part come after that. The data type of SELECT columns can be overridden by also specifying the column in the CREATE TABLE part.

If any errors occur while copying the data to the table, it is automatically dropped and not created.

You can precede the SELECT by IGNORE or REPLACE to indicate how to handle rows that duplicate unique key values. With IGNORE, new rows that duplicate an existing row on a unique key value are discarded. With REPLACE, new rows replace rows that have the same unique key value. If neither IGNORE nor REPLACE is specified, duplicate unique key values result in an error.

Because the ordering of the rows in the underlying SELECT statements cannot always be determined, CREATE TABLE ... IGNORE SELECT and CREATE TABLE ... REPLACE SELECT statements are flagged as unsafe for statement-based replication. With this change, such statements produce a warning in the log when using statement-based mode and are logged using the row-based format when using MIXED mode. See also Section 16.1.2.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”.

CREATE TABLE ... SELECT does not automatically create any indexes for you. This is done intentionally to make the statement as flexible as possible. If you want to have indexes in the created table, you should specify these before the SELECT statement:

```
mysql> CREATE TABLE bar (UNIQUE (n)) SELECT n FROM foo;
```

Some conversion of data types might occur. For example, the AUTO_INCREMENT attribute is not preserved, and VARCHAR columns can become CHAR columns. Retrained attributes are NULL (or NOT NULL) and, for those columns that have them, CHARACTER SET, COLLATION, COMMENT, and the DEFAULT clause.

When creating a table with CREATE TABLE ... SELECT, make sure to alias any function calls or expressions in the query. If you do not, the CREATE statement might fail or result in undesirable column names.

```
CREATE TABLE artists_and_works
  SELECT artist.name, COUNT(work.artist_id) AS number_of_works
  FROM artist LEFT JOIN work ON artist.id = work.artist_id
  GROUP BY artist.id;
```

You can also explicitly specify the data type for a generated column:

```
CREATE TABLE foo (a TINYINT NOT NULL) SELECT b+1 AS a FROM bar;
```

For CREATE TABLE ... SELECT, if IF NOT EXISTS is given and the target table exists, nothing is inserted into the destination table, and the statement is not logged.

To ensure that the binary log can be used to re-create the original tables, MySQL does not permit concurrent inserts during CREATE TABLE ... SELECT.

You cannot use FOR UPDATE as part of the SELECT in a statement such as CREATE TABLE *new_table* SELECT ... FROM *old_table* .... If you attempt to do so, the statement fails.

13.1.14.2 Using FOREIGN KEY Constraints

MySQL supports foreign keys, which let you cross-reference related data across tables, and foreign key constraints, which help keep this spread-out data consistent. The essential syntax for a foreign key constraint definition in a CREATE TABLE or ALTER TABLE statement looks like this:

```
[CONSTRAINT [symbol]] FOREIGN KEY
    [index_name] (index_col_name, ...)
    REFERENCES tbl_name (index_col_name,...)
    [ON DELETE reference_option]
    [ON UPDATE reference_option]

reference_option:
    RESTRICT | CASCADE | SET NULL | NO ACTION
```

index_name represents a foreign key ID. If given, this is ignored if an index for the foreign key is defined explicitly. Otherwise, if MySQL creates an index for the foreign key, it uses *index_name* for the index name.

Foreign keys definitions are subject to the following conditions:

- Foreign key relationships involve a parent table that holds the central data values, and a child table with identical values pointing back to its parent. The FOREIGN KEY clause is specified in the child table. The parent and child tables must use the same storage engine. They must not be TEMPORARY tables.

- Corresponding columns in the foreign key and the referenced key must have similar data types. *The size and sign of integer types must be the same*. The length of string types need not be the same. For nonbinary (character) string columns, the character set and collation must be the same.

- MySQL requires indexes on foreign keys and referenced keys so that foreign key checks can be fast and not require a table scan. In the referencing table, there must be an index where the foreign key columns are listed as the *first* columns in the same order. Such an index is created on the referencing table automatically if it does not exist. This index might be silently dropped later, if you create another index that can be used to enforce the foreign key constraint. *index_name*, if given, is used as described previously.

- InnoDB permits a foreign key to reference any index column or group of columns. However, in the referenced table, there must be an index where the referenced columns are listed as the *first* columns in the same order.

- Index prefixes on foreign key columns are not supported. One consequence of this is that BLOB and TEXT columns cannot be included in a foreign key because indexes on those columns must always include a prefix length.

- If the CONSTRAINT *symbol* clause is given, the *symbol* value, if used, must be unique in the database. A duplicate *symbol* will result in an error similar to: ERROR 1022 (2300): Can't write;

`duplicate key in table '#sql- 464_1'`. If the clause is not given, or a *`symbol`* is not included following the `CONSTRAINT` keyword, a name for the constraint is created automatically.

- `InnoDB` does not currently support foreign keys for tables with user-defined partitioning. This includes both parent and child tables.

Referential Actions

This section describes how foreign keys help guarantee referential integrity.

For storage engines supporting foreign keys, MySQL rejects any `INSERT` or `UPDATE` operation that attempts to create a foreign key value in a child table if there is no a matching candidate key value in the parent table.

When an `UPDATE` or `DELETE` operation affects a key value in the parent table that has matching rows in the child table, the result depends on the *referential action* specified using `ON UPDATE` and `ON DELETE` subclauses of the `FOREIGN KEY` clause. MySQL supports five options regarding the action to be taken, listed here:

- `CASCADE`: Delete or update the row from the parent table, and automatically delete or update the matching rows in the child table. Both `ON DELETE CASCADE` and `ON UPDATE CASCADE` are supported. Between two tables, do not define several `ON UPDATE CASCADE` clauses that act on the same column in the parent table or in the child table.

  Note

  Currently, cascaded foreign key actions do not activate triggers.

- `SET NULL`: Delete or update the row from the parent table, and set the foreign key column or columns in the child table to `NULL`. Both `ON DELETE SET NULL` and `ON UPDATE SET NULL` clauses are supported.

  If you specify a `SET NULL` action, *make sure that you have not declared the columns in the child table as `NOT NULL`*.

- `RESTRICT`: Rejects the delete or update operation for the parent table. Specifying `RESTRICT` (or `NO ACTION`) is the same as omitting the `ON DELETE` or `ON UPDATE` clause.

- `NO ACTION`: A keyword from standard SQL. In MySQL, equivalent to `RESTRICT`. The MySQL Server rejects the delete or update operation for the parent table if there is a related foreign key value in the referenced table. Some database systems have deferred checks, and `NO ACTION` is a deferred check. In MySQL, foreign key constraints are checked immediately, so `NO ACTION` is the same as `RESTRICT`.

- `SET DEFAULT`: This action is recognized by the MySQL parser, but `InnoDB` rejects table definitions containing `ON DELETE SET DEFAULT` or `ON UPDATE SET DEFAULT` clauses.

For an `ON DELETE` or `ON UPDATE` that is not specified, the default action is always `RESTRICT`.

MySQL supports foreign key references between one column and another within a table. (A column cannot have a foreign key reference to itself.) In these cases, “child table records” really refers to dependent records within the same table.

Examples of Foreign Key Clauses

Here is a simple example that relates `parent` and `child` tables through a single-column foreign key:

```
CREATE TABLE parent (
    id INT NOT NULL,
    PRIMARY KEY (id)
) ENGINE=INNODB;

CREATE TABLE child (
    id INT,
    parent_id INT,
    INDEX par_ind (parent_id),
    FOREIGN KEY (parent_id)
        REFERENCES parent(id)
        ON DELETE CASCADE
) ENGINE=INNODB;
```

A more complex example in which a product_order table has foreign keys for two other tables. One foreign key references a two-column index in the product table. The other references a single-column index in the customer table:

```
CREATE TABLE product (
    category INT NOT NULL, id INT NOT NULL,
    price DECIMAL,
    PRIMARY KEY(category, id)
)   ENGINE=INNODB;

CREATE TABLE customer (
    id INT NOT NULL,
    PRIMARY KEY (id)
)   ENGINE=INNODB;

CREATE TABLE product_order (
    no INT NOT NULL AUTO_INCREMENT,
    product_category INT NOT NULL,
    product_id INT NOT NULL,
    customer_id INT NOT NULL,

    PRIMARY KEY(no),
    INDEX (product_category, product_id),
    INDEX (customer_id),

    FOREIGN KEY (product_category, product_id)
      REFERENCES product(category, id)
      ON UPDATE CASCADE ON DELETE RESTRICT,

    FOREIGN KEY (customer_id)
      REFERENCES customer(id)
)   ENGINE=INNODB;
```

Adding foreign keys

You can add a new foreign key constraint to an existing table by using ALTER TABLE. The syntax relating to foreign keys for this statement is shown here:

```
ALTER TABLE tbl_name
    ADD [CONSTRAINT [symbol]] FOREIGN KEY
    [index_name] (index_col_name, ...)
    REFERENCES tbl_name (index_col_name,...)
    [ON DELETE reference_option]
    [ON UPDATE reference_option]
```

The foreign key can be self referential (referring to the same table). When you add a foreign key constraint to a table using ALTER TABLE, *remember to create the required indexes first.*

Dropping Foreign Keys

You can also use ALTER TABLE to drop foreign keys, using the syntax shown here:

```
ALTER TABLE tbl_name DROP FOREIGN KEY fk_symbol;
```

If the FOREIGN KEY clause included a CONSTRAINT name when you created the foreign key, you can refer to that name to drop the foreign key. Otherwise, the *fk_symbol* value is generated internally when the foreign key is created. To find out the symbol value when you want to drop a foreign key, use a SHOW CREATE TABLE statement, as shown here:

```
mysql> SHOW CREATE TABLE ibtest11c\G
*************************** 1. row ***************************
       Table: ibtest11c
Create Table: CREATE TABLE `ibtest11c` (
  `A` int(11) NOT NULL auto_increment,
  `D` int(11) NOT NULL default '0',
  `B` varchar(200) NOT NULL default '',
  `C` varchar(175) default NULL,
  PRIMARY KEY  (`A`,`D`,`B`),
  KEY `B` (`B`,`C`),
  KEY `C` (`C`),
  CONSTRAINT `0_38775` FOREIGN KEY (`A`, `D`)
REFERENCES `ibtest11a` (`A`, `D`)
ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT `0_38776` FOREIGN KEY (`B`, `C`)
REFERENCES `ibtest11a` (`B`, `C`)
ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=INNODB CHARSET=latin1
1 row in set (0.01 sec)

mysql> ALTER TABLE ibtest11c DROP FOREIGN KEY `0_38775`;
```

Prior to MySQL 5.6.6, adding and dropping a foreign key in the same ALTER TABLE statement may be problematic in some cases and is therefore unsupported. Separate statements should be used for each operation. As of MySQL 5.6.6, adding and dropping a foreign key in the same ALTER TABLE statement is supported for ALTER TABLE ... ALGORITHM=INPLACE but remains unsupported for ALTER TABLE ... ALGORITHM=COPY.

In MySQL 5.7, the server prohibits changes to foreign key columns with the potential to cause loss of referential integrity. A workaround is to use ALTER TABLE ... DROP FOREIGN KEY before changing the column definition and ALTER TABLE ... ADD FOREIGN KEY afterward.

Foreign Keys and Other MySQL Statements

Table and column identifiers in a FOREIGN KEY ... REFERENCES ... clause can be quoted within backticks (`). Alternatively, double quotation marks (") can be used if the ANSI_QUOTES SQL mode is enabled. The setting of the lower_case_table_names system variable is also taken into account.

You can view a child table's foreign key definitions as part of the output of the SHOW CREATE TABLE statement:

```
SHOW CREATE TABLE tbl_name;
```

You can also obtain information about foreign keys by querying the INFORMATION_SCHEMA.KEY_COLUMN_USAGE table.

You can find information about foreign keys used by InnoDB tables in the INNODB_SYS_FOREIGN and INNODB_SYS_FOREIGN_COLS tables, also in the INFORMATION_SCHEMA database.

`mysqldump` produces correct definitions of tables in the dump file, including the foreign keys for child tables.

To make it easier to reload dump files for tables that have foreign key relationships, `mysqldump` automatically includes a statement in the dump output to set `foreign_key_checks` to 0. This avoids problems with tables having to be reloaded in a particular order when the dump is reloaded. It is also possible to set this variable manually:

```
mysql> SET foreign_key_checks = 0;
mysql> SOURCE dump_file_name;
mysql> SET foreign_key_checks = 1;
```

This enables you to import the tables in any order if the dump file contains tables that are not correctly ordered for foreign keys. It also speeds up the import operation. Setting `foreign_key_checks` to 0 can also be useful for ignoring foreign key constraints during `LOAD DATA` and `ALTER TABLE` operations. However, even if `foreign_key_checks = 0`, MySQL does not permit the creation of a foreign key constraint where a column references a nonmatching column type. Also, if a table has foreign key constraints, `ALTER TABLE` cannot be used to alter the table to use another storage engine. To change the storage engine, you must drop any foreign key constraints first.

You cannot issue `DROP TABLE` for a table that is referenced by a `FOREIGN KEY` constraint, unless you do `SET foreign_key_checks = 0`. When you drop a table, any constraints that were defined in the statement used to create that table are also dropped.

If you re-create a table that was dropped, it must have a definition that conforms to the foreign key constraints referencing it. It must have the correct column names and types, and it must have indexes on the referenced keys, as stated earlier. If these are not satisfied, MySQL returns Error 1005 and refers to Error 150 in the error message, which means that a foreign key constraint was not correctly formed. Similarly, if an `ALTER TABLE` fails due to Error 150, this means that a foreign key definition would be incorrectly formed for the altered table.

For `InnoDB` tables, you can obtain a detailed explanation of the most recent `InnoDB` foreign key error in the MySQL Server, by checking the output of `SHOW ENGINE INNODB STATUS`.

Important

For users familiar with the ANSI/ISO SQL Standard, please note that no storage engine, including `InnoDB`, recognizes or enforces the `MATCH` clause used in referential-integrity constraint definitions. Use of an explicit `MATCH` clause will not have the specified effect, and also causes `ON DELETE` and `ON UPDATE` clauses to be ignored. For these reasons, specifying `MATCH` should be avoided.

The `MATCH` clause in the SQL standard controls how `NULL` values in a composite (multiple-column) foreign key are handled when comparing to a primary key. MySQL essentially implements the semantics defined by `MATCH SIMPLE`, which permit a foreign key to be all or partially `NULL`. In that case, the (child table) row containing such a foreign key is permitted to be inserted, and does not match any row in the referenced (parent) table. It is possible to implement other semantics using triggers.

Additionally, MySQL requires that the referenced columns be indexed for performance reasons. However, the system does not enforce a requirement that the referenced columns be `UNIQUE` or be declared `NOT NULL`. The handling of foreign key references to nonunique keys or keys that contain `NULL` values is not well defined for operations such as `UPDATE` or `DELETE CASCADE`. You are advised to

> use foreign keys that reference only UNIQUE (including PRIMARY) and NOT NULL keys.
>
> Furthermore, MySQL does not recognize or support “inline REFERENCES specifications” (as defined in the SQL standard) where the references are defined as part of the column specification. MySQL accepts REFERENCES clauses only when specified as part of a separate FOREIGN KEY specification. For storage engines that do not support foreign keys (such as MyISAM), MySQL Server parses and ignores foreign key specifications.

13.1.14.3 Silent Column Specification Changes

In some cases, MySQL silently changes column specifications from those given in a CREATE TABLE or ALTER TABLE statement. These might be changes to a data type, to attributes associated with a data type, or to an index specification.

All changes are subject to the internal row-size limit of 65,535 bytes, which may cause some attempts at data type changes to fail. See Section E.10.4, “Limits on Table Column Count and Row Size”.

- Columns that are part of a PRIMARY KEY are made NOT NULL even if not declared that way.
- Trailing spaces are automatically deleted from ENUM and SET member values when the table is created.
- MySQL maps certain data types used by other SQL database vendors to MySQL types. See Section 11.8, “Using Data Types from Other Database Engines”.
- If you include a USING clause to specify an index type that is not permitted for a given storage engine, but there is another index type available that the engine can use without affecting query results, the engine uses the available type.
- If strict SQL mode is not enabled, a VARCHAR column with a length specification greater than 65535 is converted to TEXT, and a VARBINARY column with a length specification greater than 65535 is converted to BLOB. Otherwise, an error occurs in either of these cases.
- Specifying the CHARACTER SET binary attribute for a character data type causes the column to be created as the corresponding binary data type: CHAR becomes BINARY, VARCHAR becomes VARBINARY, and TEXT becomes BLOB. For the ENUM and SET data types, this does not occur; they are created as declared. Suppose that you specify a table using this definition:

  ```
  CREATE TABLE t
  (
    c1 VARCHAR(10) CHARACTER SET binary,
    c2 TEXT CHARACTER SET binary,
    c3 ENUM('a','b','c') CHARACTER SET binary
  );
  ```

  The resulting table has this definition:

  ```
  CREATE TABLE t
  (
    c1 VARBINARY(10),
    c2 BLOB,
    c3 ENUM('a','b','c') CHARACTER SET binary
  );
  ```

To see whether MySQL used a data type other than the one you specified, issue a DESCRIBE or SHOW CREATE TABLE statement after creating or altering the table.

Certain other data type changes can occur if you compress a table using myisampack. See Section 14.3.3.3, “Compressed Table Characteristics”.

13.1.15 CREATE TRIGGER Syntax

```
CREATE
    [DEFINER = { user | CURRENT_USER }]
    TRIGGER trigger_name
    trigger_time trigger_event
    ON tbl_name FOR EACH ROW
    [trigger_order]
    trigger_body

trigger_time: { BEFORE | AFTER }

trigger_event: { INSERT | UPDATE | DELETE }

trigger_order: { FOLLOWS | PRECEDES } other_trigger_name
```

This statement creates a new trigger. A trigger is a named database object that is associated with a table, and that activates when a particular event occurs for the table. The trigger becomes associated with the table named *tbl_name*, which must refer to a permanent table. You cannot associate a trigger with a TEMPORARY table or a view.

Trigger names exist in the schema namespace, meaning that all triggers must have unique names within a schema. Triggers in different schemas can have the same name.

This section describes CREATE TRIGGER syntax. For additional discussion, see Section 18.3.1, “Trigger Syntax and Examples”.

CREATE TRIGGER requires the TRIGGER privilege for the table associated with the trigger. The statement might also require the SUPER privilege, depending on the DEFINER value, as described later in this section. If binary logging is enabled, CREATE TRIGGER might require the SUPER privilege, as described in Section 18.7, “Binary Logging of Stored Programs”.

The DEFINER clause determines the security context to be used when checking access privileges at trigger activation time, as described later in this section.

trigger_time is the trigger action time. It can be BEFORE or AFTER to indicate that the trigger activates before or after each row to be modified.

trigger_event indicates the kind of operation that activates the trigger. These *trigger_event* values are permitted:

- INSERT: The trigger activates whenever a new row is inserted into the table; for example, through INSERT, LOAD DATA, and REPLACE statements.

- UPDATE: The trigger activates whenever a row is modified; for example, through UPDATE statements.

- DELETE: The trigger activates whenever a row is deleted from the table; for example, through DELETE and REPLACE statements. DROP TABLE and TRUNCATE TABLE statements on the table do *not* activate this trigger, because they do not use DELETE. Dropping a partition does not activate DELETE triggers, either.

The *trigger_event* does not represent a literal type of SQL statement that activates the trigger so much as it represents a type of table operation. For example, an INSERT trigger activates not only for INSERT statements but also LOAD DATA statements because both statements insert rows into a table.

A potentially confusing example of this is the `INSERT INTO ... ON DUPLICATE KEY UPDATE ...` syntax: a `BEFORE INSERT` trigger activates for every row, followed by either an `AFTER INSERT` trigger or both the `BEFORE UPDATE` and `AFTER UPDATE` triggers, depending on whether there was a duplicate key for the row.

> **Note**
>
> Cascaded foreign key actions do not activate triggers.

As of MySQL 5.7.2, it is possible to define multiple triggers for a given table that have the same trigger event and action time. For example, you cannot have two `BEFORE UPDATE` triggers for a table. By default, triggers that have the same trigger event and action time activate in the order they were created. To affect trigger order, specify a *`trigger_order`* clause that indicates `FOLLOWS` or `PRECEDES` and the name of an existing trigger that also has the same trigger event and action time. With `FOLLOWS`, the new trigger activates after the existing trigger. With `PRECEDES`, the new trigger activates before the existing trigger.

Before MySQL 5.7.2, there cannot be multiple triggers for a given table that have the same trigger event and action time. For example, you cannot have two `BEFORE UPDATE` triggers for a table. But you can have a `BEFORE UPDATE` and a `BEFORE INSERT` trigger, or a `BEFORE UPDATE` and an `AFTER UPDATE` trigger.

`trigger_body` is the statement to execute when the trigger activates. To execute multiple statements, use the `BEGIN ... END` compound statement construct. This also enables you to use the same statements that are permitted within stored routines. See Section 13.6.1, "`BEGIN ... END` Compound-Statement Syntax". Some statements are not permitted in triggers; see Section E.1, "Restrictions on Stored Programs".

Within the trigger body, you can refer to columns in the subject table (the table associated with the trigger) by using the aliases `OLD` and `NEW`. `OLD.`*`col_name`* refers to a column of an existing row before it is updated or deleted. `NEW.`*`col_name`* refers to the column of a new row to be inserted or an existing row after it is updated.

MySQL stores the `sql_mode` system variable setting in effect when a trigger is created, and always executes the trigger body with this setting in force, *regardless of the current server SQL mode when the trigger begins executing*.

The `DEFINER` clause specifies the MySQL account to be used when checking access privileges at trigger activation time. If a *`user`* value is given, it should be a MySQL account specified as `'`*`user_name`*`'@'`*`host_name`*`'` (the same format used in the `GRANT` statement), `CURRENT_USER`, or `CURRENT_USER()`. The default `DEFINER` value is the user who executes the `CREATE TRIGGER` statement. This is the same as specifying `DEFINER = CURRENT_USER` explicitly.

If you specify the `DEFINER` clause, these rules determine the valid `DEFINER` user values:

- If you do not have the `SUPER` privilege, the only permitted *`user`* value is your own account, either specified literally or by using `CURRENT_USER`. You cannot set the definer to some other account.

- If you have the `SUPER` privilege, you can specify any syntactically valid account name. If the account does not actually exist, a warning is generated.

- Although it is possible to create a trigger with a nonexistent `DEFINER` account, it is not a good idea for such triggers to be activated until the account actually does exist. Otherwise, the behavior with respect to privilege checking is undefined.

MySQL takes the `DEFINER` user into account when checking trigger privileges as follows:

- At `CREATE TRIGGER` time, the user who issues the statement must have the `TRIGGER` privilege.

- At trigger activation time, privileges are checked against the `DEFINER` user. This user must have these privileges:
  - The `TRIGGER` privilege for the subject table.
  - The `SELECT` privilege for the subject table if references to table columns occur using `OLD.col_name` or `NEW.col_name` in the trigger body.
  - The `UPDATE` privilege for the subject table if table columns are targets of `SET NEW.col_name = value` assignments in the trigger body.
  - Whatever other privileges normally are required for the statements executed by the trigger.

For more information about trigger security, see Section 18.6, “Access Control for Stored Programs and Views”.

Within a trigger body, the `CURRENT_USER()` function returns the account used to check privileges at trigger activation time. This is the `DEFINER` user, not the user whose actions caused the trigger to be activated. For information about user auditing within triggers, see Section 6.3.14, “SQL-Based MySQL Account Activity Auditing”.

If you use `LOCK TABLES` to lock a table that has triggers, the tables used within the trigger are also locked, as described in Section 13.3.5.2, “`LOCK TABLES` and Triggers”.

For additional discussion of trigger use, see Section 18.3.1, “Trigger Syntax and Examples”.

13.1.16 `CREATE VIEW` Syntax

```
CREATE
    [OR REPLACE]
    [ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}]
    [DEFINER = { user | CURRENT_USER }]
    [SQL SECURITY { DEFINER | INVOKER }]
    VIEW view_name [(column_list)]
    AS select_statement
    [WITH [CASCADED | LOCAL] CHECK OPTION]
```

The `CREATE VIEW` statement creates a new view, or replaces an existing one if the `OR REPLACE` clause is given. If the view does not exist, `CREATE OR REPLACE VIEW` is the same as `CREATE VIEW`. If the view does exist, `CREATE OR REPLACE VIEW` is the same as `ALTER VIEW`.

The *`select_statement`* is a `SELECT` statement that provides the definition of the view. (When you select from the view, you select in effect using the `SELECT` statement.) *`select_statement`* can select from base tables or other views.

The view definition is “frozen” at creation time, so changes to the underlying tables afterward do not affect the view definition. For example, if a view is defined as `SELECT *` on a table, new columns added to the table later do not become part of the view.

The `ALGORITHM` clause affects how MySQL processes the view. The `DEFINER` and `SQL SECURITY` clauses specify the security context to be used when checking access privileges at view invocation time. The `WITH CHECK OPTION` clause can be given to constrain inserts or updates to rows in tables referenced by the view. These clauses are described later in this section.

The `CREATE VIEW` statement requires the `CREATE VIEW` privilege for the view, and some privilege for each column selected by the `SELECT` statement. For columns used elsewhere in the `SELECT` statement you must have the SELECT privilege. If the OR REPLACE clause is present, you must also have the DROP privilege for the view. CREATE VIEW might also require the SUPER privilege, depending on the DEFINER value, as described later in this section.

When a view is referenced, privilege checking occurs as described later in this section.

A view belongs to a database. By default, a new view is created in the default database. To create the view explicitly in a given database, specify the name as *db_name.view_name* when you create it:

```
mysql> CREATE VIEW test.v AS SELECT * FROM t;
```

Within a database, base tables and views share the same namespace, so a base table and a view cannot have the same name.

Columns retrieved by the SELECT statement can be simple references to table columns. They can also be expressions that use functions, constant values, operators, and so forth.

Views must have unique column names with no duplicates, just like base tables. By default, the names of the columns retrieved by the SELECT statement are used for the view column names. To define explicit names for the view columns, the optional *column_list* clause can be given as a list of comma-separated identifiers. The number of names in *column_list* must be the same as the number of columns retrieved by the SELECT statement.

Unqualified table or view names in the SELECT statement are interpreted with respect to the default database. A view can refer to tables or views in other databases by qualifying the table or view name with the proper database name.

A view can be created from many kinds of SELECT statements. It can refer to base tables or other views. It can use joins, UNION, and subqueries. The SELECT need not even refer to any tables. The following example defines a view that selects two columns from another table, as well as an expression calculated from those columns:

```
mysql> CREATE TABLE t (qty INT, price INT);
mysql> INSERT INTO t VALUES(3, 50);
mysql> CREATE VIEW v AS SELECT qty, price, qty*price AS value FROM t;
mysql> SELECT * FROM v;
+------+-------+-------+
| qty  | price | value |
+------+-------+-------+
|    3 |    50 |   150 |
+------+-------+-------+
```

A view definition is subject to the following restrictions:

- The SELECT statement cannot contain a subquery in the FROM clause.
- The SELECT statement cannot refer to system or user variables.
- Within a stored program, the definition cannot refer to program parameters or local variables.
- The SELECT statement cannot refer to prepared statement parameters.
- Any table or view referred to in the definition must exist. However, after a view has been created, it is possible to drop a table or view that the definition refers to. In this case, use of the view results in an error. To check a view definition for problems of this kind, use the CHECK TABLE statement.
- The definition cannot refer to a TEMPORARY table, and you cannot create a TEMPORARY view.

- Any tables named in the view definition must exist at definition time.
- You cannot associate a trigger with a view.
- Aliases for column names in the SELECT statement are checked against the maximum column length of 64 characters (not the maximum alias length of 256 characters).

ORDER BY is permitted in a view definition, but it is ignored if you select from a view using a statement that has its own ORDER BY.

For other options or clauses in the definition, they are added to the options or clauses of the statement that references the view, but the effect is undefined. For example, if a view definition includes a LIMIT clause, and you select from the view using a statement that has its own LIMIT clause, it is undefined which limit applies. This same principle applies to options such as ALL, DISTINCT, or SQL_SMALL_RESULT that follow the SELECT keyword, and to clauses such as INTO, FOR UPDATE, LOCK IN SHARE MODE, and PROCEDURE.

If you create a view and then change the query processing environment by changing system variables, that may affect the results that you get from the view:

```
mysql> CREATE VIEW v (mycol) AS SELECT 'abc';
Query OK, 0 rows affected (0.01 sec)

mysql> SET sql_mode = '';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT "mycol" FROM v;
+-------+
| mycol |
+-------+
| mycol |
+-------+
1 row in set (0.01 sec)

mysql> SET sql_mode = 'ANSI_QUOTES';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT "mycol" FROM v;
+-------+
| mycol |
+-------+
| abc   |
+-------+
1 row in set (0.00 sec)
```

The DEFINER and SQL SECURITY clauses determine which MySQL account to use when checking access privileges for the view when a statement is executed that references the view. The valid SQL SECURITY characteristic values are DEFINER and INVOKER. These indicate that the required privileges must be held by the user who defined or invoked the view, respectively. The default SQL SECURITY value is DEFINER.

If a *user* value is given for the DEFINER clause, it should be a MySQL account specified as '*user_name*'@'*host_name*' (the same format used in the GRANT statement), CURRENT_USER, or CURRENT_USER(). The default DEFINER value is the user who executes the CREATE VIEW statement. This is the same as specifying DEFINER = CURRENT_USER explicitly.

If you specify the DEFINER clause, these rules determine the valid DEFINER user values:

- If you do not have the SUPER privilege, the only valid *user* value is your own account, either specified literally or by using CURRENT_USER. You cannot set the definer to some other account.

- If you have the SUPER privilege, you can specify any syntactically valid account name. If the account does not actually exist, a warning is generated.
- Although it is possible to create a view with a nonexistent DEFINER account, an error occurs when the view is referenced if the SQL SECURITY value is DEFINER but the definer account does not exist.

For more information about view security, see Section 18.6, “Access Control for Stored Programs and Views”.

Within a view definition, CURRENT_USER returns the view's DEFINER value by default. For views defined with the SQL SECURITY INVOKER characteristic, CURRENT_USER returns the account for the view's invoker. For information about user auditing within views, see Section 6.3.14, “SQL-Based MySQL Account Activity Auditing”.

Within a stored routine that is defined with the SQL SECURITY DEFINER characteristic, CURRENT_USER returns the routine's DEFINER value. This also affects a view defined within such a routine, if the view definition contains a DEFINER value of CURRENT_USER.

View privileges are checked like this:

- At view definition time, the view creator must have the privileges needed to use the top-level objects accessed by the view. For example, if the view definition refers to table columns, the creator must have some privilege for each column in the select list of the definition, and the SELECT privilege for each column used elsewhere in the definition. If the definition refers to a stored function, only the privileges needed to invoke the function can be checked. The privileges required at function invocation time can be checked only as it executes: For different invocations, different execution paths within the function might be taken.
- The user who references a view must have appropriate privileges to access it (SELECT to select from it, INSERT to insert into it, and so forth.)
- When a view has been referenced, privileges for objects accessed by the view are checked against the privileges held by the view DEFINER account or invoker, depending on whether the SQL SECURITY characteristic is DEFINER or INVOKER, respectively.
- If reference to a view causes execution of a stored function, privilege checking for statements executed within the function depend on whether the function SQL SECURITY characteristic is DEFINER or INVOKER. If the security characteristic is DEFINER, the function runs with the privileges of the DEFINER account. If the characteristic is INVOKER, the function runs with the privileges determined by the view's SQL SECURITY characteristic.

Example: A view might depend on a stored function, and that function might invoke other stored routines. For example, the following view invokes a stored function f():

```
CREATE VIEW v AS SELECT * FROM t WHERE t.id = f(t.name);
```

Suppose that f() contains a statement such as this:

```
IF name IS NULL then
  CALL p1();
ELSE
  CALL p2();
END IF;
```

The privileges required for executing statements within f() need to be checked when f() executes. This might mean that privileges are needed for p1() or p2(), depending on the execution path within f().

Those privileges must be checked at runtime, and the user who must possess the privileges is determined by the `SQL SECURITY` values of the view `v` and the function `f()`.

The `DEFINER` and `SQL SECURITY` clauses for views are extensions to standard SQL. In standard SQL, views are handled using the rules for `SQL SECURITY DEFINER`. The standard says that the definer of the view, which is the same as the owner of the view's schema, gets applicable privileges on the view (for example, `SELECT`) and may grant them. MySQL has no concept of a schema "owner", so MySQL adds a clause to identify the definer. The `DEFINER` clause is an extension where the intent is to have what the standard has; that is, a permanent record of who defined the view. This is why the default `DEFINER` value is the account of the view creator.

The optional `ALGORITHM` clause is a MySQL extension to standard SQL. It affects how MySQL processes the view. `ALGORITHM` takes three values: `MERGE`, `TEMPTABLE`, or `UNDEFINED`. The default algorithm is `UNDEFINED` if no `ALGORITHM` clause is present. For more information, see Section 18.5.2, "View Processing Algorithms".

Some views are updatable. That is, you can use them in statements such as `UPDATE`, `DELETE`, or `INSERT` to update the contents of the underlying table. For a view to be updatable, there must be a one-to-one relationship between the rows in the view and the rows in the underlying table. There are also certain other constructs that make a view nonupdatable.

The `WITH CHECK OPTION` clause can be given for an updatable view to prevent inserts or updates to rows except those for which the `WHERE` clause in the *`select_statement`* is true.

In a `WITH CHECK OPTION` clause for an updatable view, the `LOCAL` and `CASCADED` keywords determine the scope of check testing when the view is defined in terms of another view. The `LOCAL` keyword restricts the `CHECK OPTION` only to the view being defined. `CASCADED` causes the checks for underlying views to be evaluated as well. When neither keyword is given, the default is `CASCADED`.

For more information about updatable views and the `WITH CHECK OPTION` clause, see Section 18.5.3, "Updatable and Insertable Views".

Views created before MySQL 5.7.3 containing `ORDER BY` *`integer`* can result in errors at view evaluation time. Consider these view definitions, which use `ORDER BY` with an ordinal number:

```
CREATE VIEW v1 AS SELECT x, y, z FROM t ORDER BY 2;
CREATE VIEW v2 AS SELECT x, 1, z FROM t ORDER BY 2;
```

In the first case, `ORDER BY 2` refers to a named column `y`. In the second case, it refers to a constant 1. For queries that select from either view fewer than 2 columns (the number named in the `ORDER BY` clause), an error occurred if the server evaluated the view using the MERGE algorithm. Examples:

```
mysql> SELECT x FROM v1;
ERROR 1054 (42S22): Unknown column '2' in 'order clause'
mysql> SELECT x FROM v2;
ERROR 1054 (42S22): Unknown column '2' in 'order clause'
```

As of MySQL 5.7.3, to handle view definitions like this, the server writes them differently into the `.frm` file that stores the view definition. This difference is visible with `SHOW CREATE VIEW`. Previously, the `.frm` file contained this for the `ORDER BY 2` clause:

```
For v1: ORDER BY 2
For v2: ORDER BY 2
```

As of 5.7.3, the `.frm` file contains this:

```
For v1: ORDER BY `t`.`y`
For v2: ORDER BY ''
```

That is, for v1, 2 is replaced by a reference to the name of the column referred to. For v2, 2 is replaced by a constant string expression (ordering by a constant has no effect, so ordering by any constant will do).

If you experience view-evaluation errors such as just described, drop and recreate the view so that the .frm file contains the updated view representation. Alternatively, for views like v2 that order by a constant value, drop and recreate the view with no ORDER BY clause.

13.1.17 DROP DATABASE Syntax

```
DROP {DATABASE | SCHEMA} [IF EXISTS] db_name
```

DROP DATABASE drops all tables in the database and deletes the database. Be *very* careful with this statement! To use DROP DATABASE, you need the DROP privilege on the database. DROP SCHEMA is a synonym for DROP DATABASE.

⚠ **Important**

When a database is dropped, user privileges on the database are *not* automatically dropped. See Section 13.7.1.4, “GRANT Syntax”.

IF EXISTS is used to prevent an error from occurring if the database does not exist.

If the default database is dropped, the default database is unset (the DATABASE() function returns NULL).

If you use DROP DATABASE on a symbolically linked database, both the link and the original database are deleted.

DROP DATABASE returns the number of tables that were removed. This corresponds to the number of .frm files removed.

The DROP DATABASE statement removes from the given database directory those files and directories that MySQL itself may create during normal operation:

- All files with the following extensions.

.BAK	.DAT	.HSH	.MRG
.MYD	.MYI	.TRG	.TRN
.db	.frm	.ibd	.ndb
.par			

- The db.opt file, if it exists.

If other files or directories remain in the database directory after MySQL removes those just listed, the database directory cannot be removed. In this case, you must remove any remaining files or directories manually and issue the DROP DATABASE statement again.

You can also drop databases with mysqladmin. See Section 4.5.2, “mysqladmin — Client for Administering a MySQL Server”.

13.1.18 DROP EVENT Syntax

```
DROP EVENT [IF EXISTS] event_name
```

This statement drops the event named *event_name*. The event immediately ceases being active, and is deleted completely from the server.

If the event does not exist, the error ERROR 1517 (HY000): Unknown event '*event_name*' results. You can override this and cause the statement to generate a warning for nonexistent events instead using IF EXISTS.

This statement requires the EVENT privilege for the schema to which the event to be dropped belongs.

13.1.19 DROP FUNCTION Syntax

The DROP FUNCTION statement is used to drop stored functions and user-defined functions (UDFs):

- For information about dropping stored functions, see Section 13.1.21, "DROP PROCEDURE and DROP FUNCTION Syntax".
- For information about dropping user-defined functions, see Section 13.7.3.2, "DROP FUNCTION Syntax".

13.1.20 DROP INDEX Syntax

```
DROP INDEX index_name ON tbl_name
    [algorithm_option | lock_option] ...

algorithm_option:
    ALGORITHM [=] {DEFAULT|INPLACE|COPY}

lock_option:
    LOCK [=] {DEFAULT|NONE|SHARED|EXCLUSIVE}
```

DROP INDEX drops the index named *index_name* from the table *tbl_name*. This statement is mapped to an ALTER TABLE statement to drop the index. See Section 13.1.6, "ALTER TABLE Syntax".

To drop a primary key, the index name is always PRIMARY, which must be specified as a quoted identifier because PRIMARY is a reserved word:

```
DROP INDEX `PRIMARY` ON t;
```

ALGORITHM and LOCK clauses may be given. These influence the table copying method and level of concurrency for reading and writing the table while its indexes are being modified. They have the same meaning as for the ALTER TABLE statement. For more information, see Section 13.1.6, "ALTER TABLE Syntax"

13.1.21 DROP PROCEDURE and DROP FUNCTION Syntax

```
DROP {PROCEDURE | FUNCTION} [IF EXISTS] sp_name
```

This statement is used to drop a stored procedure or function. That is, the specified routine is removed from the server. You must have the ALTER ROUTINE privilege for the routine. (If the automatic_sp_privileges system variable is enabled, that privilege and EXECUTE are granted automatically to the routine creator when the routine is created and dropped from the creator when the routine is dropped. See Section 18.2.2, "Stored Routines and MySQL Privileges".)

The IF EXISTS clause is a MySQL extension. It prevents an error from occurring if the procedure or function does not exist. A warning is produced that can be viewed with SHOW WARNINGS.

DROP FUNCTION is also used to drop user-defined functions (see Section 13.7.3.2, "DROP FUNCTION Syntax").

13.1.22 DROP SERVER Syntax

```
DROP SERVER [ IF EXISTS ] server_name
```

Drops the server definition for the server named *server_name*. The corresponding row in the mysql.servers table is deleted. This statement requires the SUPER privilege.

Dropping a server for a table does not affect any FEDERATED tables that used this connection information when they were created. See Section 13.1.13, "CREATE SERVER Syntax".

DROP SERVER does not cause an automatic commit.

In MySQL 5.7, DROP SERVER is not written to the binary log, regardless of the logging format that is in use.

In MySQL 5.7.1, gtid_next must be set to AUTOMATIC before issuing this statement. This restriction does not apply in MySQL 5.7.2 or later. (Bug #16062608, Bug #16715809, Bug #69045)

13.1.23 DROP TABLE Syntax

```
DROP [TEMPORARY] TABLE [IF EXISTS]
    tbl_name [, tbl_name] ...
    [RESTRICT | CASCADE]
```

DROP TABLE removes one or more tables. You must have the DROP privilege for each table. All table data and the table definition are *removed*, so *be careful* with this statement! If any of the tables named in the argument list do not exist, MySQL returns an error indicating by name which nonexisting tables it was unable to drop, but it also drops all of the tables in the list that do exist.

Important

When a table is dropped, user privileges on the table are *not* automatically dropped. See Section 13.7.1.4, "GRANT Syntax".

Note that for a partitioned table, DROP TABLE permanently removes the table definition, all of its partitions, and all of the data which was stored in those partitions. It also removes the partitioning definition (.par) file associated with the dropped table.

Use IF EXISTS to prevent an error from occurring for tables that do not exist. A NOTE is generated for each nonexistent table when using IF EXISTS. See Section 13.7.5.39, "SHOW WARNINGS Syntax".

RESTRICT and CASCADE are permitted to make porting easier. In MySQL 5.7, they do nothing.

Note

DROP TABLE automatically commits the current active transaction, unless you use the TEMPORARY keyword.

The TEMPORARY keyword has the following effects:

- The statement drops only TEMPORARY tables.
- The statement does not end an ongoing transaction.
- No access rights are checked. (A TEMPORARY table is visible only to the session that created it, so no check is necessary.)

Using TEMPORARY is a good way to ensure that you do not accidentally drop a non-TEMPORARY table.

13.1.24 DROP TRIGGER Syntax

```
DROP TRIGGER [IF EXISTS] [schema_name.]trigger_name
```

This statement drops a trigger. The schema (database) name is optional. If the schema is omitted, the trigger is dropped from the default schema. DROP TRIGGER requires the TRIGGER privilege for the table associated with the trigger.

Use IF EXISTS to prevent an error from occurring for a trigger that does not exist. A NOTE is generated for a nonexistent trigger when using IF EXISTS. See Section 13.7.5.39, “SHOW WARNINGS Syntax”.

Triggers for a table are also dropped if you drop the table.

13.1.25 DROP VIEW Syntax

```
DROP VIEW [IF EXISTS]
    view_name [, view_name] ...
    [RESTRICT | CASCADE]
```

DROP VIEW removes one or more views. You must have the DROP privilege for each view. If any of the views named in the argument list do not exist, MySQL returns an error indicating by name which nonexisting views it was unable to drop, but it also drops all of the views in the list that do exist.

The IF EXISTS clause prevents an error from occurring for views that don't exist. When this clause is given, a NOTE is generated for each nonexistent view. See Section 13.7.5.39, “SHOW WARNINGS Syntax”.

RESTRICT and CASCADE, if given, are parsed and ignored.

13.1.26 RENAME TABLE Syntax

```
RENAME TABLE tbl_name TO new_tbl_name
    [, tbl_name2 TO new_tbl_name2] ...
```

This statement renames one or more tables.

The rename operation is done atomically, which means that no other session can access any of the tables while the rename is running. For example, if you have an existing table old_table, you can create another table new_table that has the same structure but is empty, and then replace the existing table with the empty one as follows (assuming that backup_table does not already exist):

```
CREATE TABLE new_table (...);
RENAME TABLE old_table TO backup_table, new_table TO old_table;
```

If the statement renames more than one table, renaming operations are done from left to right. If you want to swap two table names, you can do so like this (assuming that tmp_table does not already exist):

```
RENAME TABLE old_table TO tmp_table,
             new_table TO old_table,
             tmp_table TO new_table;
```

As long as two databases are on the same file system, you can use RENAME TABLE to move a table from one database to another:

```
RENAME TABLE current_db.tbl_name TO other_db.tbl_name;
```

If there are any triggers associated with a table which is moved to a different database using RENAME TABLE, then the statement fails with the error Trigger in wrong schema.

RENAME TABLE also works for views, as long as you do not try to rename a view into a different database.

Any privileges granted specifically for the renamed table or view are not migrated to the new name. They must be changed manually.

When you execute RENAME, you cannot have any locked tables or active transactions. You must also have the ALTER and DROP privileges on the original table, and the CREATE and INSERT privileges on the new table.

If MySQL encounters any errors in a multiple-table rename, it does a reverse rename for all renamed tables to return everything to its original state.

You cannot use RENAME to rename a TEMPORARY table. However, you can use ALTER TABLE instead:

```
mysql> ALTER TABLE orig_name RENAME new_name;
```

13.1.27 TRUNCATE TABLE Syntax

```
TRUNCATE [TABLE] tbl_name
```

TRUNCATE TABLE empties a table completely. It requires the DROP privilege.

Logically, TRUNCATE TABLE is similar to a DELETE statement that deletes all rows, or a sequence of DROP TABLE and CREATE TABLE statements. To achieve high performance, it bypasses the DML method of deleting data. Thus, it cannot be rolled back, it does not cause ON DELETE triggers to fire, and it cannot be performed for InnoDB tables with parent-child foreign key relationships.

Although TRUNCATE TABLE is similar to DELETE, it is classified as a DDL statement rather than a DML statement. It differs from DELETE in the following ways in MySQL 5.7:

- Truncate operations drop and re-create the table, which is much faster than deleting rows one by one, particularly for large tables.
- Truncate operations cause an implicit commit, and so cannot be rolled back.
- Truncation operations cannot be performed if the session holds an active table lock.
- TRUNCATE TABLE fails for an InnoDB table if there are any FOREIGN KEY constraints from other tables that reference the table. Foreign key constraints between columns of the same table are permitted.
- Truncation operations do not return a meaningful value for the number of deleted rows. The usual result is “0 rows affected,” which should be interpreted as “no information.”

- As long as the table format file `tbl_name.frm` is valid, the table can be re-created as an empty table with `TRUNCATE TABLE`, even if the data or index files have become corrupted.

- Any `AUTO_INCREMENT` value is reset to its start value. This is true even for `MyISAM` and `InnoDB`, which normally do not reuse sequence values.

- When used with partitioned tables, `TRUNCATE TABLE` preserves the partitioning; that is, the data and index files are dropped and re-created, while the partition definitions (`.par`) file is unaffected.

- The `TRUNCATE TABLE` statement does not invoke `ON DELETE` triggers.

`TRUNCATE TABLE` for a table closes all handlers for the table that were opened with `HANDLER OPEN`.

`TRUNCATE TABLE` is treated for purposes of binary logging and replication as `DROP TABLE` followed by `CREATE TABLE`—that is, as DDL rather than DML. This is due to the fact that, when using `InnoDB` and other transactional storage engines where the transaction isolation level does not permit statement-based logging (`READ COMMITTED` or `READ UNCOMMITTED`), the statement was not logged and replicated when using `STATEMENT` or `MIXED` logging mode. (Bug #36763) However, it is still applied on replication slaves using `InnoDB` in the manner described previously.

`TRUNCATE TABLE` can be used with Performance Schema summary tables, but the effect is to reset the summary columns to 0 or `NULL`, not to remove rows. See Section 20.9.12, “Performance Schema Summary Tables”.

13.2 Data Manipulation Statements

13.2.1 `CALL` Syntax

```
CALL sp_name([parameter[,...]])
CALL sp_name[()]
```

The `CALL` statement invokes a stored procedure that was defined previously with `CREATE PROCEDURE`.

Stored procedures that take no arguments can be invoked without parentheses. That is, `CALL p()` and `CALL p` are equivalent.

`CALL` can pass back values to its caller using parameters that are declared as `OUT` or `INOUT` parameters. When the procedure returns, a client program can also obtain the number of rows affected for the final statement executed within the routine: At the SQL level, call the `ROW_COUNT()` function; from the C API, call the `mysql_affected_rows()` function.

To get back a value from a procedure using an `OUT` or `INOUT` parameter, pass the parameter by means of a user variable, and then check the value of the variable after the procedure returns. (If you are calling the procedure from within another stored procedure or function, you can also pass a routine parameter or local routine variable as an `IN` or `INOUT` parameter.) For an `INOUT` parameter, initialize its value before passing it to the procedure. The following procedure has an `OUT` parameter that the procedure sets to the current server version, and an `INOUT` value that the procedure increments by one from its current value:

```
CREATE PROCEDURE p (OUT ver_param VARCHAR(25), INOUT incr_param INT)
BEGIN
  # Set value of OUT parameter
  SELECT VERSION() INTO ver_param;
  # Increment value of INOUT parameter
  SET incr_param = incr_param + 1;
END;
```

Before calling the procedure, initialize the variable to be passed as the INOUT parameter. After calling the procedure, the values of the two variables will have been set or modified:

```
mysql> SET @increment = 10;
mysql> CALL p(@version, @increment);
mysql> SELECT @version, @increment;
+--------------+------------+
| @version     | @increment |
+--------------+------------+
| 5.5.3-m3-log |         11 |
+--------------+------------+
```

In prepared CALL statements used with PREPARE and EXECUTE, placeholders can be used for IN parameters. For OUT and INOUT parameters, placeholder support is available as of MySQL 5.5.3. These types of parameters can be used as follows:

```
mysql> SET @increment = 10;
mysql> PREPARE s FROM 'CALL p(?, ?)';
mysql> EXECUTE s USING @version, @increment;
mysql> SELECT @version, @increment;
+--------------+------------+
| @version     | @increment |
+--------------+------------+
| 5.5.3-m3-log |         11 |
+--------------+------------+
```

Before MySQL 5.5.3, placeholder support is not available for OUT or INOUT parameters. To work around this limitation for OUT and INOUT parameters, forego the use of placeholders; instead, refer to user variables in the CALL statement itself and do not specify them in the EXECUTE statement:

```
mysql> SET @increment = 10;
mysql> PREPARE s FROM 'CALL p(@version, @increment)';
mysql> EXECUTE s;
mysql> SELECT @version, @increment;
+--------------+------------+
| @version     | @increment |
+--------------+------------+
| 5.5.0-m2-log |         11 |
+--------------+------------+
```

To write C programs that use the CALL SQL statement to execute stored procedures that produce result sets, the CLIENT_MULTI_RESULTS flag must be enabled. This is because each CALL returns a result to indicate the call status, in addition to any result sets that might be returned by statements executed within the procedure. CLIENT_MULTI_RESULTS must also be enabled if CALL is used to execute any stored procedure that contains prepared statements. It cannot be determined when such a procedure is loaded whether those statements will produce result sets, so it is necessary to assume that they will.

CLIENT_MULTI_RESULTS can be enabled when you call mysql_real_connect(), either explicitly by passing the CLIENT_MULTI_RESULTS flag itself, or implicitly by passing CLIENT_MULTI_STATEMENTS (which also enables CLIENT_MULTI_RESULTS). In MySQL 5.7, CLIENT_MULTI_RESULTS is enabled by default.

To process the result of a CALL statement executed using mysql_query() or mysql_real_query(), use a loop that calls mysql_next_result() to determine whether there are more results. For an example, see Section 21.8.17, "C API Support for Multiple Statement Execution".

For programs written in a language that provides a MySQL interface, there is no native method prior to MySQL 5.5.3 for directly retrieving the results of OUT or INOUT parameters from CALL statements. To get the parameter values, pass user-defined variables to the procedure in the CALL statement and then execute a SELECT statement to produce a result set containing the variable values. To handle an INOUT parameter, execute a statement prior to the CALL that sets the corresponding user variable to the value to be passed to the procedure.

The following example illustrates the technique (without error checking) for the stored procedure p described earlier that has an OUT parameter and an INOUT parameter:

```
mysql_query(mysql, "SET @increment = 10");
mysql_query(mysql, "CALL p(@version, @increment)");
mysql_query(mysql, "SELECT @version, @increment");
result = mysql_store_result(mysql);
row = mysql_fetch_row(result);
mysql_free_result(result);
```

After the preceding code executes, row[0] and row[1] contain the values of @version and @increment, respectively.

In MySQL 5.7, C programs can use the prepared-statement interface to execute CALL statements and access OUT and INOUT parameters. This is done by processing the result of a CALL statement using a loop that calls mysql_stmt_next_result() to determine whether there are more results. For an example, see Section 21.8.20, “C API Support for Prepared CALL Statements”. Languages that provide a MySQL interface can use prepared CALL statements to directly retrieve OUT and INOUT procedure parameters.

In MySQL 5.7, metadata changes to objects referred to by stored programs are detected and cause automatic reparsing of the affected statements when the program is next executed. For more information, see Section 8.9.4, “Caching of Prepared Statements and Stored Programs”.

13.2.2 DELETE Syntax

DELETE is a DML statement that removes rows from a table.

Single-table syntax:

```
DELETE [LOW_PRIORITY] [QUICK] [IGNORE] FROM tbl_name
    [PARTITION (partition_name,...)]
    [WHERE where_condition]
    [ORDER BY ...]
    [LIMIT row_count]
```

The DELETE statement deletes rows from *tbl_name* and returns the number of deleted rows. To check the number of deleted rows, call the ROW_COUNT() function described in Section 12.14, “Information Functions”.

Main Clauses

The conditions in the optional WHERE clause identify which rows to delete. With no WHERE clause, all rows are deleted.

where_condition is an expression that evaluates to true for each row to be deleted. It is specified as described in Section 13.2.9, “SELECT Syntax”.

If the ORDER BY clause is specified, the rows are deleted in the order that is specified. The LIMIT clause places a limit on the number of rows that can be deleted. These clauses apply to single-table deletes, but not multi-table deletes.

Multiple-table syntax:

```
DELETE [LOW_PRIORITY] [QUICK] [IGNORE]
    tbl_name[.*] [, tbl_name[.*]] ...
    FROM table_references
    [WHERE where_condition]
```

Or:

```
DELETE [LOW_PRIORITY] [QUICK] [IGNORE]
    FROM tbl_name[.*] [, tbl_name[.*]] ...
    USING table_references
    [WHERE where_condition]
```

Privileges

You need the DELETE privilege on a table to delete rows from it. You need only the SELECT privilege for any columns that are only read, such as those named in the WHERE clause.

Performance

When you do not need to know the number of deleted rows, the TRUNCATE TABLE statement is a faster way to empty a table than a DELETE statement with no WHERE clause. Unlike DELETE, TRUNCATE TABLE cannot be used within a transaction or if you have a lock on the table. See Section 13.1.27, “TRUNCATE TABLE Syntax” and Section 13.3.5, “LOCK TABLES and UNLOCK TABLES Syntax”.

The speed of delete operations may also be affected by factors discussed in Section 8.2.2.3, “Speed of DELETE Statements”.

To ensure that a given DELETE statement does not take too much time, the MySQL-specific LIMIT *row_count* clause for DELETE specifies the maximum number of rows to be deleted. If the number of rows to delete is larger than the limit, repeat the DELETE statement until the number of affected rows is less than the LIMIT value.

Subqueries

Currently, you cannot delete from a table and select from the same table in a subquery.

Partitioned Tables

DELETE supports explicit partition selection using the PARTITION option, which takes a comma-separated list of the names of one or more partitions or subpartitions (or both) from which to select rows to be dropped. Partitions not included in the list are ignored. Given a partitioned table t with a partition named p0, executing the statement DELETE FROM t PARTITION (p0) has the same effect on the table as executing ALTER TABLE t TRUNCATE PARTITION (p0); in both cases, all rows in partition p0 are dropped.

PARTITION can be used along with a WHERE condition, in which case the condition is tested only on rows in the listed partitions. For example, DELETE FROM t PARTITION (p0) WHERE c < 5 deletes rows only from partition p0 for which the condition c < 5 is true; rows in any other partitions are not checked and thus not affected by the DELETE.

The PARTITION option can also be used in multiple-table DELETE statements. You can use up to one such option per table named in the FROM option.

See Section 17.5, “Partition Selection”, for more information and examples.

Auto-Increment Columns

If you delete the row containing the maximum value for an AUTO_INCREMENT column, the value is not reused for a MyISAM or InnoDB table. If you delete all rows in the table with DELETE FROM *tbl_name* (without a WHERE clause) in autocommit mode, the sequence starts over for all storage engines except InnoDB and MyISAM. There are some exceptions to this behavior for InnoDB tables, as discussed in Section 14.2.6.5, “AUTO_INCREMENT Handling in InnoDB”.

For MyISAM tables, you can specify an AUTO_INCREMENT secondary column in a multiple-column key. In this case, reuse of values deleted from the top of the sequence occurs even for MyISAM tables. See Section 3.6.9, “Using AUTO_INCREMENT”.

Modifiers

The DELETE statement supports the following modifiers:

- If you specify LOW_PRIORITY, the server delays execution of the DELETE until no other clients are reading from the table. This affects only storage engines that use only table-level locking (such as MyISAM, MEMORY, and MERGE).
- For MyISAM tables, if you use the QUICK keyword, the storage engine does not merge index leaves during delete, which may speed up some kinds of delete operations.
- The IGNORE keyword causes MySQL to ignore all errors during the process of deleting rows. (Errors encountered during the parsing stage are processed in the usual manner.) Errors that are ignored due to the use of IGNORE are returned as warnings.

Order of Deletion

If the DELETE statement includes an ORDER BY clause, rows are deleted in the order specified by the clause. This is useful primarily in conjunction with LIMIT. For example, the following statement finds rows matching the WHERE clause, sorts them by timestamp_column, and deletes the first (oldest) one:

```
DELETE FROM somelog WHERE user = 'jcole'
ORDER BY timestamp_column LIMIT 1;
```

ORDER BY also helps to delete rows in an order required to avoid referential integrity violations.

InnoDB Tables

If you are deleting many rows from a large table, you may exceed the lock table size for an InnoDB table. To avoid this problem, or simply to minimize the time that the table remains locked, the following strategy (which does not use DELETE at all) might be helpful:

1. Select the rows *not* to be deleted into an empty table that has the same structure as the original table:

   ```
   INSERT INTO t_copy SELECT * FROM t WHERE ... ;
   ```

2. Use RENAME TABLE to atomically move the original table out of the way and rename the copy to the original name:

   ```
   RENAME TABLE t TO t_old, t_copy TO t;
   ```

3. Drop the original table:

```
DROP TABLE t_old;
```

No other sessions can access the tables involved while RENAME TABLE executes, so the rename operation is not subject to concurrency problems. See Section 13.1.26, “RENAME TABLE Syntax”.

MyISAM Tables

In MyISAM tables, deleted rows are maintained in a linked list and subsequent INSERT operations reuse old row positions. To reclaim unused space and reduce file sizes, use the OPTIMIZE TABLE statement or the myisamchk utility to reorganize tables. OPTIMIZE TABLE is easier to use, but myisamchk is faster. See Section 13.7.2.4, “OPTIMIZE TABLE Syntax”, and Section 4.6.3, “myisamchk — MyISAM Table-Maintenance Utility”.

The QUICK modifier affects whether index leaves are merged for delete operations. DELETE QUICK is most useful for applications where index values for deleted rows are replaced by similar index values from rows inserted later. In this case, the holes left by deleted values are reused.

DELETE QUICK is not useful when deleted values lead to underfilled index blocks spanning a range of index values for which new inserts occur again. In this case, use of QUICK can lead to wasted space in the index that remains unreclaimed. Here is an example of such a scenario:

1. Create a table that contains an indexed AUTO_INCREMENT column.

2. Insert many rows into the table. Each insert results in an index value that is added to the high end of the index.

3. Delete a block of rows at the low end of the column range using DELETE QUICK.

In this scenario, the index blocks associated with the deleted index values become underfilled but are not merged with other index blocks due to the use of QUICK. They remain underfilled when new inserts occur, because new rows do not have index values in the deleted range. Furthermore, they remain underfilled even if you later use DELETE without QUICK, unless some of the deleted index values happen to lie in index blocks within or adjacent to the underfilled blocks. To reclaim unused index space under these circumstances, use OPTIMIZE TABLE.

If you are going to delete many rows from a table, it might be faster to use DELETE QUICK followed by OPTIMIZE TABLE. This rebuilds the index rather than performing many index block merge operations.

Multi-Table Deletes

You can specify multiple tables in a DELETE statement to delete rows from one or more tables depending on the condition in the WHERE clause. You cannot use ORDER BY or LIMIT in a multiple-table DELETE. The *table_references* clause lists the tables involved in the join, as described in Section 13.2.9.2, “JOIN Syntax”.

For the first multiple-table syntax, only matching rows from the tables listed before the FROM clause are deleted. For the second multiple-table syntax, only matching rows from the tables listed in the FROM clause (before the USING clause) are deleted. The effect is that you can delete rows from many tables at the same time and have additional tables that are used only for searching:

```
DELETE t1, t2 FROM t1 INNER JOIN t2 INNER JOIN t3
WHERE t1.id=t2.id AND t2.id=t3.id;
```

Or:

```
DELETE FROM t1, t2 USING t1 INNER JOIN t2 INNER JOIN t3
WHERE t1.id=t2.id AND t2.id=t3.id;
```

These statements use all three tables when searching for rows to delete, but delete matching rows only from tables `t1` and `t2`.

The preceding examples use `INNER JOIN`, but multiple-table `DELETE` statements can use other types of join permitted in `SELECT` statements, such as `LEFT JOIN`. For example, to delete rows that exist in `t1` that have no match in `t2`, use a `LEFT JOIN`:

```
DELETE t1 FROM t1 LEFT JOIN t2 ON t1.id=t2.id WHERE t2.id IS NULL;
```

The syntax permits `.*` after each *`tbl_name`* for compatibility with `Access`.

If you use a multiple-table `DELETE` statement involving `InnoDB` tables for which there are foreign key constraints, the MySQL optimizer might process tables in an order that differs from that of their parent/child relationship. In this case, the statement fails and rolls back. Instead, you should delete from a single table and rely on the `ON DELETE` capabilities that `InnoDB` provides to cause the other tables to be modified accordingly.

> **Note**
>
> If you declare an alias for a table, you must use the alias when referring to the table:
>
> ```
> DELETE t1 FROM test AS t1, test2 WHERE ...
> ```

Table aliases in a multiple-table `DELETE` should be declared only in the *`table_references`* part of the statement. Elsewhere, alias references are permitted but not alias declarations.

Correct:

```
DELETE a1, a2 FROM t1 AS a1 INNER JOIN t2 AS a2
WHERE a1.id=a2.id;

DELETE FROM a1, a2 USING t1 AS a1 INNER JOIN t2 AS a2
WHERE a1.id=a2.id;
```

Incorrect:

```
DELETE t1 AS a1, t2 AS a2 FROM t1 INNER JOIN t2
WHERE a1.id=a2.id;

DELETE FROM t1 AS a1, t2 AS a2 USING t1 INNER JOIN t2
WHERE a1.id=a2.id;
```

13.2.3 DO Syntax

```
DO expr [, expr] ...
```

`DO` executes the expressions but does not return any results. In most respects, `DO` is shorthand for `SELECT` *`expr, ...`*, but has the advantage that it is slightly faster when you do not care about the result.

`DO` is useful primarily with functions that have side effects, such as `RELEASE_LOCK()`.

Example: This SELECT statement pauses, but also produces a result set:

```
mysql> SELECT SLEEP(5);
+----------+
| SLEEP(5) |
+----------+
|        0 |
+----------+
1 row in set (5.02 sec)
```

DO, on the other hand, pauses without producing a result set.:

```
mysql> DO SLEEP(5);
Query OK, 0 rows affected (4.99 sec)
```

This could be useful, for example in a stored function or trigger, which prohibit statements that produce result sets.

DO only executes expressions. It cannot be used in all cases where SELECT can be used. For example, DO id FROM t1 is invalid because it references a table.

13.2.4 HANDLER Syntax

```
HANDLER tbl_name OPEN [ [AS] alias]

HANDLER tbl_name READ index_name { = | <= | >= | < | > } (value1,value2,...)
    [ WHERE where_condition ] [LIMIT ... ]
HANDLER tbl_name READ index_name { FIRST | NEXT | PREV | LAST }
    [ WHERE where_condition ] [LIMIT ... ]
HANDLER tbl_name READ { FIRST | NEXT }
    [ WHERE where_condition ] [LIMIT ... ]

HANDLER tbl_name CLOSE
```

The HANDLER statement provides direct access to table storage engine interfaces. It is available for InnoDB and MyISAM tables.

The HANDLER ... OPEN statement opens a table, making it accessible using subsequent HANDLER ... READ statements. This table object is not shared by other sessions and is not closed until the session calls HANDLER ... CLOSE or the session terminates. If you open the table using an alias, further references to the open table with other HANDLER statements must use the alias rather than the table name.

The first HANDLER ... READ syntax fetches a row where the index specified satisfies the given values and the WHERE condition is met. If you have a multiple-column index, specify the index column values as a comma-separated list. Either specify values for all the columns in the index, or specify values for a leftmost prefix of the index columns. Suppose that an index my_idx includes three columns named col_a, col_b, and col_c, in that order. The HANDLER statement can specify values for all three columns in the index, or for the columns in a leftmost prefix. For example:

```
HANDLER ... READ my_idx = (col_a_val,col_b_val,col_c_val) ...
HANDLER ... READ my_idx = (col_a_val,col_b_val) ...
HANDLER ... READ my_idx = (col_a_val) ...
```

To employ the HANDLER interface to refer to a table's PRIMARY KEY, use the quoted identifier `PRIMARY`:

```
HANDLER tbl_name READ `PRIMARY` ...
```

The second HANDLER ... READ syntax fetches a row from the table in index order that matches the WHERE condition.

The third HANDLER ... READ syntax fetches a row from the table in natural row order that matches the WHERE condition. It is faster than HANDLER *tbl_name* READ *index_name* when a full table scan is desired. Natural row order is the order in which rows are stored in a MyISAM table data file. This statement works for InnoDB tables as well, but there is no such concept because there is no separate data file.

Without a LIMIT clause, all forms of HANDLER ... READ fetch a single row if one is available. To return a specific number of rows, include a LIMIT clause. It has the same syntax as for the SELECT statement. See Section 13.2.9, “SELECT Syntax”.

HANDLER ... CLOSE closes a table that was opened with HANDLER ... OPEN.

There are several reasons to use the HANDLER interface instead of normal SELECT statements:

- HANDLER is faster than SELECT:
  - A designated storage engine handler object is allocated for the HANDLER ... OPEN. The object is reused for subsequent HANDLER statements for that table; it need not be reinitialized for each one.
  - There is less parsing involved.
  - There is no optimizer or query-checking overhead.
  - The handler interface does not have to provide a consistent look of the data (for example, dirty reads are permitted), so the storage engine can use optimizations that SELECT does not normally permit.
- HANDLER makes it easier to port to MySQL applications that use a low-level ISAM-like interface. (See Section 14.2.16, “InnoDB Integration with memcached” for an alternative way to adapt applications that use the key-value store paradigm.)
- HANDLER enables you to traverse a database in a manner that is difficult (or even impossible) to accomplish with SELECT. The HANDLER interface is a more natural way to look at data when working with applications that provide an interactive user interface to the database.

HANDLER is a somewhat low-level statement. For example, it does not provide consistency. That is, HANDLER ... OPEN does *not* take a snapshot of the table, and does *not* lock the table. This means that after a HANDLER ... OPEN statement is issued, table data can be modified (by the current session or other sessions) and these modifications might be only partially visible to HANDLER ... NEXT or HANDLER ... PREV scans.

An open handler can be closed and marked for reopen, in which case the handler loses its position in the table. This occurs when both of the following circumstances are true:

- Any session executes FLUSH TABLES or DDL statements on the handler's table.
- The session in which the handler is open executes non-HANDLER statements that use tables.

TRUNCATE TABLE for a table closes all handlers for the table that were opened with HANDLER OPEN.

If a table is flushed with FLUSH TABLES *tbl_name* WITH READ LOCK was opened with HANDLER, the handler is implicitly flushed and loses its position.

In previous versions of MySQL, HANDLER was not supported with partitioned tables. This limitation is removed beginning with MySQL 5.7.1.

13.2.5 INSERT Syntax

```
INSERT [LOW_PRIORITY | DELAYED | HIGH_PRIORITY] [IGNORE]
    [INTO] tbl_name
    [PARTITION (partition_name,...)]
    [(col_name,...)]
    {VALUES | VALUE} ({expr | DEFAULT},...),(...),...
    [ ON DUPLICATE KEY UPDATE
      col_name=expr
        [, col_name=expr] ... ]
```

Or:

```
INSERT [LOW_PRIORITY | DELAYED | HIGH_PRIORITY] [IGNORE]
    [INTO] tbl_name
    [PARTITION (partition_name,...)]
    SET col_name={expr | DEFAULT}, ...
    [ ON DUPLICATE KEY UPDATE
      col_name=expr
        [, col_name=expr] ... ]
```

Or:

```
INSERT [LOW_PRIORITY | HIGH_PRIORITY] [IGNORE]
    [INTO] tbl_name
    [PARTITION (partition_name,...)]
    [(col_name,...)]
    SELECT ...
    [ ON DUPLICATE KEY UPDATE
      col_name=expr
        [, col_name=expr] ... ]
```

`INSERT` inserts new rows into an existing table. The `INSERT ... VALUES` and `INSERT ... SET` forms of the statement insert rows based on explicitly specified values. The `INSERT ... SELECT` form inserts rows selected from another table or tables. `INSERT ... SELECT` is discussed further in Section 13.2.5.1, “`INSERT ... SELECT` Syntax”.

When inserting into a partitioned table, you can control which partitions and subpartitions accept new rows. The `PARTITION` option takes a comma-separated list of the names of one or more partitions or subpartitions (or both) of the table. If any of the rows to be inserted by a given `INSERT` statement do not match one of the partitions listed, the `INSERT` statement fails with the error `Found a row not matching the given partition set`. See Section 17.5, “Partition Selection”, for more information and examples.

You can use `REPLACE` instead of `INSERT` to overwrite old rows. `REPLACE` is the counterpart to `INSERT IGNORE` in the treatment of new rows that contain unique key values that duplicate old rows: The new rows are used to replace the old rows rather than being discarded. See Section 13.2.8, “REPLACE Syntax”.

`tbl_name` is the table into which rows should be inserted. The columns for which the statement provides values can be specified as follows:

- You can provide a comma-separated list of column names following the table name. In this case, a value for each named column must be provided by the `VALUES` list or the `SELECT` statement.

- If you do not specify a list of column names for `INSERT ... VALUES` or `INSERT ... SELECT`, values for every column in the table must be provided by the `VALUES` list or the `SELECT` statement. If you do not know the order of the columns in the table, use `DESCRIBE tbl_name` to find out.

- The SET clause indicates the column names explicitly.

Column values can be given in several ways:

- If you are not running in strict SQL mode, any column not explicitly given a value is set to its default (explicit or implicit) value. For example, if you specify a column list that does not name all the columns in the table, unnamed columns are set to their default values. Default value assignment is described in Section 11.5, “Data Type Default Values”. See also Section 1.8.3.3, “Constraints on Invalid Data”.

  If you want an INSERT statement to generate an error unless you explicitly specify values for all columns that do not have a default value, you should use strict mode. See Section 5.1.7, “Server SQL Modes”.

- Use the keyword DEFAULT to set a column explicitly to its default value. This makes it easier to write INSERT statements that assign values to all but a few columns, because it enables you to avoid writing an incomplete VALUES list that does not include a value for each column in the table. Otherwise, you would have to write out the list of column names corresponding to each value in the VALUES list.

  You can also use DEFAULT(*col_name*) as a more general form that can be used in expressions to produce a given column's default value.

- If both the column list and the VALUES list are empty, INSERT creates a row with each column set to its default value:

  ```
  INSERT INTO tbl_name () VALUES();
  ```

  In strict mode, an error occurs if any column doesn't have a default value. Otherwise, MySQL uses the implicit default value for any column that does not have an explicitly defined default.

- You can specify an expression *expr* to provide a column value. This might involve type conversion if the type of the expression does not match the type of the column, and conversion of a given value can result in different inserted values depending on the data type. For example, inserting the string '1999.0e-2' into an INT, FLOAT, DECIMAL(10,6), or YEAR column results in the values 1999, 19.9921, 19.992100, and 1999 being inserted, respectively. The reason the value stored in the INT and YEAR columns is 1999 is that the string-to-integer conversion looks only at as much of the initial part of the string as may be considered a valid integer or year. For the floating-point and fixed-point columns, the string-to-floating-point conversion considers the entire string a valid floating-point value.

  An expression *expr* can refer to any column that was set earlier in a value list. For example, you can do this because the value for col2 refers to col1, which has previously been assigned:

  ```
  INSERT INTO tbl_name (col1,col2) VALUES(15,col1*2);
  ```

  But the following is not legal, because the value for col1 refers to col2, which is assigned after col1:

  ```
  INSERT INTO tbl_name (col1,col2) VALUES(col2*2,15);
  ```

  One exception involves columns that contain AUTO_INCREMENT values. Because the AUTO_INCREMENT value is generated after other value assignments, any reference to an AUTO_INCREMENT column in the assignment returns a 0.

INSERT statements that use VALUES syntax can insert multiple rows. To do this, include multiple lists of column values, each enclosed within parentheses and separated by commas. Example:

```
INSERT INTO tbl_name (a,b,c) VALUES(1,2,3),(4,5,6),(7,8,9);
```

The values list for each row must be enclosed within parentheses. The following statement is illegal because the number of values in the list does not match the number of column names:

```
INSERT INTO tbl_name (a,b,c) VALUES(1,2,3,4,5,6,7,8,9);
```

VALUE is a synonym for VALUES in this context. Neither implies anything about the number of values lists, and either may be used whether there is a single values list or multiple lists.

The affected-rows value for an INSERT can be obtained using the ROW_COUNT() function (see Section 12.14, “Information Functions”), or the mysql_affected_rows() C API function (see Section 21.8.7.1, “mysql_affected_rows()”).

If you use an INSERT ... VALUES statement with multiple value lists or INSERT ... SELECT, the statement returns an information string in this format:

```
Records: 100 Duplicates: 0 Warnings: 0
```

Records indicates the number of rows processed by the statement. (This is not necessarily the number of rows actually inserted because Duplicates can be nonzero.) Duplicates indicates the number of rows that could not be inserted because they would duplicate some existing unique index value. Warnings indicates the number of attempts to insert column values that were problematic in some way. Warnings can occur under any of the following conditions:

- Inserting NULL into a column that has been declared NOT NULL. For multiple-row INSERT statements or INSERT INTO ... SELECT statements, the column is set to the implicit default value for the column data type. This is 0 for numeric types, the empty string ('') for string types, and the “zero” value for date and time types. INSERT INTO ... SELECT statements are handled the same way as multiple-row inserts because the server does not examine the result set from the SELECT to see whether it returns a single row. (For a single-row INSERT, no warning occurs when NULL is inserted into a NOT NULL column. Instead, the statement fails with an error.)

- Setting a numeric column to a value that lies outside the column's range. The value is clipped to the closest endpoint of the range.

- Assigning a value such as '10.34 a' to a numeric column. The trailing nonnumeric text is stripped off and the remaining numeric part is inserted. If the string value has no leading numeric part, the column is set to 0.

- Inserting a string into a string column (CHAR, VARCHAR, TEXT, or BLOB) that exceeds the column's maximum length. The value is truncated to the column's maximum length.

- Inserting a value into a date or time column that is illegal for the data type. The column is set to the appropriate zero value for the type.

If you are using the C API, the information string can be obtained by invoking the mysql_info() function. See Section 21.8.7.36, “mysql_info()”.

If INSERT inserts a row into a table that has an AUTO_INCREMENT column, you can find the value used for that column by using the SQL LAST_INSERT_ID() function. From within the C API, use the mysql_insert_id() function. However, you should note that the two functions do not always behave identically. The behavior of INSERT statements with respect to AUTO_INCREMENT columns is discussed further in Section 12.14, “Information Functions”, and Section 21.8.7.38, “mysql_insert_id()”.

The INSERT statement supports the following modifiers:

- INSERT DELAYED was deprecated in MySQL 5.6, and is scheduled for eventual removal. Use INSERT (without DELAYED) instead. See Section 13.2.5.2, “INSERT DELAYED Syntax”.

- If you use the LOW_PRIORITY keyword, execution of the INSERT is delayed until no other clients are reading from the table. This includes other clients that began reading while existing clients are reading, and while the INSERT LOW_PRIORITY statement is waiting. It is possible, therefore, for a client that issues an INSERT LOW_PRIORITY statement to wait for a very long time Note that LOW_PRIORITY should normally not be used with MyISAM tables because doing so disables concurrent inserts. See Section 8.10.3, “Concurrent Inserts”.

  If you specify HIGH_PRIORITY, it overrides the effect of the --low-priority-updates option if the server was started with that option. It also causes concurrent inserts not to be used. See Section 8.10.3, “Concurrent Inserts”.

  LOW_PRIORITY and HIGH_PRIORITY affect only storage engines that use only table-level locking (such as MyISAM, MEMORY, and MERGE).

- If you use the IGNORE keyword, errors that occur while executing the INSERT statement are ignored. For example, without IGNORE, a row that duplicates an existing UNIQUE index or PRIMARY KEY value in the table causes a duplicate-key error and the statement is aborted. With IGNORE, the row still is not inserted, but no error occurs. Ignored errors may generate warnings instead, although duplicate-key errors do not.

  IGNORE has a similar effect on inserts into partitioned tables where no partition matching a given value is found. Without IGNORE, such INSERT statements are aborted with an error; however, when INSERT IGNORE is used, the insert operation fails silently for the row containing the unmatched value, but any rows that are matched are inserted. For an example, see Section 17.2.2, “LIST Partitioning”.

  Data conversions that would trigger errors abort the statement if IGNORE is not specified. With IGNORE, invalid values are adjusted to the closest values and inserted; warnings are produced but the statement does not abort. You can determine with the mysql_info() C API function how many rows were actually inserted into the table.

- If you specify ON DUPLICATE KEY UPDATE, and a row is inserted that would cause a duplicate value in a UNIQUE index or PRIMARY KEY, an UPDATE of the old row is performed. The affected-rows value per row is 1 if the row is inserted as a new row, 2 if an existing row is updated, and 0 if an existing row is set to its current values. If you specify the CLIENT_FOUND_ROWS flag to mysql_real_connect() when connecting to mysqld, the affected-rows value is 1 (not 0) if an existing row is set to its current values. See Section 13.2.5.3, “INSERT ... ON DUPLICATE KEY UPDATE Syntax”.

Inserting into a table requires the INSERT privilege for the table. If the ON DUPLICATE KEY UPDATE clause is used and a duplicate key causes an UPDATE to be performed instead, the statement requires the UPDATE privilege for the columns to be updated. For columns that are read but not modified you need only the SELECT privilege (such as for a column referenced only on the right hand side of an *col_name*=*expr* assignment in an ON DUPLICATE KEY UPDATE clause).

In MySQL 5.7, an INSERT statement affecting a partitioned table using a storage engine such as MyISAM that employs table-level locks locks only those partitions into which rows are actually inserted. (For storage engines such as InnoDB that employ row-level locking, no locking of partitions takes place.) For more information, see Section 17.6.4, “Partitioning and Locking”.

13.2.5.1 INSERT ... SELECT Syntax

```
INSERT [LOW_PRIORITY | HIGH_PRIORITY] [IGNORE]
    [INTO] tbl_name
    [PARTITION (partition_name,...)]
    [(col_name,...)]
    SELECT ...
    [ ON DUPLICATE KEY UPDATE col_name=expr, ... ]
```

With INSERT ... SELECT, you can quickly insert many rows into a table from one or many tables. For example:

```
INSERT INTO tbl_temp2 (fld_id)
  SELECT tbl_temp1.fld_order_id
  FROM tbl_temp1 WHERE tbl_temp1.fld_order_id > 100;
```

The following conditions hold for a INSERT ... SELECT statements:

- Specify IGNORE to ignore rows that would cause duplicate-key violations.
- The target table of the INSERT statement may appear in the FROM clause of the SELECT part of the query. (This was not possible in some older versions of MySQL.) However, you cannot insert into a table and select from the same table in a subquery.

  When selecting from and inserting into a table at the same time, MySQL creates a temporary table to hold the rows from the SELECT and then inserts those rows into the target table. However, it remains true that you cannot use INSERT INTO t ... SELECT ... FROM t when t is a TEMPORARY table, because TEMPORARY tables cannot be referred to twice in the same statement (see Section C.5.7.2, “TEMPORARY Table Problems”).
- AUTO_INCREMENT columns work as usual.
- To ensure that the binary log can be used to re-create the original tables, MySQL does not permit concurrent inserts for INSERT ... SELECT statements.
- To avoid ambiguous column reference problems when the SELECT and the INSERT refer to the same table, provide a unique alias for each table used in the SELECT part, and qualify column names in that part with the appropriate alias.

You can explicitly select which partitions or subpartitions (or both) of the source or target table (or both) are to be used with a PARTITION option following the name of the table. When PARTITION is used with the name of the source table in the SELECT portion of the statement, rows are selected only from the partitions or subpartitions named in its partition list. When PARTITION is used with the name of the target table for the INSERT portion of the statement, then it must be possible to insert all rows selected into the partitions or subpartitions named in the partition list following the option, else the INSERT ... SELECT statement fails. For more information and examples, see Section 17.5, “Partition Selection”.

In the values part of ON DUPLICATE KEY UPDATE, you can refer to columns in other tables, as long as you do not use GROUP BY in the SELECT part. One side effect is that you must qualify nonunique column names in the values part.

The order in which rows are returned by a SELECT statement with no ORDER BY clause is not determined. This means that, when using replication, there is no guarantee that such a SELECT returns rows in the same order on the master and the slave; this can lead to inconsistencies between them. To prevent this from occurring, you should always write INSERT ... SELECT statements that are to be replicated as INSERT ... SELECT ... ORDER BY *column*. The choice of *column* does not matter as long as the same order for returning the rows is enforced on both the master and the slave. See also Section 16.4.1.16, “Replication and LIMIT”.

Due to this issue, INSERT ... SELECT ON DUPLICATE KEY UPDATE and INSERT IGNORE ... SELECT statements are flagged as unsafe for statement-based replication. With this change, such statements produce a warning in the log when using statement-based mode and are logged using the row-based format when using MIXED mode. (Bug #11758262, Bug #50439)

See also Section 16.1.2.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”.

In MySQL 5.7, an INSERT ... SELECT statement that acted on partitioned tables using a storage engine such as MyISAM that employs table-level locks locks all partitions of the target table; however, only those partitions that are actually read from the source table are locked. (This does not occur with tables using storage engines such as InnoDB that employ row-level locking.) See Section 17.6.4, “Partitioning and Locking”, for more information.

13.2.5.2 INSERT DELAYED Syntax

```
INSERT DELAYED ...
```

The DELAYED option for the INSERT statement is a MySQL extension to standard SQL. In previous versions of MySQL, it can be used for certain kinds of tables (such as MyISAM), such that when a client uses INSERT DELAYED, it gets an okay from the server at once, and the row is queued to be inserted when the table is not in use by any other thread.

DELAYED inserts and replaces were deprecated in MySQL 5.6.6. In MySQL 5.7, DELAYED is not supported. The server recognizes but ignores the DELAYED keyword, handles the insert as a nondelayed insert, and generates an ER_WARN_LEGACY_SYNTAX_CONVERTED warning. (“INSERT DELAYED is no longer supported. The statement was converted to INSERT.”) The DELAYED keyword will be removed in a future release.

13.2.5.3 INSERT ... ON DUPLICATE KEY UPDATE Syntax

If you specify ON DUPLICATE KEY UPDATE, and a row is inserted that would cause a duplicate value in a UNIQUE index or PRIMARY KEY, MySQL performs an UPDATE of the old row. For example, if column a is declared as UNIQUE and contains the value 1, the following two statements have similar effect:

```
INSERT INTO table (a,b,c) VALUES (1,2,3)
  ON DUPLICATE KEY UPDATE c=c+1;

UPDATE table SET c=c+1 WHERE a=1;
```

(The effects are not identical for an InnoDB table where a is an auto-increment column. With an auto-increment column, an INSERT statement increases the auto-increment value but UPDATE does not.)

The ON DUPLICATE KEY UPDATE clause can contain multiple column assignments, separated by commas.

With ON DUPLICATE KEY UPDATE, the affected-rows value per row is 1 if the row is inserted as a new row, 2 if an existing row is updated, and 0 if an existing row is set to its current values. If you specify the CLIENT_FOUND_ROWS flag to mysql_real_connect() when connecting to mysqld, the affected-rows value is 1 (not 0) if an existing row is set to its current values.

If column b is also unique, the INSERT is equivalent to this UPDATE statement instead:

```
UPDATE table SET c=c+1 WHERE a=1 OR b=2 LIMIT 1;
```

If a=1 OR b=2 matches several rows, only *one* row is updated. In general, you should try to avoid using an ON DUPLICATE KEY UPDATE clause on tables with multiple unique indexes.

You can use the VALUES(*col_name*) function in the UPDATE clause to refer to column values from the INSERT portion of the INSERT ... ON DUPLICATE KEY UPDATE statement. In other words, VALUES(*col_name*) in the ON DUPLICATE KEY UPDATE clause refers to the value of *col_name* that would be inserted, had no duplicate-key conflict occurred. This function is especially useful in multiple-row inserts. The VALUES() function is meaningful only in INSERT ... UPDATE statements and returns NULL otherwise. Example:

```
INSERT INTO table (a,b,c) VALUES (1,2,3),(4,5,6)
  ON DUPLICATE KEY UPDATE c=VALUES(a)+VALUES(b);
```

That statement is identical to the following two statements:

```
INSERT INTO table (a,b,c) VALUES (1,2,3)
  ON DUPLICATE KEY UPDATE c=3;
INSERT INTO table (a,b,c) VALUES (4,5,6)
  ON DUPLICATE KEY UPDATE c=9;
```

If a table contains an AUTO_INCREMENT column and INSERT ... ON DUPLICATE KEY UPDATE inserts or updates a row, the LAST_INSERT_ID() function returns the AUTO_INCREMENT value.

The DELAYED option is ignored when you use ON DUPLICATE KEY UPDATE.

Because the results of INSERT ... SELECT statements depend on the ordering of rows from the SELECT and this order cannot always be guaranteed, it is possible when logging INSERT ... SELECT ON DUPLICATE KEY UPDATE statements for the master and the slave to diverge. Thus, INSERT ... SELECT ON DUPLICATE KEY UPDATE statements are flagged as unsafe for statement-based replication. With this change, such statements produce a warning in the log when using statement-based mode and are logged using the row-based format when using MIXED mode. In addition, an INSERT ... ON DUPLICATE KEY UPDATE statement against a table having more than one unique or primary key is also marked as unsafe. (Bug #11765650, Bug #58637) See also Section 16.1.2.1, "Advantages and Disadvantages of Statement-Based and Row-Based Replication".

In MySQL 5.7, an INSERT ... ON DUPLICATE KEY UPDATE on a partitioned table using a storage engine such as MyISAM that employs table-level locks locks any partitions of the table in which a partitioning key column is updated. (This does not occur with tables using storage engines such as InnoDB that employ row-level locking.) See Section 17.6.4, "Partitioning and Locking", for more information.

13.2.6 LOAD DATA INFILE Syntax

```
LOAD DATA [LOW_PRIORITY | CONCURRENT] [LOCAL] INFILE 'file_name'
    [REPLACE | IGNORE]
    INTO TABLE tbl_name
    [PARTITION (partition_name,...)]
    [CHARACTER SET charset_name]
    [{FIELDS | COLUMNS}
        [TERMINATED BY 'string']
        [[OPTIONALLY] ENCLOSED BY 'char']
        [ESCAPED BY 'char']
    ]
    [LINES
        [STARTING BY 'string']
        [TERMINATED BY 'string']
    ]
    [IGNORE number {LINES | ROWS}]
    [(col_name_or_user_var,...)]
    [SET col_name = expr,...]
```

The LOAD DATA INFILE statement reads rows from a text file into a table at a very high speed. LOAD DATA INFILE is the complement of SELECT ... INTO OUTFILE. (See Section 13.2.9.1, "SELECT ... INTO Syntax".) To write data from a table to a file, use SELECT ... INTO OUTFILE. To read the file back into a table, use LOAD DATA INFILE. The syntax of the FIELDS and LINES clauses is the same for both statements. Both clauses are optional, but FIELDS must precede LINES if both are specified.

You can also load data files by using the mysqlimport utility; it operates by sending a LOAD DATA INFILE statement to the server. The --local option causes mysqlimport to read data files from the client host. You can specify the --compress option to get better performance over slow networks if the client and server support the compressed protocol. See Section 4.5.5, “mysqlimport — A Data Import Program”.

For more information about the efficiency of INSERT versus LOAD DATA INFILE and speeding up LOAD DATA INFILE, see Section 8.2.2.1, “Speed of INSERT Statements”.

The file name must be given as a literal string. On Windows, specify backslashes in path names as forward slashes or doubled backslashes. The character_set_filesystem system variable controls the interpretation of the file name.

LOAD DATA supports explicit partition selection using the PARTITION option with a comma-separated list of more or more names of partitions, subpartitions, or both. When this option is used, if any rows from the file cannot be inserted into any of the partitions or subpartitions named in the list, the statement fails with the error Found a row not matching the given partition set. For more information, see Section 17.5, “Partition Selection”.

For partitioned tables using storage engines that employ table locks, such as MyISAM, LOAD DATA cannot prune any partition locks. This does not apply to tables using storage engines which employ row-level locking, such as InnoDB. For more information, see Section 17.6.4, “Partitioning and Locking”.

The character set indicated by the character_set_database system variable is used to interpret the information in the file. SET NAMES and the setting of character_set_client do not affect interpretation of input. If the contents of the input file use a character set that differs from the default, it is usually preferable to specify the character set of the file by using the CHARACTER SET clause. A character set of binary specifies “no conversion.”

LOAD DATA INFILE interprets all fields in the file as having the same character set, regardless of the data types of the columns into which field values are loaded. For proper interpretation of file contents, you must ensure that it was written with the correct character set. For example, if you write a data file with mysqldump -T or by issuing a SELECT ... INTO OUTFILE statement in mysql, be sure to use a --default-character-set option so that output is written in the character set to be used when the file is loaded with LOAD DATA INFILE.

> **Note**
>
> It is not possible to load data files that use the ucs2, utf16, utf16le, or utf32 character set.

If you use LOW_PRIORITY, execution of the LOAD DATA statement is delayed until no other clients are reading from the table. This affects only storage engines that use only table-level locking (such as MyISAM, MEMORY, and MERGE).

If you specify CONCURRENT with a MyISAM table that satisfies the condition for concurrent inserts (that is, it contains no free blocks in the middle), other threads can retrieve data from the table while LOAD DATA is executing. This option affects the performance of LOAD DATA a bit, even if no other thread is using the table at the same time.

With row-based replication, CONCURRENT is replicated regardless of MySQL version. With statement-based replication CONCURRENT is not replicated prior to MySQL 5.5.1 (see Bug #34628). For more information, see Section 16.4.1.17, “Replication and LOAD DATA INFILE”.

The LOCAL keyword affects expected location of the file and error handling, as described later. LOCAL works only if your server and your client both have been configured to permit it. For example, if mysqld was started with --local-infile=0, LOCAL does not work. See Section 6.1.6, “Security Issues with LOAD DATA LOCAL”.

The LOCAL keyword affects where the file is expected to be found:

- If LOCAL is specified, the file is read by the client program on the client host and sent to the server. The file can be given as a full path name to specify its exact location. If given as a relative path name, the name is interpreted relative to the directory in which the client program was started.

  When using LOCAL with LOAD DATA, a copy of the file is created in the server's temporary directory. This is *not* the directory determined by the value of tmpdir or slave_load_tmpdir, but rather the operating system's temporary directory, and is not configurable in the MySQL Server. (Typically the system temporary directory is /tmp on Linux systems and C:\WINDOWS\TEMP on Windows.) Lack of sufficient space for the copy in this directory can cause the LOAD DATA LOCAL statement to fail.

- If LOCAL is not specified, the file must be located on the server host and is read directly by the server. The server uses the following rules to locate the file:

  - If the file name is an absolute path name, the server uses it as given.

  - If the file name is a relative path name with one or more leading components, the server searches for the file relative to the server's data directory.

  - If a file name with no leading components is given, the server looks for the file in the database directory of the default database.

In the non-LOCAL case, these rules mean that a file named as ./myfile.txt is read from the server's data directory, whereas the file named as myfile.txt is read from the database directory of the default database. For example, if db1 is the default database, the following LOAD DATA statement reads the file data.txt from the database directory for db1, even though the statement explicitly loads the file into a table in the db2 database:

```
LOAD DATA INFILE 'data.txt' INTO TABLE db2.my_table;
```

For security reasons, when reading text files located on the server, the files must either reside in the database directory or be readable by all. Also, to use LOAD DATA INFILE on server files, you must have the FILE privilege. See Section 6.2.1, “Privileges Provided by MySQL”. For non-LOCAL load operations, if the secure_file_priv system variable is set to a nonempty directory name, the file to be loaded must be located in that directory.

Using LOCAL is a bit slower than letting the server access the files directly, because the contents of the file must be sent over the connection by the client to the server. On the other hand, you do not need the FILE privilege to load local files.

LOCAL also affects error handling:

- With LOAD DATA INFILE, data-interpretation and duplicate-key errors terminate the operation.

- With LOAD DATA LOCAL INFILE, data-interpretation and duplicate-key errors become warnings and the operation continues because the server has no way to stop transmission of the file in the middle of the operation. For duplicate-key errors, this is the same as if IGNORE is specified. IGNORE is explained further later in this section.

The REPLACE and IGNORE keywords control handling of input rows that duplicate existing rows on unique key values:

- If you specify REPLACE, input rows replace existing rows. In other words, rows that have the same value for a primary key or unique index as an existing row. See Section 13.2.8, “REPLACE Syntax”.
- If you specify IGNORE, input rows that duplicate an existing row on a unique key value are skipped.
- If you do not specify either option, the behavior depends on whether the LOCAL keyword is specified. Without LOCAL, an error occurs when a duplicate key value is found, and the rest of the text file is ignored. With LOCAL, the default behavior is the same as if IGNORE is specified; this is because the server has no way to stop transmission of the file in the middle of the operation.

To ignore foreign key constraints during the load operation, issue a SET foreign_key_checks = 0 statement before executing LOAD DATA.

If you use LOAD DATA INFILE on an empty MyISAM table, all nonunique indexes are created in a separate batch (as for REPAIR TABLE). Normally, this makes LOAD DATA INFILE much faster when you have many indexes. In some extreme cases, you can create the indexes even faster by turning them off with ALTER TABLE ... DISABLE KEYS before loading the file into the table and using ALTER TABLE ... ENABLE KEYS to re-create the indexes after loading the file. See Section 8.2.2.1, “Speed of INSERT Statements”.

For both the LOAD DATA INFILE and SELECT ... INTO OUTFILE statements, the syntax of the FIELDS and LINES clauses is the same. Both clauses are optional, but FIELDS must precede LINES if both are specified.

If you specify a FIELDS clause, each of its subclauses (TERMINATED BY, [OPTIONALLY] ENCLOSED BY, and ESCAPED BY) is also optional, except that you must specify at least one of them.

If you specify no FIELDS or LINES clause, the defaults are the same as if you had written this:

```
FIELDS TERMINATED BY '\t' ENCLOSED BY '' ESCAPED BY '\\'
LINES TERMINATED BY '\n' STARTING BY ''
```

(Backslash is the MySQL escape character within strings in SQL statements, so to specify a literal backslash, you must specify two backslashes for the value to be interpreted as a single backslash. The escape sequences '\t' and '\n' specify tab and newline characters, respectively.)

In other words, the defaults cause LOAD DATA INFILE to act as follows when reading input:

- Look for line boundaries at newlines.
- Do not skip over any line prefix.
- Break lines into fields at tabs.
- Do not expect fields to be enclosed within any quoting characters.
- Interpret characters preceded by the escape character “\” as escape sequences. For example, “\t”, “\n”, and “\\” signify tab, newline, and backslash, respectively. See the discussion of FIELDS ESCAPED BY later for the full list of escape sequences.

Conversely, the defaults cause SELECT ... INTO OUTFILE to act as follows when writing output:

- Write tabs between fields.
- Do not enclose fields within any quoting characters.
- Use “\” to escape instances of tab, newline, or “\” that occur within field values.

- Write newlines at the ends of lines.

> **Note**
>
> If you have generated the text file on a Windows system, you might have to use `LINES TERMINATED BY '\r\n'` to read the file properly, because Windows programs typically use two characters as a line terminator. Some programs, such as `WordPad`, might use `\r` as a line terminator when writing files. To read such files, use `LINES TERMINATED BY '\r'`.

If all the lines you want to read in have a common prefix that you want to ignore, you can use `LINES STARTING BY 'prefix_string'` to skip over the prefix, *and anything before it*. If a line does not include the prefix, the entire line is skipped. Suppose that you issue the following statement:

```
LOAD DATA INFILE '/tmp/test.txt' INTO TABLE test
  FIELDS TERMINATED BY ','  LINES STARTING BY 'xxx';
```

If the data file looks like this:

```
xxx"abc",1
something xxx"def",2
"ghi",3
```

The resulting rows will be `("abc",1)` and `("def",2)`. The third row in the file is skipped because it does not contain the prefix.

The `IGNORE number LINES` option can be used to ignore lines at the start of the file. For example, you can use `IGNORE 1 LINES` to skip over an initial header line containing column names:

```
LOAD DATA INFILE '/tmp/test.txt' INTO TABLE test IGNORE 1 LINES;
```

When you use `SELECT ... INTO OUTFILE` in tandem with `LOAD DATA INFILE` to write data from a database into a file and then read the file back into the database later, the field- and line-handling options for both statements must match. Otherwise, `LOAD DATA INFILE` will not interpret the contents of the file properly. Suppose that you use `SELECT ... INTO OUTFILE` to write a file with fields delimited by commas:

```
SELECT * INTO OUTFILE 'data.txt'
  FIELDS TERMINATED BY ','
  FROM table2;
```

To read the comma-delimited file back in, the correct statement would be:

```
LOAD DATA INFILE 'data.txt' INTO TABLE table2
  FIELDS TERMINATED BY ',';
```

If instead you tried to read in the file with the statement shown following, it wouldn't work because it instructs `LOAD DATA INFILE` to look for tabs between fields:

```
LOAD DATA INFILE 'data.txt' INTO TABLE table2
  FIELDS TERMINATED BY '\t';
```

The likely result is that each input line would be interpreted as a single field.

LOAD DATA INFILE can be used to read files obtained from external sources. For example, many programs can export data in comma-separated values (CSV) format, such that lines have fields separated by commas and enclosed within double quotation marks, with an initial line of column names. If the lines in such a file are terminated by carriage return/newline pairs, the statement shown here illustrates the field- and line-handling options you would use to load the file:

```
LOAD DATA INFILE 'data.txt' INTO TABLE tbl_name
  FIELDS TERMINATED BY ',' ENCLOSED BY '"'
  LINES TERMINATED BY '\r\n'
  IGNORE 1 LINES;
```

If the input values are not necessarily enclosed within quotation marks, use OPTIONALLY before the ENCLOSED BY keywords.

Any of the field- or line-handling options can specify an empty string (''). If not empty, the FIELDS [OPTIONALLY] ENCLOSED BY and FIELDS ESCAPED BY values must be a single character. The FIELDS TERMINATED BY, LINES STARTING BY, and LINES TERMINATED BY values can be more than one character. For example, to write lines that are terminated by carriage return/linefeed pairs, or to read a file containing such lines, specify a LINES TERMINATED BY '\r\n' clause.

To read a file containing jokes that are separated by lines consisting of %%, you can do this

```
CREATE TABLE jokes
  (a INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
  joke TEXT NOT NULL);
LOAD DATA INFILE '/tmp/jokes.txt' INTO TABLE jokes
  FIELDS TERMINATED BY ''
  LINES TERMINATED BY '\n%%\n' (joke);
```

FIELDS [OPTIONALLY] ENCLOSED BY controls quoting of fields. For output (SELECT ... INTO OUTFILE), if you omit the word OPTIONALLY, all fields are enclosed by the ENCLOSED BY character. An example of such output (using a comma as the field delimiter) is shown here:

```
"1","a string","100.20"
"2","a string containing a , comma","102.20"
"3","a string containing a \" quote","102.20"
"4","a string containing a \", quote and comma","102.20"
```

If you specify OPTIONALLY, the ENCLOSED BY character is used only to enclose values from columns that have a string data type (such as CHAR, BINARY, TEXT, or ENUM):

```
1,"a string",100.20
2,"a string containing a , comma",102.20
3,"a string containing a \" quote",102.20
4,"a string containing a \", quote and comma",102.20
```

Note that occurrences of the ENCLOSED BY character within a field value are escaped by prefixing them with the ESCAPED BY character. Also note that if you specify an empty ESCAPED BY value, it is possible to inadvertently generate output that cannot be read properly by LOAD DATA INFILE. For example, the preceding output just shown would appear as follows if the escape character is empty. Observe that the second field in the fourth line contains a comma following the quote, which (erroneously) appears to terminate the field:

```
1,"a string",100.20
```

```
2,"a string containing a , comma",102.20
3,"a string containing a " quote",102.20
4,"a string containing a ", quote and comma",102.20
```

For input, the ENCLOSED BY character, if present, is stripped from the ends of field values. (This is true regardless of whether OPTIONALLY is specified; OPTIONALLY has no effect on input interpretation.) Occurrences of the ENCLOSED BY character preceded by the ESCAPED BY character are interpreted as part of the current field value.

If the field begins with the ENCLOSED BY character, instances of that character are recognized as terminating a field value only if followed by the field or line TERMINATED BY sequence. To avoid ambiguity, occurrences of the ENCLOSED BY character within a field value can be doubled and are interpreted as a single instance of the character. For example, if ENCLOSED BY '"' is specified, quotation marks are handled as shown here:

```
"The ""BIG"" boss"  -> The "BIG" boss
The "BIG" boss      -> The "BIG" boss
The ""BIG"" boss    -> The ""BIG"" boss
```

FIELDS ESCAPED BY controls how to read or write special characters:

- For input, if the FIELDS ESCAPED BY character is not empty, occurrences of that character are stripped and the following character is taken literally as part of a field value. Some two-character sequences that are exceptions, where the first character is the escape character. These sequences are shown in the following table (using "\" for the escape character). The rules for NULL handling are described later in this section.

Character	Escape Sequence
\0	An ASCII NUL (0x00) character
\b	A backspace character
\n	A newline (linefeed) character
\r	A carriage return character
\t	A tab character.
\Z	ASCII 26 (Control+Z)
\N	NULL

  For more information about "\"-escape syntax, see Section 9.1.1, "String Literals".

  If the FIELDS ESCAPED BY character is empty, escape-sequence interpretation does not occur.

- For output, if the FIELDS ESCAPED BY character is not empty, it is used to prefix the following characters on output:

  - The FIELDS ESCAPED BY character
  - The FIELDS [OPTIONALLY] ENCLOSED BY character
  - The first character of the FIELDS TERMINATED BY and LINES TERMINATED BY values
  - ASCII 0 (what is actually written following the escape character is ASCII "0", not a zero-valued byte)

  If the FIELDS ESCAPED BY character is empty, no characters are escaped and NULL is output as NULL, not \N. It is probably not a good idea to specify an empty escape character, particularly if field values in your data contain any of the characters in the list just given.

In certain cases, field- and line-handling options interact:

- If LINES TERMINATED BY is an empty string and FIELDS TERMINATED BY is nonempty, lines are also terminated with FIELDS TERMINATED BY.
- If the FIELDS TERMINATED BY and FIELDS ENCLOSED BY values are both empty (''), a fixed-row (nondelimited) format is used. With fixed-row format, no delimiters are used between fields (but you can still have a line terminator). Instead, column values are read and written using a field width wide enough to hold all values in the field. For TINYINT, SMALLINT, MEDIUMINT, INT, and BIGINT, the field widths are 4, 6, 8, 11, and 20, respectively, no matter what the declared display width is.

  LINES TERMINATED BY is still used to separate lines. If a line does not contain all fields, the rest of the columns are set to their default values. If you do not have a line terminator, you should set this to ''. In this case, the text file must contain all fields for each row.

  Fixed-row format also affects handling of NULL values, as described later. Note that fixed-size format does not work if you are using a multi-byte character set.

Handling of NULL values varies according to the FIELDS and LINES options in use:

- For the default FIELDS and LINES values, NULL is written as a field value of \N for output, and a field value of \N is read as NULL for input (assuming that the ESCAPED BY character is “\”).
- If FIELDS ENCLOSED BY is not empty, a field containing the literal word NULL as its value is read as a NULL value. This differs from the word NULL enclosed within FIELDS ENCLOSED BY characters, which is read as the string 'NULL'.
- If FIELDS ESCAPED BY is empty, NULL is written as the word NULL.
- With fixed-row format (which is used when FIELDS TERMINATED BY and FIELDS ENCLOSED BY are both empty), NULL is written as an empty string. Note that this causes both NULL values and empty strings in the table to be indistinguishable when written to the file because both are written as empty strings. If you need to be able to tell the two apart when reading the file back in, you should not use fixed-row format.

An attempt to load NULL into a NOT NULL column causes assignment of the implicit default value for the column's data type and a warning, or an error in strict SQL mode. Implicit default values are discussed in Section 11.5, “Data Type Default Values”.

Some cases are not supported by LOAD DATA INFILE:

- Fixed-size rows (FIELDS TERMINATED BY and FIELDS ENCLOSED BY both empty) and BLOB or TEXT columns.
- If you specify one separator that is the same as or a prefix of another, LOAD DATA INFILE cannot interpret the input properly. For example, the following FIELDS clause would cause problems:

  ```
  FIELDS TERMINATED BY '"' ENCLOSED BY '"'
  ```

- If FIELDS ESCAPED BY is empty, a field value that contains an occurrence of FIELDS ENCLOSED BY or LINES TERMINATED BY followed by the FIELDS TERMINATED BY value causes LOAD DATA INFILE to stop reading a field or line too early. This happens because LOAD DATA INFILE cannot properly determine where the field or line value ends.

The following example loads all columns of the persondata table:

```
LOAD DATA INFILE 'persondata.txt' INTO TABLE persondata;
```

By default, when no column list is provided at the end of the LOAD DATA INFILE statement, input lines are expected to contain a field for each table column. If you want to load only some of a table's columns, specify a column list:

```
LOAD DATA INFILE 'persondata.txt' INTO TABLE persondata (col1,col2,...);
```

You must also specify a column list if the order of the fields in the input file differs from the order of the columns in the table. Otherwise, MySQL cannot tell how to match input fields with table columns.

The column list can contain either column names or user variables. With user variables, the SET clause enables you to perform transformations on their values before assigning the result to columns.

User variables in the SET clause can be used in several ways. The following example uses the first input column directly for the value of t1.column1, and assigns the second input column to a user variable that is subjected to a division operation before being used for the value of t1.column2:

```
LOAD DATA INFILE 'file.txt'
  INTO TABLE t1
  (column1, @var1)
  SET column2 = @var1/100;
```

The SET clause can be used to supply values not derived from the input file. The following statement sets column3 to the current date and time:

```
LOAD DATA INFILE 'file.txt'
  INTO TABLE t1
  (column1, column2)
  SET column3 = CURRENT_TIMESTAMP;
```

You can also discard an input value by assigning it to a user variable and not assigning the variable to a table column:

```
LOAD DATA INFILE 'file.txt'
  INTO TABLE t1
  (column1, @dummy, column2, @dummy, column3);
```

Use of the column/variable list and SET clause is subject to the following restrictions:

- Assignments in the SET clause should have only column names on the left hand side of assignment operators.
- You can use subqueries in the right hand side of SET assignments. A subquery that returns a value to be assigned to a column may be a scalar subquery only. Also, you cannot use a subquery to select from the table that is being loaded.
- Lines ignored by an IGNORE clause are not processed for the column/variable list or SET clause.
- User variables cannot be used when loading data with fixed-row format because user variables do not have a display width.

When processing an input line, LOAD DATA splits it into fields and uses the values according to the column/variable list and the SET clause, if they are present. Then the resulting row is inserted into the table. If there are BEFORE INSERT or AFTER INSERT triggers for the table, they are activated before or after inserting the row, respectively.

If an input line has too many fields, the extra fields are ignored and the number of warnings is incremented.

If an input line has too few fields, the table columns for which input fields are missing are set to their default values. Default value assignment is described in Section 11.5, “Data Type Default Values”.

An empty field value is interpreted different from a missing field:

- For string types, the column is set to the empty string.
- For numeric types, the column is set to 0.
- For date and time types, the column is set to the appropriate “zero” value for the type. See Section 11.3, “Date and Time Types”.

These are the same values that result if you assign an empty string explicitly to a string, numeric, or date or time type explicitly in an INSERT or UPDATE statement.

Treatment of empty or incorrect field values differs from that just described if the SQL mode is set to a restrictive value. For example, if sql_mode='TRADITIONAL, conversion of an empty value or a value such as 'x' for a numeric column results in an error, not conversion to 0. (With LOCAL, warnings occur rather than errors, even with a restrictive sql_mode value, because the server has no way to stop transmission of the file in the middle of the operation.)

TIMESTAMP columns are set to the current date and time only if there is a NULL value for the column (that is, \N) and the column is not declared to permit NULL values, or if the TIMESTAMP column's default value is the current timestamp and it is omitted from the field list when a field list is specified.

LOAD DATA INFILE regards all input as strings, so you cannot use numeric values for ENUM or SET columns the way you can with INSERT statements. All ENUM and SET values must be specified as strings.

BIT values cannot be loaded using binary notation (for example, b'011010'). To work around this, specify the values as regular integers and use the SET clause to convert them so that MySQL performs a numeric type conversion and loads them into the BIT column properly:

```
shell> cat /tmp/bit_test.txt
2
127
shell> mysql test
mysql> LOAD DATA INFILE '/tmp/bit_test.txt'
    -> INTO TABLE bit_test (@var1) SET b = CAST(@var1 AS UNSIGNED);
Query OK, 2 rows affected (0.00 sec)
Records: 2  Deleted: 0  Skipped: 0  Warnings: 0

mysql> SELECT BIN(b+0) FROM bit_test;
+----------+
| bin(b+0) |
+----------+
| 10       |
| 1111111  |
+----------+
2 rows in set (0.00 sec)
```

On Unix, if you need LOAD DATA to read from a pipe, you can use the following technique (the example loads a listing of the / directory into the table db1.t1):

```
mkfifo /mysql/data/db1/ls.dat
chmod 666 /mysql/data/db1/ls.dat
find / -ls > /mysql/data/db1/ls.dat &
```

```
mysql -e "LOAD DATA INFILE 'ls.dat' INTO TABLE t1" db1
```

Note that you must run the command that generates the data to be loaded and the mysql commands either on separate terminals, or run the data generation process in the background (as shown in the preceding example). If you do not do this, the pipe will block until data is read by the mysql process.

When the LOAD DATA INFILE statement finishes, it returns an information string in the following format:

```
Records: 1  Deleted: 0  Skipped: 0  Warnings: 0
```

Warnings occur under the same circumstances as when values are inserted using the INSERT statement (see Section 13.2.5, “INSERT Syntax”), except that LOAD DATA INFILE also generates warnings when there are too few or too many fields in the input row.

You can use SHOW WARNINGS to get a list of the first max_error_count warnings as information about what went wrong. See Section 13.7.5.39, “SHOW WARNINGS Syntax”.

If you are using the C API, you can get information about the statement by calling the mysql_info() function. See Section 21.8.7.36, “mysql_info()”.

13.2.7 LOAD XML Syntax

```
LOAD XML [LOW_PRIORITY | CONCURRENT] [LOCAL] INFILE 'file_name'
    [REPLACE | IGNORE]
    INTO TABLE [db_name.]tbl_name
    [PARTITION (partition_name,...)]
    [CHARACTER SET charset_name]
    [ROWS IDENTIFIED BY '<tagname>']
    [IGNORE number {LINES | ROWS}]
    [(column_or_user_var,...)]
    [SET col_name = expr,...]
```

The LOAD XML statement reads data from an XML file into a table. The *file_name* must be given as a literal string. The *tagname* in the optional ROWS IDENTIFIED BY clause must also be given as a literal string, and must be surrounded by angle brackets (< and >).

LOAD XML acts as the complement of running the mysql client in XML output mode (that is, starting the client with the --xml option). To write data from a table to an XML file, use a command such as the following one from the system shell:

```
shell> mysql --xml -e 'SELECT * FROM mytable' > file.xml
```

To read the file back into a table, use LOAD XML INFILE. By default, the <row> element is considered to be the equivalent of a database table row; this can be changed using the ROWS IDENTIFIED BY clause.

This statement supports three different XML formats:

- Column names as attributes and column values as attribute values:

  ```
  <row column1="value1" column2="value2" .../>
  ```

- Column names as tags and column values as the content of these tags:

  ```
  <row>
  ```

```
  <column1>value1</column1>
  <column2>value2</column2>
</row>
```

- Column names are the `name` attributes of `<field>` tags, and values are the contents of these tags:

  ```
  <row>
    <field name='column1'>value1</field>
    <field name='column2'>value2</field>
  </row>
  ```

  This is the format used by other MySQL tools, such as `mysqldump`.

All three formats can be used in the same XML file; the import routine automatically detects the format for each row and interprets it correctly. Tags are matched based on the tag or attribute name and the column name.

The following clauses work essentially the same way for `LOAD XML` as they do for `LOAD DATA`:

- `LOW_PRIORITY` or `CONCURRENT`
- `LOCAL`
- `REPLACE` or `IGNORE`
- `PARTITION`
- `CHARACTER SET`
- `(column_or_user_var,...)`
- `SET`

See Section 13.2.6, “`LOAD DATA INFILE` Syntax”, for more information about these clauses.

The `IGNORE number LINES` or `IGNORE number ROWS` clause causes the first *`number`* rows in the XML file to be skipped. It is analogous to the `LOAD DATA` statement's `IGNORE ... LINES` clause.

To illustrate how this statement is used, suppose that we have a table created as follows:

```
USE test;

CREATE TABLE person (
    person_id INT NOT NULL PRIMARY KEY,
    fname VARCHAR(40) NULL,
    lname VARCHAR(40) NULL,
    created TIMESTAMP
);
```

Suppose further that this table is initially empty.

Now suppose that we have a simple XML file `person.xml`, whose contents are as shown here:

```
<?xml version="1.0"?>
<list>
  <person person_id="1" fname="Pekka" lname="Nousiainen"/>
  <person person_id="2" fname="Jonas" lname="Oreland"/>
  <person person_id="3"><fname>Mikael</fname><lname>Ronström</lname></person>
```

```
  <person person_id="4"><fname>Lars</fname><lname>Thalmann</lname></person>
  <person><field name="person_id">5</field><field name="fname">Tomas</field>
  <field name="lname">Ulin</field></person>
  <person><field name="person_id">6</field><field name="fname">Martin</field>
  <field name="lname">Sköld</field></person>
</list>
```

Each of the permissible XML formats discussed previously is represented in this example file.

To import the data in person.xml into the person table, you can use this statement:

```
mysql> LOAD XML LOCAL INFILE 'person.xml'
    ->   INTO TABLE person
    ->   ROWS IDENTIFIED BY '<person>';

Query OK, 6 rows affected (0.00 sec)
Records: 6  Deleted: 0  Skipped: 0  Warnings: 0
```

Here, we assume that person.xml is located in the MySQL data directory. If the file cannot be found, the following error results:

```
ERROR 2 (HY000): File '/person.xml' not found (Errcode: 2)
```

The ROWS IDENTIFIED BY '<person>' clause means that each <person> element in the XML file is considered equivalent to a row in the table into which the data is to be imported. In this case, this is the person table in the test database.

As can be seen by the response from the server, 6 rows were imported into the test.person table. This can be verified by a simple SELECT statement:

```
mysql> SELECT * FROM person;
+-----------+--------+------------+---------------------+
| person_id | fname  | lname      | created             |
+-----------+--------+------------+---------------------+
|         1 | Pekka  | Nousiainen | 2007-07-13 16:18:47 |
|         2 | Jonas  | Oreland    | 2007-07-13 16:18:47 |
|         3 | Mikael | Ronström   | 2007-07-13 16:18:47 |
|         4 | Lars   | Thalmann   | 2007-07-13 16:18:47 |
|         5 | Tomas  | Ulin       | 2007-07-13 16:18:47 |
|         6 | Martin | Sköld      | 2007-07-13 16:18:47 |
+-----------+--------+------------+---------------------+
6 rows in set (0.00 sec)
```

This shows, as stated earlier in this section, that any or all of the 3 permitted XML formats may appear in a single file and be read in using LOAD XML.

The inverse of the above operation—that is, dumping MySQL table data into an XML file—can be accomplished using the mysql client from the system shell, as shown here:

Note

The --xml option causes the mysql client to use XML formatting for its output; the -e option causes the client to execute the SQL statement immediately following the option.

```
shell> mysql --xml -e "SELECT * FROM test.person" > person-dump.xml
shell> cat person-dump.xml
```

```
<?xml version="1.0"?>

<resultset statement="SELECT * FROM test.person" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <row>
        <field name="person_id">1</field>
        <field name="fname">Pekka</field>
        <field name="lname">Nousiainen</field>
        <field name="created">2007-07-13 16:18:47</field>
  </row>

  <row>
        <field name="person_id">2</field>
        <field name="fname">Jonas</field>
        <field name="lname">Oreland</field>
        <field name="created">2007-07-13 16:18:47</field>
  </row>

  <row>
        <field name="person_id">3</field>
        <field name="fname">Mikael</field>
        <field name="lname">Ronström</field>
        <field name="created">2007-07-13 16:18:47</field>
  </row>

  <row>
        <field name="person_id">4</field>
        <field name="fname">Lars</field>
        <field name="lname">Thalmann</field>
        <field name="created">2007-07-13 16:18:47</field>
  </row>

  <row>
        <field name="person_id">5</field>
        <field name="fname">Tomas</field>
        <field name="lname">Ulin</field>
        <field name="created">2007-07-13 16:18:47</field>
  </row>

  <row>
        <field name="person_id">6</field>
        <field name="fname">Martin</field>
        <field name="lname">Sköld</field>
        <field name="created">2007-07-13 16:18:47</field>
  </row>
</resultset>
```

You can verify that the dump is valid by creating a copy of the `person` and then importing the dump file into the new table, like this:

```

mysql> USE test;
mysql> CREATE TABLE person2 LIKE person;
Query OK, 0 rows affected (0.00 sec)

mysql> LOAD XML LOCAL INFILE 'person-dump.xml'
    ->   INTO TABLE person2;
Query OK, 6 rows affected (0.01 sec)
Records: 6  Deleted: 0  Skipped: 0  Warnings: 0

mysql> SELECT * FROM person2;
+-----------+--------+------------+---------------------+
| person_id | fname  | lname      | created             |
+-----------+--------+------------+---------------------+
|         1 | Pekka  | Nousiainen | 2007-07-13 16:18:47 |
|         2 | Jonas  | Oreland    | 2007-07-13 16:18:47 |
|         3 | Mikael | Ronström   | 2007-07-13 16:18:47 |
|         4 | Lars   | Thalmann   | 2007-07-13 16:18:47 |
|         5 | Tomas  | Ulin       | 2007-07-13 16:18:47 |
|         6 | Martin | Sköld      | 2007-07-13 16:18:47 |
+-----------+--------+------------+---------------------+
6 rows in set (0.00 sec)
```

Using a `ROWS IDENTIFIED BY '<tagname>'` clause, it is possible to import data from the same XML file into database tables with different definitions. For this example, suppose that you have a file named `address.xml` which contains the following XML:

```
<?xml version="1.0"?>

<list>
  <person person_id="1">
    <fname>Robert</fname>
    <lname>Jones</lname>
    <address address_id="1" street="Mill Creek Road" zip="45365" city="Sidney"/>
    <address address_id="2" street="Main Street" zip="28681" city="Taylorsville"/>
  </person>

  <person person_id="2">
    <fname>Mary</fname>
    <lname>Smith</lname>
    <address address_id="3" street="River Road" zip="80239" city="Denver"/>
    <!-- <address address_id="4" street="North Street" zip="37920" city="Knoxville"/> -->
  </person>

</list>
```

You can again use the `test.person` table as defined previously in this section, after clearing all the existing records from the table and then showing its structure as shown here:

```
mysql< TRUNCATE person;
Query OK, 0 rows affected (0.04 sec)

mysql< SHOW CREATE TABLE person\G
*************************** 1. row ***************************
       Table: person
Create Table: CREATE TABLE `person` (
  `person_id` int(11) NOT NULL,
  `fname` varchar(40) DEFAULT NULL,
  `lname` varchar(40) DEFAULT NULL,
  `created` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  PRIMARY KEY (`person_id`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1
1 row in set (0.00 sec)
```

Now create an `address` table in the `test` database using the following `CREATE TABLE` statement:

```
CREATE TABLE address (
    address_id INT NOT NULL PRIMARY KEY,
    person_id INT NULL,
    street VARCHAR(40) NULL,
    zip INT NULL,
    city VARCHAR(40) NULL,
    created TIMESTAMP
);
```

To import the data from the XML file into the `person` table, execute the following `LOAD XML` statement, which specifies that rows are to be specified by the `<person>` element, as shown here;

```
mysql> LOAD XML LOCAL INFILE 'address.xml'
```

```
    ->   INTO TABLE person
    ->   ROWS IDENTIFIED BY '<person>';
Query OK, 2 rows affected (0.00 sec)
Records: 2  Deleted: 0  Skipped: 0  Warnings: 0
```

You can verify that the records were imported using a SELECT statement:

```
mysql> SELECT * FROM person;
+-----------+--------+-------+---------------------+
| person_id | fname  | lname | created             |
+-----------+--------+-------+---------------------+
|         1 | Robert | Jones | 2007-07-24 17:37:06 |
|         2 | Mary   | Smith | 2007-07-24 17:37:06 |
+-----------+--------+-------+---------------------+
2 rows in set (0.00 sec)
```

Since the <address> elements in the XML file have no corresponding columns in the person table, they are skipped.

To import the data from the <address> elements into the address table, use the LOAD XML statement shown here:

```
mysql> LOAD XML LOCAL INFILE 'address.xml'
    ->   INTO TABLE address
    ->   ROWS IDENTIFIED BY '<address>';
Query OK, 3 rows affected (0.00 sec)
Records: 3  Deleted: 0  Skipped: 0  Warnings: 0
```

You can see that the data was imported using a SELECT statement such as this one:

```
mysql> SELECT * FROM address;
+------------+-----------+-----------------+-------+--------------+---------------------+
| address_id | person_id | street          | zip   | city         | created             |
+------------+-----------+-----------------+-------+--------------+---------------------+
|          1 |         1 | Mill Creek Road | 45365 | Sidney       | 2007-07-24 17:37:37 |
|          2 |         1 | Main Street     | 28681 | Taylorsville | 2007-07-24 17:37:37 |
|          3 |         2 | River Road      | 80239 | Denver       | 2007-07-24 17:37:37 |
+------------+-----------+-----------------+-------+--------------+---------------------+
3 rows in set (0.00 sec)
```

The data from the <address> element that is enclosed in XML comments is not imported. However, since there is a person_id column in the address table, the value of the person_id attribute from the parent <person> element for each <address> *is* imported into the address table.

Security Considerations. As with the LOAD DATA statement, the transfer of the XML file from the client host to the server host is initiated by the MySQL server. In theory, a patched server could be built that would tell the client program to transfer a file of the server's choosing rather than the file named by the client in the LOAD XML statement. Such a server could access any file on the client host to which the client user has read access.

In a Web environment, clients usually connect to MySQL from a Web server. A user that can run any command against the MySQL server can use LOAD XML LOCAL to read any files to which the Web server process has read access. In this environment, the client with respect to the MySQL server is actually the Web server, not the remote program being run by the user who connects to the Web server.

You can disable loading of XML files from clients by starting the server with --local-infile=0 or --local-infile=OFF. This option can also be used when starting the mysql client to disable LOAD XML for the duration of the client session.

To prevent a client from loading XML files from the server, do not grant the FILE privilege to the corresponding MySQL user account, or revoke this privilege if the client user account already has it.

> ⚠ **Important**
>
> Revoking the FILE privilege (or not granting it in the first place) keeps the user only from executing the LOAD XML INFILE statement (as well as the LOAD_FILE() function; it does *not* prevent the user from executing LOAD XML LOCAL INFILE. To disallow this statement, you must start the server or the client with --local-infile=OFF.
>
> In other words, the FILE privilege affects only whether the client can read files on the server; it has no bearing on whether the client can read files on the local file system.

For partitioned tables using storage engines that employ table locks, such as MyISAM, LOAD XML cannot prune any partition locks. This does not apply to tables using storage engines which employ row-level locking, such as InnoDB. For more information, see Section 17.6.4, "Partitioning and Locking".

13.2.8 REPLACE Syntax

```
REPLACE [LOW_PRIORITY | DELAYED]
    [INTO] tbl_name
    [PARTITION (partition_name,...)]
    [(col_name,...)]
    {VALUES | VALUE} ({expr | DEFAULT},...),(...),...
```

Or:

```
REPLACE [LOW_PRIORITY | DELAYED]
    [INTO] tbl_name
    [PARTITION (partition_name,...)]
    SET col_name={expr | DEFAULT}, ...
```

Or:

```
REPLACE [LOW_PRIORITY | DELAYED]
    [INTO] tbl_name
    [PARTITION (partition_name,...)]
    [(col_name,...)]
    SELECT ...
```

REPLACE works exactly like INSERT, except that if an old row in the table has the same value as a new row for a PRIMARY KEY or a UNIQUE index, the old row is deleted before the new row is inserted. See Section 13.2.5, "INSERT Syntax".

REPLACE is a MySQL extension to the SQL standard. It either inserts, or *deletes* and inserts. For another MySQL extension to standard SQL—that either inserts or *updates*—see Section 13.2.5.3, "INSERT ... ON DUPLICATE KEY UPDATE Syntax".

DELAYED inserts and replaces were deprecated in MySQL 5.6.6. In MySQL 5.7, DELAYED is not supported. The server recognizes but ignores the DELAYED keyword, handles the replace as a nondelayed replace, and generates an ER_WARN_LEGACY_SYNTAX_CONVERTED warning. ("REPLACE DELAYED is no longer supported. The statement was converted to REPLACE.") The DELAYED keyword will be removed in a future release.

Note that unless the table has a PRIMARY KEY or UNIQUE index, using a REPLACE statement makes no sense. It becomes equivalent to INSERT, because there is no index to be used to determine whether a new row duplicates another.

Values for all columns are taken from the values specified in the REPLACE statement. Any missing columns are set to their default values, just as happens for INSERT. You cannot refer to values from the current row and use them in the new row. If you use an assignment such as SET *col_name* = *col_name* + 1, the reference to the column name on the right hand side is treated as DEFAULT(*col_name*), so the assignment is equivalent to SET *col_name* = DEFAULT(*col_name*) + 1.

To use REPLACE, you must have both the INSERT and DELETE privileges for the table.

REPLACE supports explicit partition selection using the PARTITION keyword with a comma-separated list of names of partitions, subpartitions, or both. As with INSERT, if it is not possible to insert the new row into any of these partitions or subpartitions, the REPLACE statement fails with the error Found a row not matching the given partition set. See Section 17.5, “Partition Selection”, for more information.

The REPLACE statement returns a count to indicate the number of rows affected. This is the sum of the rows deleted and inserted. If the count is 1 for a single-row REPLACE, a row was inserted and no rows were deleted. If the count is greater than 1, one or more old rows were deleted before the new row was inserted. It is possible for a single row to replace more than one old row if the table contains multiple unique indexes and the new row duplicates values for different old rows in different unique indexes.

The affected-rows count makes it easy to determine whether REPLACE only added a row or whether it also replaced any rows: Check whether the count is 1 (added) or greater (replaced).

If you are using the C API, the affected-rows count can be obtained using the mysql_affected_rows() function.

Currently, you cannot replace into a table and select from the same table in a subquery.

MySQL uses the following algorithm for REPLACE (and LOAD DATA ... REPLACE):

1. Try to insert the new row into the table

2. While the insertion fails because a duplicate-key error occurs for a primary key or unique index:

   a. Delete from the table the conflicting row that has the duplicate key value

   b. Try again to insert the new row into the table

It is possible that in the case of a duplicate-key error, a storage engine may perform the REPLACE as an update rather than a delete plus insert, but the semantics are the same. There are no user-visible effects other than a possible difference in how the storage engine increments Handler_*xxx* status variables.

Because the results of REPLACE ... SELECT statements depend on the ordering of rows from the SELECT and this order cannot always be guaranteed, it is possible when logging these statements for the master and the slave to diverge. For this reason, REPLACE ... SELECT statements are flagged as unsafe for statement-based replication. With this change, such statements produce a warning in the log when using the STATEMENT binary logging mode, and are logged using the row-based format when using MIXED mode. See also Section 16.1.2.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”.

In MySQL 5.7, a REPLACE statement affecting a partitioned table using a storage engine such as MyISAM that employs table-level locks locks only those partitions containing rows that match the REPLACE statement's WHERE clause, as long as none of the table's partitioning columns are updated; otherwise the entire table is locked. (For storage engines such as InnoDB that employ row-level locking, no locking of partitions takes place.) For more information, see Section 17.6.4, “Partitioning and Locking”.

13.2.9 SELECT Syntax

```
SELECT
    [ALL | DISTINCT | DISTINCTROW ]
      [HIGH_PRIORITY]
      [MAX_STATEMENT_TIME]
      [STRAIGHT_JOIN]
      [SQL_SMALL_RESULT] [SQL_BIG_RESULT] [SQL_BUFFER_RESULT]
      [SQL_CACHE | SQL_NO_CACHE] [SQL_CALC_FOUND_ROWS]
    select_expr [, select_expr ...]
    [FROM table_references
      [PARTITION partition_list]
    [WHERE where_condition]
    [GROUP BY {col_name | expr | position}
      [ASC | DESC], ... [WITH ROLLUP]]
    [HAVING where_condition]
    [ORDER BY {col_name | expr | position}
      [ASC | DESC], ...]
    [LIMIT {[offset,] row_count | row_count OFFSET offset}]
    [PROCEDURE procedure_name(argument_list)]
    [INTO OUTFILE 'file_name'
        [CHARACTER SET charset_name]
        export_options
      | INTO DUMPFILE 'file_name'
      | INTO var_name [, var_name]]
    [FOR UPDATE | LOCK IN SHARE MODE]]
```

SELECT is used to retrieve rows selected from one or more tables, and can include UNION statements and subqueries. See Section 13.2.9.4, “UNION Syntax”, and Section 13.2.10, “Subquery Syntax”.

The most commonly used clauses of SELECT statements are these:

- Each *select_expr* indicates a column that you want to retrieve. There must be at least one *select_expr*.

- *table_references* indicates the table or tables from which to retrieve rows. Its syntax is described in Section 13.2.9.2, “JOIN Syntax”.

- SELECT supports explicit partition selection using the PARTITION with a list of partitions or subpartitions (or both) following the name of the table in a *table_reference* (see Section 13.2.9.2, “JOIN Syntax”). In this case, rows are selected only from the partitions listed, and any other partitions of the table are ignored. For more information and examples, see Section 17.5, “Partition Selection”.

  SELECT ... PARTITION from tables using storage engines such as MyISAM that perform table-level locks (and thus partition locks) lock only the partitions or subpartitions named by the PARTITION option.

  See Section 17.6.4, “Partitioning and Locking”, for more information.

- The WHERE clause, if given, indicates the condition or conditions that rows must satisfy to be selected. *where_condition* is an expression that evaluates to true for each row to be selected. The statement selects all rows if there is no WHERE clause.

  In the WHERE expression, you can use any of the functions and operators that MySQL supports, except for aggregate (summary) functions. See Section 9.5, “Expression Syntax”, and Chapter 12, *Functions and Operators*.

SELECT can also be used to retrieve rows computed without reference to any table.

For example:

```
mysql> SELECT 1 + 1;
        -> 2
```

You are permitted to specify `DUAL` as a dummy table name in situations where no tables are referenced:

```
mysql> SELECT 1 + 1 FROM DUAL;
        -> 2
```

`DUAL` is purely for the convenience of people who require that all `SELECT` statements should have `FROM` and possibly other clauses. MySQL may ignore the clauses. MySQL does not require `FROM DUAL` if no tables are referenced.

In general, clauses used must be given in exactly the order shown in the syntax description. For example, a `HAVING` clause must come after any `GROUP BY` clause and before any `ORDER BY` clause. The exception is that the `INTO` clause can appear either as shown in the syntax description or immediately following the *`select_expr`* list. For more information about `INTO`, see Section 13.2.9.1, “`SELECT ... INTO` Syntax”.

The list of *`select_expr`* terms comprises the select list that indicates which columns to retrieve. Terms specify a column or expression or can use `*`-shorthand:

- A select list consisting only of a single unqualified `*` can be used as shorthand to select all columns from all tables:

  ```
  SELECT * FROM t1 INNER JOIN t2 ...
  ```

- *`tbl_name`*`.*` can be used as a qualified shorthand to select all columns from the named table:

  ```
  SELECT t1.*, t2.* FROM t1 INNER JOIN t2 ...
  ```

- Use of an unqualified `*` with other items in the select list may produce a parse error. To avoid this problem, use a qualified *`tbl_name`*`.*` reference

  ```
  SELECT AVG(score), t1.* FROM t1 ...
  ```

The following list provides additional information about other `SELECT` clauses:

- A *`select_expr`* can be given an alias using `AS` *`alias_name`*. The alias is used as the expression's column name and can be used in `GROUP BY`, `ORDER BY`, or `HAVING` clauses. For example:

  ```
  SELECT CONCAT(last_name,', ',first_name) AS full_name
    FROM mytable ORDER BY full_name;
  ```

  The `AS` keyword is optional when aliasing a *`select_expr`* with an identifier. The preceding example could have been written like this:

  ```
  SELECT CONCAT(last_name,', ',first_name) full_name
    FROM mytable ORDER BY full_name;
  ```

  However, because the `AS` is optional, a subtle problem can occur if you forget the comma between two *`select_expr`* expressions: MySQL interprets the second as an alias name. For example, in the following statement, `columnb` is treated as an alias name:

```
SELECT columna columnb FROM mytable;
```

For this reason, it is good practice to be in the habit of using AS explicitly when specifying column aliases.

It is not permissible to refer to a column alias in a WHERE clause, because the column value might not yet be determined when the WHERE clause is executed. See Section C.5.5.4, "Problems with Column Aliases".

- The FROM *table_references* clause indicates the table or tables from which to retrieve rows. If you name more than one table, you are performing a join. For information on join syntax, see Section 13.2.9.2, "JOIN Syntax". For each table specified, you can optionally specify an alias.

  ```
  tbl_name [[AS] alias] [index_hint]
  ```

  The use of index hints provides the optimizer with information about how to choose indexes during query processing. For a description of the syntax for specifying these hints, see Section 13.2.9.3, "Index Hint Syntax".

  You can use SET max_seeks_for_key=*value* as an alternative way to force MySQL to prefer key scans instead of table scans. See Section 5.1.4, "Server System Variables".

- You can refer to a table within the default database as *tbl_name*, or as *db_name*.*tbl_name* to specify a database explicitly. You can refer to a column as *col_name*, *tbl_name*.*col_name*, or *db_name*.*tbl_name*.*col_name*. You need not specify a *tbl_name* or *db_name*.*tbl_name* prefix for a column reference unless the reference would be ambiguous. See Section 9.2.1, "Identifier Qualifiers", for examples of ambiguity that require the more explicit column reference forms.

- A table reference can be aliased using *tbl_name* AS *alias_name* or *tbl_name alias_name*:

  ```
  SELECT t1.name, t2.salary FROM employee AS t1, info AS t2
    WHERE t1.name = t2.name;

  SELECT t1.name, t2.salary FROM employee t1, info t2
    WHERE t1.name = t2.name;
  ```

- Columns selected for output can be referred to in ORDER BY and GROUP BY clauses using column names, column aliases, or column positions. Column positions are integers and begin with 1:

  ```
  SELECT college, region, seed FROM tournament
    ORDER BY region, seed;

  SELECT college, region AS r, seed AS s FROM tournament
    ORDER BY r, s;

  SELECT college, region, seed FROM tournament
    ORDER BY 2, 3;
  ```

  To sort in reverse order, add the DESC (descending) keyword to the name of the column in the ORDER BY clause that you are sorting by. The default is ascending order; this can be specified explicitly using the ASC keyword.

  If ORDER BY occurs within a subquery and also is applied in the outer query, the outermost ORDER BY takes precedence. For example, results for the following statement are sorted in descending order, not ascending order:

```
(SELECT ... ORDER BY a) ORDER BY a DESC;
```

Use of column positions is deprecated because the syntax has been removed from the SQL standard.

- If you use `GROUP BY`, output rows are sorted according to the `GROUP BY` columns as if you had an `ORDER BY` for the same columns. To avoid the overhead of sorting that `GROUP BY` produces, add `ORDER BY NULL`:

  ```
  SELECT a, COUNT(b) FROM test_table GROUP BY a ORDER BY NULL;
  ```

  Relying on implicit `GROUP BY` sorting in MySQL 5.7 is deprecated. To achieve a specific sort order of grouped results, it is preferable to use an explicit `ORDER BY` clause. `GROUP BY` sorting is a MySQL extension that may change in a future release; for example, to make it possible for the optimizer to order groupings in whatever manner it deems most efficient and to avoid the sorting overhead.

- MySQL extends the `GROUP BY` clause so that you can also specify `ASC` and `DESC` after columns named in the clause:

  ```
  SELECT a, COUNT(b) FROM test_table GROUP BY a DESC;
  ```

- MySQL extends the use of `GROUP BY` to permit selecting fields that are not mentioned in the `GROUP BY` clause. If you are not getting the results that you expect from your query, please read the description of `GROUP BY` found in Section 12.17, "Functions and Modifiers for Use with `GROUP BY` Clauses".

- `GROUP BY` permits a `WITH ROLLUP` modifier. See Section 12.17.2, "`GROUP BY` Modifiers".

- The `HAVING` clause is applied nearly last, just before items are sent to the client, with no optimization. (`LIMIT` is applied after `HAVING`.)

  The SQL standard requires that `HAVING` must reference only columns in the `GROUP BY` clause or columns used in aggregate functions. However, MySQL supports an extension to this behavior, and permits `HAVING` to refer to columns in the `SELECT` list and columns in outer subqueries as well.

  If the `HAVING` clause refers to a column that is ambiguous, a warning occurs. In the following statement, `col2` is ambiguous because it is used as both an alias and a column name:

  ```
  SELECT COUNT(col1) AS col2 FROM t GROUP BY col2 HAVING col2 = 2;
  ```

  Preference is given to standard SQL behavior, so if a `HAVING` column name is used both in `GROUP BY` and as an aliased column in the output column list, preference is given to the column in the `GROUP BY` column.

- Do not use `HAVING` for items that should be in the `WHERE` clause. For example, do not write the following:

  ```
  SELECT col_name FROM tbl_name HAVING col_name > 0;
  ```

  Write this instead:

  ```
  SELECT col_name FROM tbl_name WHERE col_name > 0;
  ```

- The `HAVING` clause can refer to aggregate functions, which the `WHERE` clause cannot:

```
SELECT user, MAX(salary) FROM users
  GROUP BY user HAVING MAX(salary) > 10;
```

(This did not work in some older versions of MySQL.)

- MySQL permits duplicate column names. That is, there can be more than one *select_expr* with the same name. This is an extension to standard SQL. Because MySQL also permits GROUP BY and HAVING to refer to *select_expr* values, this can result in an ambiguity:

  ```
  SELECT 12 AS a, a FROM t GROUP BY a;
  ```

  In that statement, both columns have the name a. To ensure that the correct column is used for grouping, use different names for each *select_expr*.

- MySQL resolves unqualified column or alias references in ORDER BY clauses by searching in the *select_expr* values, then in the columns of the tables in the FROM clause. For GROUP BY or HAVING clauses, it searches the FROM clause before searching in the *select_expr* values. (For GROUP BY and HAVING, this differs from the pre-MySQL 5.0 behavior that used the same rules as for ORDER BY.)

- The LIMIT clause can be used to constrain the number of rows returned by the SELECT statement. LIMIT takes one or two numeric arguments, which must both be nonnegative integer constants, with these exceptions:

  - Within prepared statements, LIMIT parameters can be specified using ? placeholder markers.

  - Within stored programs, LIMIT parameters can be specified using integer-valued routine parameters or local variables.

  With two arguments, the first argument specifies the offset of the first row to return, and the second specifies the maximum number of rows to return. The offset of the initial row is 0 (not 1):

  ```
  SELECT * FROM tbl LIMIT 5,10;  # Retrieve rows 6-15
  ```

  To retrieve all rows from a certain offset up to the end of the result set, you can use some large number for the second parameter. This statement retrieves all rows from the 96th row to the last:

  ```
  SELECT * FROM tbl LIMIT 95,18446744073709551615;
  ```

  With one argument, the value specifies the number of rows to return from the beginning of the result set:

  ```
  SELECT * FROM tbl LIMIT 5;     # Retrieve first 5 rows
  ```

  In other words, LIMIT *row_count* is equivalent to LIMIT 0, *row_count*.

  For prepared statements, you can use placeholders. The following statements will return one row from the tbl table:

  ```
  SET @a=1;
  PREPARE STMT FROM 'SELECT * FROM tbl LIMIT ?';
  EXECUTE STMT USING @a;
  ```

  The following statements will return the second to sixth row from the tbl table:

  ```
  SET @skip=1; SET @numrows=5;
  PREPARE STMT FROM 'SELECT * FROM tbl LIMIT ?, ?';
  ```

```
EXECUTE STMT USING @skip, @numrows;
```

For compatibility with PostgreSQL, MySQL also supports the `LIMIT row_count OFFSET offset` syntax.

If `LIMIT` occurs within a subquery and also is applied in the outer query, the outermost `LIMIT` takes precedence. For example, the following statement produces two rows, not one:

```
(SELECT ... LIMIT 1) LIMIT 2;
```

- A `PROCEDURE` clause names a procedure that should process the data in the result set. For an example, see Section 8.4.2.4, "Using `PROCEDURE ANALYSE`", which describes `ANALYSE`, a procedure that can be used to obtain suggestions for optimal column data types that may help reduce table sizes.

- The `SELECT ... INTO` form of `SELECT` enables the query result to be written to a file or stored in variables. For more information, see Section 13.2.9.1, "`SELECT ... INTO` Syntax".

- If you use `FOR UPDATE` with a storage engine that uses page or row locks, rows examined by the query are write-locked until the end of the current transaction. Using `LOCK IN SHARE MODE` sets a shared lock that permits other transactions to read the examined rows but not to update or delete them. See Section 14.2.2.5, "Locking Reads (`SELECT ... FOR UPDATE` and `SELECT ... LOCK IN SHARE MODE`)".

  In addition, you cannot use `FOR UPDATE` as part of the `SELECT` in a statement such as `CREATE TABLE new_table SELECT ... FROM old_table ...`. (If you attempt to do so, the statement is rejected with the error `Can't update table 'old_table' while 'new_table' is being created`.) This is a change in behavior from MySQL 5.5 and earlier, which permitted `CREATE TABLE ... SELECT` statements to make changes in tables other than the table being created.

Following the `SELECT` keyword, you can use a number of options that affect the operation of the statement. `HIGH_PRIORITY`, `MAX_STATEMENT_TIME`, `STRAIGHT_JOIN`, and options beginning with `SQL_` are MySQL extensions to standard SQL.

- The `ALL` and `DISTINCT` options specify whether duplicate rows should be returned. `ALL` (the default) specifies that all matching rows should be returned, including duplicates. `DISTINCT` specifies removal of duplicate rows from the result set. It is an error to specify both options. `DISTINCTROW` is a synonym for `DISTINCT`.

- `HIGH_PRIORITY` gives the `SELECT` higher priority than a statement that updates a table. You should use this only for queries that are very fast and must be done at once. A `SELECT HIGH_PRIORITY` query that is issued while the table is locked for reading runs even if there is an update statement waiting for the table to be free. This affects only storage engines that use only table-level locking (such as `MyISAM`, `MEMORY`, and `MERGE`).

  `HIGH_PRIORITY` cannot be used with `SELECT` statements that are part of a `UNION`.

- `MAX_STATEMENT_TIME = N` sets a statement execution timeout of *N* milliseconds. If this option is absent or *N* is 0, the statement timeout established by the `max_statement_time` system variable applies.

  The `MAX_STATEMENT_TIME` option is applicable as follows:

  - It applies to top-level `SELECT` statements. It does not apply to non-top-level statements, such as subqueries.

  - It applies to read-only `SELECT` statements. Statements that are not read only are those that invoke a stored function that modifies data as a side effect.

- It does not apply to SELECT statements in stored programs; an error occurs.

This option was added in MySQL 5.7.4.

- STRAIGHT_JOIN forces the optimizer to join the tables in the order in which they are listed in the FROM clause. You can use this to speed up a query if the optimizer joins the tables in nonoptimal order. STRAIGHT_JOIN also can be used in the *table_references* list. See Section 13.2.9.2, “JOIN Syntax”.

  STRAIGHT_JOIN does not apply to any table that the optimizer treats as a const or system table. Such a table produces a single row, is read during the optimization phase of query execution, and references to its columns are replaced with the appropriate column values before query execution proceeds. These tables will appear first in the query plan displayed by EXPLAIN. See Section 8.8.1, “Optimizing Queries with EXPLAIN”. This exception may not apply to const or system tables that are used on the NULL-complemented side of an outer join (that is, the right-side table of a LEFT JOIN or the left-side table of a RIGHT JOIN.

- SQL_BIG_RESULT or SQL_SMALL_RESULT can be used with GROUP BY or DISTINCT to tell the optimizer that the result set has many rows or is small, respectively. For SQL_BIG_RESULT, MySQL directly uses disk-based temporary tables if needed, and prefers sorting to using a temporary table with a key on the GROUP BY elements. For SQL_SMALL_RESULT, MySQL uses fast temporary tables to store the resulting table instead of using sorting. This should not normally be needed.

- SQL_BUFFER_RESULT forces the result to be put into a temporary table. This helps MySQL free the table locks early and helps in cases where it takes a long time to send the result set to the client. This option can be used only for top-level SELECT statements, not for subqueries or following UNION.

- SQL_CALC_FOUND_ROWS tells MySQL to calculate how many rows there would be in the result set, disregarding any LIMIT clause. The number of rows can then be retrieved with SELECT FOUND_ROWS(). See Section 12.14, “Information Functions”.

- The SQL_CACHE and SQL_NO_CACHE options affect caching of query results in the query cache (see Section 8.9.3, “The MySQL Query Cache”). SQL_CACHE tells MySQL to store the result in the query cache if it is cacheable and the value of the query_cache_type system variable is 2 or DEMAND. With SQL_NO_CACHE, the server does not use the query cache. It neither checks the query cache to see whether the result is already cached, nor does it cache the query result.

  For views, SQL_NO_CACHE applies if it appears in any SELECT in the query. For a cacheable query, SQL_CACHE applies if it appears in the first SELECT of a view referred to by the query.

  In MySQL 5.7, these two options are mutually exclusive and an error occurs if they are both specified. Also, these options are not permitted in subqueries (including subqueries in the FROM clause), and SELECT statements in unions other than the first SELECT.

In MySQL 5.7, a SELECT from a partitioned table using a storage engine such as MyISAM that employs table-level locks locks only those partitions containing rows that match the SELECT statement's WHERE clause. (This does not occur with storage engines such as InnoDB that employ row-level locking.) For more information, see Section 17.6.4, “Partitioning and Locking”.

13.2.9.1 SELECT ... INTO Syntax

The SELECT ... INTO form of SELECT enables a query result to be stored in variables or written to a file:

- SELECT ... INTO *var_list* selects column values and stores them into variables.

- `SELECT ... INTO OUTFILE` writes the selected rows to a file. Column and line terminators can be specified to produce a specific output format.
- `SELECT ... INTO DUMPFILE` writes a single row to a file without any formatting.

The `SELECT` syntax description (see Section 13.2.9, "`SELECT` Syntax") shows the `INTO` clause near the end of the statement. It is also possible to use `INTO` immediately following the *`select_expr`* list.

An `INTO` clause should not be used in a nested `SELECT` because such a `SELECT` must return its result to the outer context.

The `INTO` clause can name a list of one or more variables, which can be user-defined variables, stored procedure or function parameters, or stored program local variables. (Within a prepared `SELECT ... INTO OUTFILE` statement, only user-defined variables are permitted;see Section 13.6.4.2, "Local Variable Scope and Resolution".)

The selected values are assigned to the variables. The number of variables must match the number of columns. The query should return a single row. If the query returns no rows, a warning with error code 1329 occurs (`No data`), and the variable values remain unchanged. If the query returns multiple rows, error 1172 occurs (`Result consisted of more than one row`). If it is possible that the statement may retrieve multiple rows, you can use `LIMIT 1` to limit the result set to a single row.

```
SELECT id, data INTO @x, @y FROM test.t1 LIMIT 1;
```

User variable names are not case sensitive. See Section 9.4, "User-Defined Variables".

The `SELECT ... INTO OUTFILE '`*`file_name`*`'` form of `SELECT` writes the selected rows to a file. The file is created on the server host, so you must have the `FILE` privilege to use this syntax. *`file_name`* cannot be an existing file, which among other things prevents files such as `/etc/passwd` and database tables from being destroyed. The `character_set_filesystem` system variable controls the interpretation of the file name.

The `SELECT ... INTO OUTFILE` statement is intended primarily to let you very quickly dump a table to a text file on the server machine. If you want to create the resulting file on some other host than the server host, you normally cannot use `SELECT ... INTO OUTFILE` since there is no way to write a path to the file relative to the server host's file system.

However, if the MySQL client software is installed on the remote machine, you can instead use a client command such as `mysql -e "SELECT ..." >` *`file_name`* to generate the file on the client host.

It is also possible to create the resulting file on a different host other than the server host, if the location of the file on the remote host can be accessed using a network-mapped path on the server's file system. In this case, the presence of `mysql` (or some other MySQL client program) is not required on the target host.

`SELECT ... INTO OUTFILE` is the complement of `LOAD DATA INFILE`. Column values are written converted to the character set specified in the `CHARACTER SET` clause. If no such clause is present, values are dumped using the `binary` character set. In effect, there is no character set conversion. If a result set contains columns in several character sets, the output data file will as well and you may not be able to reload the file correctly.

The syntax for the *`export_options`* part of the statement consists of the same `FIELDS` and `LINES` clauses that are used with the `LOAD DATA INFILE` statement. See Section 13.2.6, "`LOAD DATA INFILE` Syntax", for information about the `FIELDS` and `LINES` clauses, including their default values and permissible values.

FIELDS ESCAPED BY controls how to write special characters. If the FIELDS ESCAPED BY character is not empty, it is used when necessary to avoid ambiguity as a prefix that precedes following characters on output:

- The FIELDS ESCAPED BY character
- The FIELDS [OPTIONALLY] ENCLOSED BY character
- The first character of the FIELDS TERMINATED BY and LINES TERMINATED BY values
- ASCII NUL (the zero-valued byte; what is actually written following the escape character is ASCII “0”, not a zero-valued byte)

The FIELDS TERMINATED BY, ENCLOSED BY, ESCAPED BY, or LINES TERMINATED BY characters *must* be escaped so that you can read the file back in reliably. ASCII NUL is escaped to make it easier to view with some pagers.

The resulting file does not have to conform to SQL syntax, so nothing else need be escaped.

If the FIELDS ESCAPED BY character is empty, no characters are escaped and NULL is output as NULL, not \N. It is probably not a good idea to specify an empty escape character, particularly if field values in your data contain any of the characters in the list just given.

Here is an example that produces a file in the comma-separated values (CSV) format used by many programs:

```
SELECT a,b,a+b INTO OUTFILE '/tmp/result.txt'
  FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
  LINES TERMINATED BY '\n'
  FROM test_table;
```

If you use INTO DUMPFILE instead of INTO OUTFILE, MySQL writes only one row into the file, without any column or line termination and without performing any escape processing. This is useful if you want to store a BLOB value in a file.

Note

Any file created by INTO OUTFILE or INTO DUMPFILE is writable by all users on the server host. The reason for this is that the MySQL server cannot create a file that is owned by anyone other than the user under whose account it is running. (You should *never* run mysqld as root for this and other reasons.) The file thus must be world-writable so that you can manipulate its contents.

If the secure_file_priv system variable is set to a nonempty directory name, the file to be written must be located in that directory.

In the context of SELECT ... INTO statements that occur as part of events executed by the Event Scheduler, diagnostics messages (not only errors, but also warnings) are written to the error log, and, on Windows, to the application event log. For additional information, see Section 18.4.5, “Event Scheduler Status”.

13.2.9.2 JOIN Syntax

MySQL supports the following JOIN syntaxes for the *table_references* part of SELECT statements and multiple-table DELETE and UPDATE statements:

```
table_references:
    escaped_table_reference [, escaped_table_reference] ...

escaped_table_reference:
    table_reference
  | { OJ table_reference }

table_reference:
    table_factor
  | join_table

table_factor:
    tbl_name [PARTITION (partition_names)]
        [[AS] alias] [index_hint_list]
  | table_subquery [AS] alias
  | ( table_references )

join_table:
    table_reference [INNER | CROSS] JOIN table_factor [join_condition]
  | table_reference STRAIGHT_JOIN table_factor
  | table_reference STRAIGHT_JOIN table_factor ON conditional_expr
  | table_reference {LEFT|RIGHT} [OUTER] JOIN table_reference join_condition
  | table_reference NATURAL [{LEFT|RIGHT} [OUTER]] JOIN table_factor

join_condition:
    ON conditional_expr
  | USING (column_list)

index_hint_list:
    index_hint [, index_hint] ...

index_hint:
    USE {INDEX|KEY}
      [FOR {JOIN|ORDER BY|GROUP BY}] ([index_list])
  | IGNORE {INDEX|KEY}
      [FOR {JOIN|ORDER BY|GROUP BY}] (index_list)
  | FORCE {INDEX|KEY}
      [FOR {JOIN|ORDER BY|GROUP BY}] (index_list)

index_list:
    index_name [, index_name] ...
```

A table reference is also known as a join expression.

A table reference (when it refers to a partitioned table) may contain a `PARTITION` option, including a comma-separated list of partitions, subpartitions, or both. This option follows the name of the table and precedes any alias declaration. The effect of this option is that rows are selected only from the listed partitions or subpartitions—in other words, any partitions or subpartitions not named in the list are ignored For more information, see Section 17.5, “Partition Selection”.

The syntax of *`table_factor`* is extended in comparison with the SQL Standard. The latter accepts only *`table_reference`*, not a list of them inside a pair of parentheses.

This is a conservative extension if we consider each comma in a list of *`table_reference`* items as equivalent to an inner join. For example:

```
SELECT * FROM t1 LEFT JOIN (t2, t3, t4)
                 ON (t2.a=t1.a AND t3.b=t1.b AND t4.c=t1.c)
```

is equivalent to:

```
SELECT * FROM t1 LEFT JOIN (t2 CROSS JOIN t3 CROSS JOIN t4)
```

```
                 ON (t2.a=t1.a AND t3.b=t1.b AND t4.c=t1.c)
```

In MySQL, JOIN, CROSS JOIN, and INNER JOIN are syntactic equivalents (they can replace each other). In standard SQL, they are not equivalent. INNER JOIN is used with an ON clause, CROSS JOIN is used otherwise.

In general, parentheses can be ignored in join expressions containing only inner join operations. MySQL also supports nested joins (see Section 8.2.1.11, "Nested Join Optimization").

Index hints can be specified to affect how the MySQL optimizer makes use of indexes. For more information, see Section 13.2.9.3, "Index Hint Syntax".

The following list describes general factors to take into account when writing joins.

- A table reference can be aliased using *tbl_name* AS *alias_name* or *tbl_name alias_name*:

  ```
  SELECT t1.name, t2.salary
    FROM employee AS t1 INNER JOIN info AS t2 ON t1.name = t2.name;

  SELECT t1.name, t2.salary
    FROM employee t1 INNER JOIN info t2 ON t1.name = t2.name;
  ```

- A *table_subquery* is also known as a subquery in the FROM clause. Such subqueries *must* include an alias to give the subquery result a table name. A trivial example follows; see also Section 13.2.10.8, "Subqueries in the FROM Clause".

  ```
  SELECT * FROM (SELECT 1, 2, 3) AS t1;
  ```

- INNER JOIN and , (comma) are semantically equivalent in the absence of a join condition: both produce a Cartesian product between the specified tables (that is, each and every row in the first table is joined to each and every row in the second table).

  However, the precedence of the comma operator is less than of INNER JOIN, CROSS JOIN, LEFT JOIN, and so on. If you mix comma joins with the other join types when there is a join condition, an error of the form Unknown column '*col_name*' in 'on clause' may occur. Information about dealing with this problem is given later in this section.

- The *conditional_expr* used with ON is any conditional expression of the form that can be used in a WHERE clause. Generally, you should use the ON clause for conditions that specify how to join tables, and the WHERE clause to restrict which rows you want in the result set.

- If there is no matching row for the right table in the ON or USING part in a LEFT JOIN, a row with all columns set to NULL is used for the right table. You can use this fact to find rows in a table that have no counterpart in another table:

  ```
  SELECT left_tbl.*
    FROM left_tbl LEFT JOIN right_tbl ON left_tbl.id = right_tbl.id
    WHERE right_tbl.id IS NULL;
  ```

  This example finds all rows in left_tbl with an id value that is not present in right_tbl (that is, all rows in left_tbl with no corresponding row in right_tbl). This assumes that right_tbl.id is declared NOT NULL. See Section 8.2.1.9, "LEFT JOIN and RIGHT JOIN Optimization".

- The USING(*column_list*) clause names a list of columns that must exist in both tables. If tables a and b both contain columns c1, c2, and c3, the following join compares corresponding columns from the two tables:

```
a LEFT JOIN b USING (c1,c2,c3)
```

- The NATURAL [LEFT] JOIN of two tables is defined to be semantically equivalent to an INNER JOIN or a LEFT JOIN with a USING clause that names all columns that exist in both tables.

- RIGHT JOIN works analogously to LEFT JOIN. To keep code portable across databases, it is recommended that you use LEFT JOIN instead of RIGHT JOIN.

- The { OJ ... } syntax shown in the join syntax description exists only for compatibility with ODBC. The curly braces in the syntax should be written literally; they are not metasyntax as used elsewhere in syntax descriptions.

  ```
  SELECT left_tbl.*
      FROM { OJ left_tbl LEFT OUTER JOIN right_tbl ON left_tbl.id = right_tbl.id }
      WHERE right_tbl.id IS NULL;
  ```

  You can use other types of joins within { OJ ... }, such as INNER JOIN or RIGHT OUTER JOIN. This helps with compatibility with some third-party applications, but is not official ODBC syntax.

- STRAIGHT_JOIN is similar to JOIN, except that the left table is always read before the right table. This can be used for those (few) cases for which the join optimizer puts the tables in the wrong order.

Some join examples:

```
SELECT * FROM table1, table2;

SELECT * FROM table1 INNER JOIN table2 ON table1.id=table2.id;

SELECT * FROM table1 LEFT JOIN table2 ON table1.id=table2.id;

SELECT * FROM table1 LEFT JOIN table2 USING (id);

SELECT * FROM table1 LEFT JOIN table2 ON table1.id=table2.id
  LEFT JOIN table3 ON table2.id=table3.id;
```

Join Processing Changes in MySQL 5.0.12

Note

Natural joins and joins with USING, including outer join variants, are processed according to the SQL:2003 standard. The goal was to align the syntax and semantics of MySQL with respect to NATURAL JOIN and JOIN ... USING according to SQL:2003. However, these changes in join processing can result in different output columns for some joins. Also, some queries that appeared to work correctly in older versions (prior to 5.0.12) must be rewritten to comply with the standard.

These changes have five main aspects:

- The way that MySQL determines the result columns of NATURAL or USING join operations (and thus the result of the entire FROM clause).

- Expansion of SELECT * and SELECT *tbl_name*.* into a list of selected columns.

- Resolution of column names in NATURAL or USING joins.

- Transformation of NATURAL or USING joins into JOIN ... ON.

- Resolution of column names in the ON condition of a JOIN ... ON.

The following list provides more detail about several effects of current join processing versus join processing in older versions. The term “previously” means “prior to MySQL 5.0.12.”

- The columns of a NATURAL join or a USING join may be different from previously. Specifically, redundant output columns no longer appear, and the order of columns for SELECT * expansion may be different from before.

  Consider this set of statements:

  ```
  CREATE TABLE t1 (i INT, j INT);
  CREATE TABLE t2 (k INT, j INT);
  INSERT INTO t1 VALUES(1,1);
  INSERT INTO t2 VALUES(1,1);
  SELECT * FROM t1 NATURAL JOIN t2;
  SELECT * FROM t1 JOIN t2 USING (j);
  ```

  Previously, the statements produced this output:

  ```
  +------+------+------+------+
  | i    | j    | k    | j    |
  +------+------+------+------+
  |    1 |    1 |    1 |    1 |
  +------+------+------+------+
  +------+------+------+------+
  | i    | j    | k    | j    |
  +------+------+------+------+
  |    1 |    1 |    1 |    1 |
  +------+------+------+------+
  ```

  In the first SELECT statement, column j appears in both tables and thus becomes a join column, so, according to standard SQL, it should appear only once in the output, not twice. Similarly, in the second SELECT statement, column j is named in the USING clause and should appear only once in the output, not twice. But in both cases, the redundant column is not eliminated. Also, the order of the columns is not correct according to standard SQL.

  Now the statements produce this output:

  ```
  +------+------+------+
  | j    | i    | k    |
  +------+------+------+
  |    1 |    1 |    1 |
  +------+------+------+
  +------+------+------+
  | j    | i    | k    |
  +------+------+------+
  |    1 |    1 |    1 |
  +------+------+------+
  ```

  The redundant column is eliminated and the column order is correct according to standard SQL:

  - First, coalesced common columns of the two joined tables, in the order in which they occur in the first table
  - Second, columns unique to the first table, in order in which they occur in that table
  - Third, columns unique to the second table, in order in which they occur in that table

The single result column that replaces two common columns is defined using the coalesce operation. That is, for two t1.a and t2.a the resulting single join column a is defined as a = COALESCE(t1.a, t2.a), where:

```
COALESCE(x, y) = (CASE WHEN V1 IS NOT NULL THEN V1 ELSE V2 END)
```

If the join operation is any other join, the result columns of the join consists of the concatenation of all columns of the joined tables. This is the same as previously.

A consequence of the definition of coalesced columns is that, for outer joins, the coalesced column contains the value of the non-NULL column if one of the two columns is always NULL. If neither or both columns are NULL, both common columns have the same value, so it doesn't matter which one is chosen as the value of the coalesced column. A simple way to interpret this is to consider that a coalesced column of an outer join is represented by the common column of the inner table of a JOIN. Suppose that the tables t1(a,b) and t2(a,c) have the following contents:

```
t1    t2
----  ----
1 x   2 z
2 y   3 w
```

Then:

```
mysql> SELECT * FROM t1 NATURAL LEFT JOIN t2;
+------+------+------+
| a    | b    | c    |
+------+------+------+
|    1 | x    | NULL |
|    2 | y    | z    |
+------+------+------+
```

Here column a contains the values of t1.a.

```
mysql> SELECT * FROM t1 NATURAL RIGHT JOIN t2;
+------+------+------+
| a    | c    | b    |
+------+------+------+
|    2 | z    | y    |
|    3 | w    | NULL |
+------+------+------+
```

Here column a contains the values of t2.a.

Compare these results to the otherwise equivalent queries with JOIN ... ON:

```
mysql> SELECT * FROM t1 LEFT JOIN t2 ON (t1.a = t2.a);
+------+------+------+------+
| a    | b    | a    | c    |
+------+------+------+------+
|    1 | x    | NULL | NULL |
|    2 | y    |    2 | z    |
+------+------+------+------+
```

```
mysql> SELECT * FROM t1 RIGHT JOIN t2 ON (t1.a = t2.a);
+------+------+------+------+
| a    | b    | a    | c    |
+------+------+------+------+
|    2 | y    |    2 | z    |
| NULL | NULL |    3 | w    |
+------+------+------+------+
```

- Previously, a `USING` clause could be rewritten as an `ON` clause that compares corresponding columns. For example, the following two clauses were semantically identical:

  ```
  a LEFT JOIN b USING (c1,c2,c3)
  a LEFT JOIN b ON a.c1=b.c1 AND a.c2=b.c2 AND a.c3=b.c3
  ```

  Now the two clauses no longer are quite the same:

  - With respect to determining which rows satisfy the join condition, both joins remain semantically identical.

  - With respect to determining which columns to display for `SELECT *` expansion, the two joins are not semantically identical. The `USING` join selects the coalesced value of corresponding columns, whereas the `ON` join selects all columns from all tables. For the preceding `USING` join, `SELECT *` selects these values:

    ```
    COALESCE(a.c1,b.c1), COALESCE(a.c2,b.c2), COALESCE(a.c3,b.c3)
    ```

    For the `ON` join, `SELECT *` selects these values:

    ```
    a.c1, a.c2, a.c3, b.c1, b.c2, b.c3
    ```

    With an inner join, `COALESCE(a.c1,b.c1)` is the same as either `a.c1` or `b.c1` because both columns will have the same value. With an outer join (such as `LEFT JOIN`), one of the two columns can be `NULL`. That column will be omitted from the result.

- The evaluation of multi-way natural joins differs in a very important way that affects the result of `NATURAL` or `USING` joins and that can require query rewriting. Suppose that you have three tables `t1(a,b)`, `t2(c,b)`, and `t3(a,c)` that each have one row: `t1(1,2)`, `t2(10,2)`, and `t3(7,10)`. Suppose also that you have this `NATURAL JOIN` on the three tables:

  ```
  SELECT ... FROM t1 NATURAL JOIN t2 NATURAL JOIN t3;
  ```

  Previously, the left operand of the second join was considered to be `t2`, whereas it should be the nested join `(t1 NATURAL JOIN t2)`. As a result, the columns of `t3` are checked for common columns only in `t2`, and, if `t3` has common columns with `t1`, these columns are not used as equi-join columns. Thus, previously, the preceding query was transformed to the following equi-join:

  ```
  SELECT ... FROM t1, t2, t3
    WHERE t1.b = t2.b AND t2.c = t3.c;
  ```

  That join is missing one more equi-join predicate `(t1.a = t3.a)`. As a result, it produces one row, not the empty result that it should. The correct equivalent query is this:

  ```
  SELECT ... FROM t1, t2, t3
    WHERE t1.b = t2.b AND t2.c = t3.c AND t1.a = t3.a;
  ```

  If you require the same query result in current versions of MySQL as in older versions, rewrite the natural join as the first equi-join.

- Previously, the comma operator (,) and JOIN both had the same precedence, so the join expression t1, t2 JOIN t3 was interpreted as ((t1, t2) JOIN t3). Now JOIN has higher precedence, so the expression is interpreted as (t1, (t2 JOIN t3)). This change affects statements that use an ON clause, because that clause can refer only to columns in the operands of the join, and the change in precedence changes interpretation of what those operands are.

  Example:

  ```
  CREATE TABLE t1 (i1 INT, j1 INT);
  CREATE TABLE t2 (i2 INT, j2 INT);
  CREATE TABLE t3 (i3 INT, j3 INT);
  INSERT INTO t1 VALUES(1,1);
  INSERT INTO t2 VALUES(1,1);
  INSERT INTO t3 VALUES(1,1);
  SELECT * FROM t1, t2 JOIN t3 ON (t1.i1 = t3.i3);
  ```

  Previously, the SELECT was legal due to the implicit grouping of t1,t2 as (t1,t2). Now the JOIN takes precedence, so the operands for the ON clause are t2 and t3. Because t1.i1 is not a column in either of the operands, the result is an Unknown column 't1.i1' in 'on clause' error. To allow the join to be processed, group the first two tables explicitly with parentheses so that the operands for the ON clause are (t1,t2) and t3:

  ```
  SELECT * FROM (t1, t2) JOIN t3 ON (t1.i1 = t3.i3);
  ```

  Alternatively, avoid the use of the comma operator and use JOIN instead:

  ```
  SELECT * FROM t1 JOIN t2 JOIN t3 ON (t1.i1 = t3.i3);
  ```

  This change also applies to statements that mix the comma operator with INNER JOIN, CROSS JOIN, LEFT JOIN, and RIGHT JOIN, all of which now have higher precedence than the comma operator.

- Previously, the ON clause could refer to columns in tables named to its right. Now an ON clause can refer only to its operands.

  Example:

  ```
  CREATE TABLE t1 (i1 INT);
  CREATE TABLE t2 (i2 INT);
  CREATE TABLE t3 (i3 INT);
  SELECT * FROM t1 JOIN t2 ON (i1 = i3) JOIN t3;
  ```

  Previously, the SELECT statement was legal. Now the statement fails with an Unknown column 'i3' in 'on clause' error because i3 is a column in t3, which is not an operand of the ON clause. The statement should be rewritten as follows:

  ```
  SELECT * FROM t1 JOIN t2 JOIN t3 ON (i1 = i3);
  ```

- Resolution of column names in NATURAL or USING joins is different than previously. For column names that are outside the FROM clause, MySQL now handles a superset of the queries compared to previously. That is, in cases when MySQL formerly issued an error that some column is ambiguous, the query now is handled correctly. This is due to the fact that MySQL now treats the common columns of NATURAL or USING joins as a single column, so when a query refers to such columns, the query compiler does not consider them as ambiguous.

  Example:

```
SELECT * FROM t1 NATURAL JOIN t2 WHERE b > 1;
```

Previously, this query would produce an error `ERROR 1052 (23000): Column 'b' in where clause is ambiguous`. Now the query produces the correct result:

```
+------+------+------+
| b    | c    | y    |
+------+------+------+
|    4 |    2 |    3 |
+------+------+------+
```

One extension of MySQL compared to the SQL:2003 standard is that MySQL enables you to qualify the common (coalesced) columns of `NATURAL` or `USING` joins (just as previously), while the standard disallows that.

13.2.9.3 Index Hint Syntax

You can provide hints to give the optimizer information about how to choose indexes during query processing. Section 13.2.9.2, “JOIN Syntax”, describes the general syntax for specifying tables in a `SELECT` statement. The syntax for an individual table, including that for index hints, looks like this:

```
tbl_name [[AS] alias] [index_hint_list]

index_hint_list:
    index_hint [, index_hint] ...

index_hint:
    USE {INDEX|KEY}
      [FOR {JOIN|ORDER BY|GROUP BY}] ([index_list])
  | IGNORE {INDEX|KEY}
      [FOR {JOIN|ORDER BY|GROUP BY}] (index_list)
  | FORCE {INDEX|KEY}
      [FOR {JOIN|ORDER BY|GROUP BY}] (index_list)

index_list:
    index_name [, index_name] ...
```

By specifying `USE INDEX (index_list)`, you can tell MySQL to use only one of the named indexes to find rows in the table. The alternative syntax `IGNORE INDEX (index_list)` can be used to tell MySQL to not use some particular index or indexes. These hints are useful if `EXPLAIN` shows that MySQL is using the wrong index from the list of possible indexes.

You can also use `FORCE INDEX`, which acts like `USE INDEX (index_list)` but with the addition that a table scan is assumed to be *very* expensive. In other words, a table scan is used only if there is no way to use one of the given indexes to find rows in the table.

Each hint requires the names of *indexes*, not the names of columns. The name of a `PRIMARY KEY` is `PRIMARY`. To see the index names for a table, use `SHOW INDEX`.

An *`index_name`* value need not be a full index name. It can be an unambiguous prefix of an index name. If a prefix is ambiguous, an error occurs.

Examples:

```
SELECT * FROM table1 USE INDEX (col1_index,col2_index)
  WHERE col1=1 AND col2=2 AND col3=3;
```

```
SELECT * FROM table1 IGNORE INDEX (col3_index)
  WHERE col1=1 AND col2=2 AND col3=3;
```

The syntax for index hints has the following characteristics:

- It is syntactically valid to specify an empty *index_list* for USE INDEX, which means “use no indexes.” Specifying an empty *index_list* for FORCE INDEX or IGNORE INDEX is a syntax error.

- You can specify the scope of a index hint by adding a FOR clause to the hint. This provides more fine-grained control over the optimizer's selection of an execution plan for various phases of query processing. To affect only the indexes used when MySQL decides how to find rows in the table and how to process joins, use FOR JOIN. To influence index usage for sorting or grouping rows, use FOR ORDER BY or FOR GROUP BY. (However, if there is a covering index for the table and it is used to access the table, the optimizer will ignore IGNORE INDEX FOR {ORDER BY|GROUP BY} hints that disable that index.)

- You can specify multiple index hints:

  ```
  SELECT * FROM t1 USE INDEX (i1) IGNORE INDEX FOR ORDER BY (i2) ORDER BY a;
  ```

  It is not a error to name the same index in several hints (even within the same hint):

  ```
  SELECT * FROM t1 USE INDEX (i1) USE INDEX (i1,i1);
  ```

  However, it is an error to mix USE INDEX and FORCE INDEX for the same table:

  ```
  SELECT * FROM t1 USE INDEX FOR JOIN (i1) FORCE INDEX FOR JOIN (i2);
  ```

if you specify no FOR clause for an index hint, the hint by default applies to all parts of the statement. For example, this hint:

```
IGNORE INDEX (i1)
```

is equivalent to this combination of hints:

```
IGNORE INDEX FOR JOIN (i1)
IGNORE INDEX FOR ORDER BY (i1)
IGNORE INDEX FOR GROUP BY (i1)
```

To cause the server to use the older behavior for hint scope when no FOR clause is present (so that hints apply only to row retrieval), enable the old system variable at server startup. Take care about enabling this variable in a replication setup. With statement-based binary logging, having different modes for the master and slaves might lead to replication errors.

When index hints are processed, they are collected in a single list by type (USE, FORCE, IGNORE) and by scope (FOR JOIN, FOR ORDER BY, FOR GROUP BY). For example:

```
SELECT * FROM t1
  USE INDEX () IGNORE INDEX (i2) USE INDEX (i1) USE INDEX (i2);
```

is equivalent to:

```
SELECT * FROM t1
```

```
  USE INDEX (i1,i2) IGNORE INDEX (i2);
```

The index hints then are applied for each scope in the following order:

1. {USE|FORCE} INDEX is applied if present. (If not, the optimizer-determined set of indexes is used.)

2. IGNORE INDEX is applied over the result of the previous step. For example, the following two queries are equivalent:

   ```
   SELECT * FROM t1 USE INDEX (i1) IGNORE INDEX (i2) USE INDEX (i2);

   SELECT * FROM t1 USE INDEX (i1);
   ```

For FULLTEXT searches, index hints work as follows:

- For natural language mode searches, index hints are silently ignored. For example, IGNORE INDEX(i) is ignored with no warning and the index is still used.

  For boolean mode searches, index hints with FOR ORDER BY or FOR GROUP BY are silently ignored. Index hints with FOR JOIN or no FOR modifier are honored. In contrast to how hints apply for non-FULLTEXT searches, the hint is used for all phases of query execution (finding rows and retrieval, grouping, and ordering). This is true even if the hint is given for a non-FULLTEXT index.

For example, the following two queries are equivalent:

```
SELECT * FROM t
  USE INDEX (index1)
  IGNORE INDEX (index1) FOR ORDER BY
  IGNORE INDEX (index1) FOR GROUP BY
  WHERE ... IN BOOLEAN MODE ... ;

SELECT * FROM t
  USE INDEX (index1)
  WHERE ... IN BOOLEAN MODE ... ;
```

13.2.9.4 UNION Syntax

```
SELECT ...
UNION [ALL | DISTINCT] SELECT ...
[UNION [ALL | DISTINCT] SELECT ...]
```

UNION is used to combine the result from multiple SELECT statements into a single result set.

The column names from the first SELECT statement are used as the column names for the results returned. Selected columns listed in corresponding positions of each SELECT statement should have the same data type. (For example, the first column selected by the first statement should have the same type as the first column selected by the other statements.)

If the data types of corresponding SELECT columns do not match, the types and lengths of the columns in the UNION result take into account the values retrieved by all of the SELECT statements. For example, consider the following:

```
mysql> SELECT REPEAT('a',1) UNION SELECT REPEAT('b',10);
+---------------+
| REPEAT('a',1) |
+---------------+
| a             |
```

```
| bbbbbbbbbb    |
+---------------+
```

The SELECT statements are normal select statements, but with the following restrictions:

- Only the last SELECT statement can use INTO OUTFILE. (However, the entire UNION result is written to the file.)

- HIGH_PRIORITY cannot be used with SELECT statements that are part of a UNION. If you specify it for the first SELECT, it has no effect. If you specify it for any subsequent SELECT statements, a syntax error results.

The default behavior for UNION is that duplicate rows are removed from the result. The optional DISTINCT keyword has no effect other than the default because it also specifies duplicate-row removal. With the optional ALL keyword, duplicate-row removal does not occur and the result includes all matching rows from all the SELECT statements.

You can mix UNION ALL and UNION DISTINCT in the same query. Mixed UNION types are treated such that a DISTINCT union overrides any ALL union to its left. A DISTINCT union can be produced explicitly by using UNION DISTINCT or implicitly by using UNION with no following DISTINCT or ALL keyword.

To apply ORDER BY or LIMIT to an individual SELECT, place the clause inside the parentheses that enclose the SELECT:

```
(SELECT a FROM t1 WHERE a=10 AND B=1 ORDER BY a LIMIT 10)
UNION
(SELECT a FROM t2 WHERE a=11 AND B=2 ORDER BY a LIMIT 10);
```

However, use of ORDER BY for individual SELECT statements implies nothing about the order in which the rows appear in the final result because UNION by default produces an unordered set of rows. Therefore, the use of ORDER BY in this context is typically in conjunction with LIMIT, so that it is used to determine the subset of the selected rows to retrieve for the SELECT, even though it does not necessarily affect the order of those rows in the final UNION result. If ORDER BY appears without LIMIT in a SELECT, it is optimized away because it will have no effect anyway.

To use an ORDER BY or LIMIT clause to sort or limit the entire UNION result, parenthesize the individual SELECT statements and place the ORDER BY or LIMIT after the last one. The following example uses both clauses:

```
(SELECT a FROM t1 WHERE a=10 AND B=1)
UNION
(SELECT a FROM t2 WHERE a=11 AND B=2)
ORDER BY a LIMIT 10;
```

A statement without parentheses is equivalent to one parenthesized as just shown.

This kind of ORDER BY cannot use column references that include a table name (that is, names in *tbl_name*.*col_name* format). Instead, provide a column alias in the first SELECT statement and refer to the alias in the ORDER BY. (Alternatively, refer to the column in the ORDER BY using its column position. However, use of column positions is deprecated.)

Also, if a column to be sorted is aliased, the ORDER BY clause *must* refer to the alias, not the column name. The first of the following statements will work, but the second will fail with an Unknown column 'a' in 'order clause' error:

```
(SELECT a AS b FROM t) UNION (SELECT ...) ORDER BY b;
(SELECT a AS b FROM t) UNION (SELECT ...) ORDER BY a;
```

To cause rows in a `UNION` result to consist of the sets of rows retrieved by each `SELECT` one after the other, select an additional column in each `SELECT` to use as a sort column and add an `ORDER BY` following the last `SELECT`:

```
(SELECT 1 AS sort_col, col1a, col1b, ... FROM t1)
UNION
(SELECT 2, col2a, col2b, ... FROM t2) ORDER BY sort_col;
```

To additionally maintain sort order within individual `SELECT` results, add a secondary column to the `ORDER BY` clause:

```
(SELECT 1 AS sort_col, col1a, col1b, ... FROM t1)
UNION
(SELECT 2, col2a, col2b, ... FROM t2) ORDER BY sort_col, col1a;
```

Use of an additional column also enables you to determine which `SELECT` each row comes from. Extra columns can provide other identifying information as well, such as a string that indicates a table name.

13.2.10 Subquery Syntax

A subquery is a `SELECT` statement within another statement.

Starting with MySQL 4.1, all subquery forms and operations that the SQL standard requires are supported, as well as a few features that are MySQL-specific.

Here is an example of a subquery:

```
SELECT * FROM t1 WHERE column1 = (SELECT column1 FROM t2);
```

In this example, `SELECT * FROM t1 ...` is the *outer query* (or *outer statement*), and `(SELECT column1 FROM t2)` is the *subquery*. We say that the subquery is *nested* within the outer query, and in fact it is possible to nest subqueries within other subqueries, to a considerable depth. A subquery must always appear within parentheses.

The main advantages of subqueries are:

- They allow queries that are *structured* so that it is possible to isolate each part of a statement.
- They provide alternative ways to perform operations that would otherwise require complex joins and unions.
- Many people find subqueries more readable than complex joins or unions. Indeed, it was the innovation of subqueries that gave people the original idea of calling the early SQL “Structured Query Language.”

Here is an example statement that shows the major points about subquery syntax as specified by the SQL standard and supported in MySQL:

```
DELETE FROM t1
WHERE s11 > ANY
 (SELECT COUNT(*) /* no hint */ FROM t2
  WHERE NOT EXISTS
   (SELECT * FROM t3
    WHERE ROW(5*t2.s1,77)=
     (SELECT 50,11*s1 FROM t4 UNION SELECT 50,77 FROM
      (SELECT * FROM t5) AS t5)));
```

A subquery can return a scalar (a single value), a single row, a single column, or a table (one or more rows of one or more columns). These are called scalar, column, row, and table subqueries. Subqueries that return a particular kind of result often can be used only in certain contexts, as described in the following sections.

There are few restrictions on the type of statements in which subqueries can be used. A subquery can contain many of the keywords or clauses that an ordinary `SELECT` can contain: `DISTINCT`, `GROUP BY`, `ORDER BY`, `LIMIT`, joins, index hints, `UNION` constructs, comments, functions, and so on.

A subquery's outer statement can be any one of: `SELECT`, `INSERT`, `UPDATE`, `DELETE`, `SET`, or `DO`.

In MySQL, you cannot modify a table and select from the same table in a subquery. This applies to statements such as `DELETE`, `INSERT`, `REPLACE`, `UPDATE`, and (because subqueries can be used in the `SET` clause) `LOAD DATA INFILE`.

For information about how the optimizer handles subqueries, see Section 8.2.1.18, “Subquery Optimization”. For a discussion of restrictions on subquery use, including performance issues for certain forms of subquery syntax, see Section E.4, “Restrictions on Subqueries”.

13.2.10.1 The Subquery as Scalar Operand

In its simplest form, a subquery is a scalar subquery that returns a single value. A scalar subquery is a simple operand, and you can use it almost anywhere a single column value or literal is legal, and you can expect it to have those characteristics that all operands have: a data type, a length, an indication that it can be `NULL`, and so on. For example:

```
CREATE TABLE t1 (s1 INT, s2 CHAR(5) NOT NULL);
INSERT INTO t1 VALUES(100, 'abcde');
SELECT (SELECT s2 FROM t1);
```

The subquery in this `SELECT` returns a single value (`'abcde'`) that has a data type of `CHAR`, a length of 5, a character set and collation equal to the defaults in effect at `CREATE TABLE` time, and an indication that the value in the column can be `NULL`. Nullability of the value selected by a scalar subquery is not copied because if the subquery result is empty, the result is `NULL`. For the subquery just shown, if `t1` were empty, the result would be `NULL` even though `s2` is `NOT NULL`.

There are a few contexts in which a scalar subquery cannot be used. If a statement permits only a literal value, you cannot use a subquery. For example, `LIMIT` requires literal integer arguments, and `LOAD DATA INFILE` requires a literal string file name. You cannot use subqueries to supply these values.

When you see examples in the following sections that contain the rather spartan construct `(SELECT column1 FROM t1)`, imagine that your own code contains much more diverse and complex constructions.

Suppose that we make two tables:

```
CREATE TABLE t1 (s1 INT);
INSERT INTO t1 VALUES (1);
CREATE TABLE t2 (s1 INT);
INSERT INTO t2 VALUES (2);
```

Then perform a `SELECT`:

```
SELECT (SELECT s1 FROM t2) FROM t1;
```

The result is 2 because there is a row in t2 containing a column s1 that has a value of 2.

A scalar subquery can be part of an expression, but remember the parentheses, even if the subquery is an operand that provides an argument for a function. For example:

```
SELECT UPPER((SELECT s1 FROM t1)) FROM t2;
```

13.2.10.2 Comparisons Using Subqueries

The most common use of a subquery is in the form:

```
non_subquery_operand comparison_operator (subquery)
```

Where *comparison_operator* is one of these operators:

```
=  >  <  >=  <=  <>  !=  <=>
```

For example:

```
... WHERE 'a' = (SELECT column1 FROM t1)
```

MySQL also permits this construct:

```
non_subquery_operand LIKE (subquery)
```

At one time the only legal place for a subquery was on the right side of a comparison, and you might still find some old DBMSs that insist on this.

Here is an example of a common-form subquery comparison that you cannot do with a join. It finds all the rows in table t1 for which the column1 value is equal to a maximum value in table t2:

```
SELECT * FROM t1
  WHERE column1 = (SELECT MAX(column2) FROM t2);
```

Here is another example, which again is impossible with a join because it involves aggregating for one of the tables. It finds all rows in table t1 containing a value that occurs twice in a given column:

```
SELECT * FROM t1 AS t
  WHERE 2 = (SELECT COUNT(*) FROM t1 WHERE t1.id = t.id);
```

For a comparison of the subquery to a scalar, the subquery must return a scalar. For a comparison of the subquery to a row constructor, the subquery must be a row subquery that returns a row with the same number of values as the row constructor. See Section 13.2.10.5, “Row Subqueries”.

13.2.10.3 Subqueries with ANY, IN, or SOME

Syntax:

```
operand comparison_operator ANY (subquery)
operand IN (subquery)
operand comparison_operator SOME (subquery)
```

Where *comparison_operator* is one of these operators:

```
=  >  <  >=  <=  <>  !=
```

The ANY keyword, which must follow a comparison operator, means “return TRUE if the comparison is TRUE for ANY of the values in the column that the subquery returns.” For example:

```
SELECT s1 FROM t1 WHERE s1 > ANY (SELECT s1 FROM t2);
```

Suppose that there is a row in table t1 containing (10). The expression is TRUE if table t2 contains (21,14,7) because there is a value 7 in t2 that is less than 10. The expression is FALSE if table t2 contains (20,10), or if table t2 is empty. The expression is *unknown* (that is, NULL) if table t2 contains (NULL,NULL,NULL).

When used with a subquery, the word IN is an alias for = ANY. Thus, these two statements are the same:

```
SELECT s1 FROM t1 WHERE s1 = ANY (SELECT s1 FROM t2);
SELECT s1 FROM t1 WHERE s1 IN    (SELECT s1 FROM t2);
```

IN and = ANY are not synonyms when used with an expression list. IN can take an expression list, but = ANY cannot. See Section 12.3.2, “Comparison Functions and Operators”.

NOT IN is not an alias for <> ANY, but for <> ALL. See Section 13.2.10.4, “Subqueries with ALL”.

The word SOME is an alias for ANY. Thus, these two statements are the same:

```
SELECT s1 FROM t1 WHERE s1 <> ANY  (SELECT s1 FROM t2);
SELECT s1 FROM t1 WHERE s1 <> SOME (SELECT s1 FROM t2);
```

Use of the word SOME is rare, but this example shows why it might be useful. To most people, the English phrase “a is not equal to any b” means “there is no b which is equal to a,” but that is not what is meant by the SQL syntax. The syntax means “there is some b to which a is not equal.” Using <> SOME instead helps ensure that everyone understands the true meaning of the query.

13.2.10.4 Subqueries with ALL

Syntax:

```
operand comparison_operator ALL (subquery)
```

The word ALL, which must follow a comparison operator, means “return TRUE if the comparison is TRUE for ALL of the values in the column that the subquery returns.” For example:

```
SELECT s1 FROM t1 WHERE s1 > ALL (SELECT s1 FROM t2);
```

Suppose that there is a row in table t1 containing (10). The expression is TRUE if table t2 contains (-5,0,+5) because 10 is greater than all three values in t2. The expression is FALSE if table t2 contains (12,6,NULL,-100) because there is a single value 12 in table t2 that is greater than 10. The expression is *unknown* (that is, NULL) if table t2 contains (0,NULL,1).

Finally, the expression is TRUE if table t2 is empty. So, the following expression is TRUE when table t2 is empty:

```
SELECT * FROM t1 WHERE 1 > ALL (SELECT s1 FROM t2);
```

But this expression is NULL when table t2 is empty:

```
SELECT * FROM t1 WHERE 1 > (SELECT s1 FROM t2);
```

In addition, the following expression is NULL when table t2 is empty:

```
SELECT * FROM t1 WHERE 1 > ALL (SELECT MAX(s1) FROM t2);
```

In general, *tables containing NULL values* and *empty tables* are “edge cases.” When writing subqueries, always consider whether you have taken those two possibilities into account.

NOT IN is an alias for <> ALL. Thus, these two statements are the same:

```
SELECT s1 FROM t1 WHERE s1 <> ALL (SELECT s1 FROM t2);
SELECT s1 FROM t1 WHERE s1 NOT IN (SELECT s1 FROM t2);
```

13.2.10.5 Row Subqueries

The discussion to this point has been of scalar or column subqueries; that is, subqueries that return a single value or a column of values. A *row subquery* is a subquery variant that returns a single row and can thus return more than one column value. Legal operators for row subquery comparisons are:

```
=  >  <  >=  <=  <>  !=  <=>
```

Here are two examples:

```
SELECT * FROM t1
  WHERE (col1,col2) = (SELECT col3, col4 FROM t2 WHERE id = 10);
SELECT * FROM t1
  WHERE ROW(col1,col2) = (SELECT col3, col4 FROM t2 WHERE id = 10);
```

For both queries, if the table t2 contains a single row with id = 10, the subquery returns a single row. If this row has col3 and col4 values equal to the col1 and col2 values of any rows in t1, the WHERE expression is TRUE and each query returns those t1 rows. If the t2 row col3 and col4 values are not equal the col1 and col2 values of any t1 row, the expression is FALSE and the query returns an empty result set. The expression is *unknown* (that is, NULL) if the subquery produces no rows. An error occurs if the subquery produces multiple rows because a row subquery can return at most one row.

The expressions (1,2) and ROW(1,2) are sometimes called *row constructors*. The two are equivalent. The row constructor and the row returned by the subquery must contain the same number of values.

A row constructor is used for comparisons with subqueries that return two or more columns. When a subquery returns a single column, this is regarded as a scalar value and not as a row, so a row constructor cannot be used with a subquery that does not return at least two columns. Thus, the following query fails with a syntax error:

```
SELECT * FROM t1 WHERE ROW(1) = (SELECT column1 FROM t2)
```

Row constructors are legal in other contexts. For example, the following two statements are semantically equivalent (and are handled in the same way by the optimizer):

```
SELECT * FROM t1 WHERE (column1,column2) = (1,1);
SELECT * FROM t1 WHERE column1 = 1 AND column2 = 1;
```

The following query answers the request, “find all rows in table `t1` that also exist in table `t2`”:

```
SELECT column1,column2,column3
  FROM t1
  WHERE (column1,column2,column3) IN
         (SELECT column1,column2,column3 FROM t2);
```

13.2.10.6 Subqueries with `EXISTS` or `NOT EXISTS`

If a subquery returns any rows at all, `EXISTS` *`subquery`* is `TRUE`, and `NOT EXISTS` *`subquery`* is `FALSE`. For example:

```
SELECT column1 FROM t1 WHERE EXISTS (SELECT * FROM t2);
```

Traditionally, an `EXISTS` subquery starts with `SELECT *`, but it could begin with `SELECT 5` or `SELECT column1` or anything at all. MySQL ignores the `SELECT` list in such a subquery, so it makes no difference.

For the preceding example, if `t2` contains any rows, even rows with nothing but `NULL` values, the `EXISTS` condition is `TRUE`. This is actually an unlikely example because a `[NOT] EXISTS` subquery almost always contains correlations. Here are some more realistic examples:

- What kind of store is present in one or more cities?

  ```
  SELECT DISTINCT store_type FROM stores
    WHERE EXISTS (SELECT * FROM cities_stores
                  WHERE cities_stores.store_type = stores.store_type);
  ```

- What kind of store is present in no cities?

  ```
  SELECT DISTINCT store_type FROM stores
    WHERE NOT EXISTS (SELECT * FROM cities_stores
                      WHERE cities_stores.store_type = stores.store_type);
  ```

- What kind of store is present in all cities?

  ```
  SELECT DISTINCT store_type FROM stores s1
    WHERE NOT EXISTS (
      SELECT * FROM cities WHERE NOT EXISTS (
        SELECT * FROM cities_stores
         WHERE cities_stores.city = cities.city
         AND cities_stores.store_type = stores.store_type));
  ```

The last example is a double-nested `NOT EXISTS` query. That is, it has a `NOT EXISTS` clause within a `NOT EXISTS` clause. Formally, it answers the question “does a city exist with a store that is not in `Stores`”? But it is easier to say that a nested `NOT EXISTS` answers the question “is *`x`* `TRUE` for all *`y`*?”

13.2.10.7 Correlated Subqueries

A *correlated subquery* is a subquery that contains a reference to a table that also appears in the outer query. For example:

```
SELECT * FROM t1
  WHERE column1 = ANY (SELECT column1 FROM t2
                       WHERE t2.column2 = t1.column2);
```

Notice that the subquery contains a reference to a column of `t1`, even though the subquery's `FROM` clause does not mention a table `t1`. So, MySQL looks outside the subquery, and finds `t1` in the outer query.

Suppose that table `t1` contains a row where `column1 = 5` and `column2 = 6`; meanwhile, table `t2` contains a row where `column1 = 5` and `column2 = 7`. The simple expression `... WHERE column1 = ANY (SELECT column1 FROM t2)` would be `TRUE`, but in this example, the `WHERE` clause within the subquery is `FALSE` (because `(5,6)` is not equal to `(5,7)`), so the expression as a whole is `FALSE`.

Scoping rule: MySQL evaluates from inside to outside. For example:

```
SELECT column1 FROM t1 AS x
  WHERE x.column1 = (SELECT column1 FROM t2 AS x
    WHERE x.column1 = (SELECT column1 FROM t3
      WHERE x.column2 = t3.column1));
```

In this statement, `x.column2` must be a column in table `t2` because `SELECT column1 FROM t2 AS x ...` renames `t2`. It is not a column in table `t1` because `SELECT column1 FROM t1 ...` is an outer query that is *farther out*.

For subqueries in `HAVING` or `ORDER BY` clauses, MySQL also looks for column names in the outer select list.

For certain cases, a correlated subquery is optimized. For example:

```
val IN (SELECT key_val FROM tbl_name WHERE correlated_condition)
```

Otherwise, they are inefficient and likely to be slow. Rewriting the query as a join might improve performance.

Aggregate functions in correlated subqueries may contain outer references, provided the function contains nothing but outer references, and provided the function is not contained in another function or expression.

13.2.10.8 Subqueries in the `FROM` Clause

Subqueries are legal in a `SELECT` statement's `FROM` clause. The actual syntax is:

```
SELECT ... FROM (subquery) [AS] name ...
```

The `[AS] name` clause is mandatory, because every table in a `FROM` clause must have a name. Any columns in the *`subquery`* select list must have unique names.

For the sake of illustration, assume that you have this table:

```
CREATE TABLE t1 (s1 INT, s2 CHAR(5), s3 FLOAT);
```

Here is how to use a subquery in the `FROM` clause, using the example table:

```
INSERT INTO t1 VALUES (1,'1',1.0);
INSERT INTO t1 VALUES (2,'2',2.0);
SELECT sb1,sb2,sb3
  FROM (SELECT s1 AS sb1, s2 AS sb2, s3*2 AS sb3 FROM t1) AS sb
  WHERE sb1 > 1;
```

Result: `2, '2', 4.0`.

Here is another example: Suppose that you want to know the average of a set of sums for a grouped table. This does not work:

```
SELECT AVG(SUM(column1)) FROM t1 GROUP BY column1;
```

However, this query provides the desired information:

```
SELECT AVG(sum_column1)
  FROM (SELECT SUM(column1) AS sum_column1
        FROM t1 GROUP BY column1) AS t1;
```

Notice that the column name used within the subquery (`sum_column1`) is recognized in the outer query.

Subqueries in the `FROM` clause can return a scalar, column, row, or table. Subqueries in the `FROM` clause cannot be correlated subqueries, unless used within the `ON` clause of a `JOIN` operation.

In MySQL 5.7, the optimizer determines information about derived tables in such a way that materialization of them does not occur for `EXPLAIN`. See Optimizing Subqueries in the `FROM` Clause (Derived Tables).

It is possible under certain circumstances to modify table data using `EXPLAIN SELECT`. This can occur if the outer query accesses any tables and an inner query invokes a stored function that changes one or more rows of a table. Suppose that there are two tables `t1` and `t2` in database `d1`, created as shown here:

```
mysql> CREATE DATABASE d1;
Query OK, 1 row affected (0.00 sec)

mysql> USE d1;
Database changed

mysql> CREATE TABLE t1 (c1 INT);
Query OK, 0 rows affected (0.15 sec)

mysql> CREATE TABLE t2 (c1 INT);
Query OK, 0 rows affected (0.08 sec)
```

Now we create a stored function `f1` which modifies `t2`:

```
mysql> DELIMITER //
mysql> CREATE FUNCTION f1(p1 INT) RETURNS INT
mysql>   BEGIN
mysql>     INSERT INTO t2 VALUES (p1);
mysql>     RETURN p1;
mysql>   END //
Query OK, 0 rows affected (0.01 sec)

mysql> DELIMITER ;
```

Referencing the function directly in an `EXPLAIN SELECT` does not have any effect on `t2`, as shown here:

```
mysql> SELECT * FROM t2;
Empty set (0.00 sec)

mysql> EXPLAIN SELECT f1(5);
+----+-------------+-------+------+---------------+------+---------+------+------+----------------+
| id | select_type | table | type | possible_keys | key  | key_len | ref  | rows | Extra          |
+----+-------------+-------+------+---------------+------+---------+------+------+----------------+
|  1 | SIMPLE      | NULL  | NULL | NULL          | NULL | NULL    | NULL | NULL | No tables used |
+----+-------------+-------+------+---------------+------+---------+------+------+----------------+
1 row in set (0.00 sec)

mysql> SELECT * FROM t2;
Empty set (0.00 sec)
```

This is because the `SELECT` statement did not reference any tables, as can be seen in the `table` and `Extra` columns of the output. This is also true of the following nested `SELECT`:

```
mysql> EXPLAIN SELECT NOW() AS a1, (SELECT f1(5)) AS a2;
+----+-------------+-------+------+---------------+------+---------+------+------+----------------+
| id | select_type | table | type | possible_keys | key  | key_len | ref  | rows | Extra          |
+----+-------------+-------+------+---------------+------+---------+------+------+----------------+
|  1 | PRIMARY     | NULL  | NULL | NULL          | NULL | NULL    | NULL | NULL | No tables used |
+----+-------------+-------+------+---------------+------+---------+------+------+----------------+
1 row in set, 1 warning (0.00 sec)

mysql> SHOW WARNINGS;
+-------+------+------------------------------------------+
| Level | Code | Message                                  |
+-------+------+------------------------------------------+
| Note  | 1249 | Select 2 was reduced during optimization |
+-------+------+------------------------------------------+
1 row in set (0.00 sec)

mysql> SELECT * FROM t2;
Empty set (0.00 sec)
```

However, if the outer `SELECT` references any tables, the optimizer executes the statement in the subquery as well:

```
mysql> EXPLAIN SELECT * FROM t1 AS a1, (SELECT f1(5)) AS a2;
+----+-------------+------------+--------+---------------+------+---------+------+------+-----------------
| id | select_type | table      | type   | possible_keys | key  | key_len | ref  | rows | Extra
+----+-------------+------------+--------+---------------+------+---------+------+------+-----------------
|  1 | PRIMARY     | a1         | system | NULL          | NULL | NULL    | NULL |    0 | const row not fou
|  1 | PRIMARY     | <derived2> | system | NULL          | NULL | NULL    | NULL |    1 |
|  2 | DERIVED     | NULL       | NULL   | NULL          | NULL | NULL    | NULL | NULL | No tables used
+----+-------------+------------+--------+---------------+------+---------+------+------+-----------------
3 rows in set (0.00 sec)

mysql> SELECT * FROM t2;
+------+
| c1   |
+------+
|    5 |
+------+
1 row in set (0.00 sec)
```

This also means that an `EXPLAIN SELECT` statement such as the one shown here may take a long time to execute because the `BENCHMARK()` function is executed once for each row in `t1`:

```
EXPLAIN SELECT * FROM t1 AS a1, (SELECT BENCHMARK(1000000, MD5(NOW())));
```

13.2.10.9 Subquery Errors

There are some errors that apply only to subqueries. This section describes them.

- Unsupported subquery syntax:

  ```
  ERROR 1235 (ER_NOT_SUPPORTED_YET)
  SQLSTATE = 42000
  Message = "This version of MySQL doesn't yet support
  'LIMIT & IN/ALL/ANY/SOME subquery'"
  ```

  This means that MySQL does not support statements of the following form:

  ```
  SELECT * FROM t1 WHERE s1 IN (SELECT s2 FROM t2 ORDER BY s1 LIMIT 1)
  ```

- Incorrect number of columns from subquery:

  ```
  ERROR 1241 (ER_OPERAND_COL)
  SQLSTATE = 21000
  Message = "Operand should contain 1 column(s)"
  ```

  This error occurs in cases like this:

  ```
  SELECT (SELECT column1, column2 FROM t2) FROM t1;
  ```

  You may use a subquery that returns multiple columns, if the purpose is row comparison. In other contexts, the subquery must be a scalar operand. See Section 13.2.10.5, “Row Subqueries”.

- Incorrect number of rows from subquery:

  ```
  ERROR 1242 (ER_SUBSELECT_NO_1_ROW)
  SQLSTATE = 21000
  Message = "Subquery returns more than 1 row"
  ```

  This error occurs for statements where the subquery must return at most one row but returns multiple rows. Consider the following example:

  ```
  SELECT * FROM t1 WHERE column1 = (SELECT column1 FROM t2);
  ```

  If `SELECT column1 FROM t2` returns just one row, the previous query will work. If the subquery returns more than one row, error 1242 will occur. In that case, the query should be rewritten as:

  ```
  SELECT * FROM t1 WHERE column1 = ANY (SELECT column1 FROM t2);
  ```

- Incorrectly used table in subquery:

  ```
  Error 1093 (ER_UPDATE_TABLE_USED)
  SQLSTATE = HY000
  Message = "You can't specify target table 'x'
  for update in FROM clause"
  ```

  This error occurs in cases such as the following, which attempts to modify a table and select from the same table in the subquery:

```
UPDATE t1 SET column2 = (SELECT MAX(column1) FROM t1);
```

You can use a subquery for assignment within an UPDATE statement because subqueries are legal in UPDATE and DELETE statements as well as in SELECT statements. However, you cannot use the same table (in this case, table t1) for both the subquery FROM clause and the update target.

For transactional storage engines, the failure of a subquery causes the entire statement to fail. For nontransactional storage engines, data modifications made before the error was encountered are preserved.

13.2.10.10 Optimizing Subqueries

Development is ongoing, so no optimization tip is reliable for the long term. The following list provides some interesting tricks that you might want to play with:

- Use subquery clauses that affect the number or order of the rows in the subquery. For example:

```
SELECT * FROM t1 WHERE t1.column1 IN
  (SELECT column1 FROM t2 ORDER BY column1);
SELECT * FROM t1 WHERE t1.column1 IN
  (SELECT DISTINCT column1 FROM t2);
SELECT * FROM t1 WHERE EXISTS
  (SELECT * FROM t2 LIMIT 1);
```

- Replace a join with a subquery. For example, try this:

```
SELECT DISTINCT column1 FROM t1 WHERE t1.column1 IN (
  SELECT column1 FROM t2);
```

  Instead of this:

```
SELECT DISTINCT t1.column1 FROM t1, t2
  WHERE t1.column1 = t2.column1;
```

- Some subqueries can be transformed to joins for compatibility with older versions of MySQL that do not support subqueries. However, in some cases, converting a subquery to a join may improve performance. See Section 13.2.10.11, “Rewriting Subqueries as Joins”.

- Move clauses from outside to inside the subquery. For example, use this query:

```
SELECT * FROM t1
  WHERE s1 IN (SELECT s1 FROM t1 UNION ALL SELECT s1 FROM t2);
```

  Instead of this query:

```
SELECT * FROM t1
  WHERE s1 IN (SELECT s1 FROM t1) OR s1 IN (SELECT s1 FROM t2);
```

  For another example, use this query:

```
SELECT (SELECT column1 + 5 FROM t1) FROM t2;
```

  Instead of this query:

```
SELECT (SELECT column1 FROM t1) + 5 FROM t2;
```

- Use a row subquery instead of a correlated subquery. For example, use this query:

  ```
  SELECT * FROM t1
    WHERE (column1,column2) IN (SELECT column1,column2 FROM t2);
  ```

  Instead of this query:

  ```
  SELECT * FROM t1
    WHERE EXISTS (SELECT * FROM t2 WHERE t2.column1=t1.column1
                  AND t2.column2=t1.column2);
  ```

- Use `NOT (a = ANY (...))` rather than `a <> ALL (...)`.
- Use `x = ANY (table containing (1,2))` rather than `x=1 OR x=2`.
- Use `= ANY` rather than `EXISTS`.
- For uncorrelated subqueries that always return one row, `IN` is always slower than `=`. For example, use this query:

  ```
  SELECT * FROM t1
    WHERE t1.col_name = (SELECT a FROM t2 WHERE b = some_const);
  ```

  Instead of this query:

  ```
  SELECT * FROM t1
    WHERE t1.col_name IN (SELECT a FROM t2 WHERE b = some_const);
  ```

These tricks might cause programs to go faster or slower. Using MySQL facilities like the `BENCHMARK()` function, you can get an idea about what helps in your own situation. See Section 12.14, “Information Functions”.

Some optimizations that MySQL itself makes are:

- MySQL executes uncorrelated subqueries only once. Use `EXPLAIN` to make sure that a given subquery really is uncorrelated.
- MySQL rewrites `IN`, `ALL`, `ANY`, and `SOME` subqueries in an attempt to take advantage of the possibility that the select-list columns in the subquery are indexed.
- MySQL replaces subqueries of the following form with an index-lookup function, which `EXPLAIN` describes as a special join type (`unique_subquery` or `index_subquery`):

  ```
  ... IN (SELECT indexed_column FROM single_table ...)
  ```

- MySQL enhances expressions of the following form with an expression involving `MIN()` or `MAX()`, unless `NULL` values or empty sets are involved:

  ```
  value {ALL|ANY|SOME} {> | < | >= | <=} (uncorrelated subquery)
  ```

  For example, this `WHERE` clause:

  ```
  WHERE 5 > ALL (SELECT x FROM t)
  ```

  might be treated by the optimizer like this:

```
WHERE 5 > (SELECT MAX(x) FROM t)
```

See also MySQL Internals: How MySQL Transforms Subqueries.

13.2.10.11 Rewriting Subqueries as Joins

Sometimes there are other ways to test membership in a set of values than by using a subquery. Also, on some occasions, it is not only possible to rewrite a query without a subquery, but it can be more efficient to make use of some of these techniques rather than to use subqueries. One of these is the `IN()` construct:

For example, this query:

```
SELECT * FROM t1 WHERE id IN (SELECT id FROM t2);
```

Can be rewritten as:

```
SELECT DISTINCT t1.* FROM t1, t2 WHERE t1.id=t2.id;
```

The queries:

```
SELECT * FROM t1 WHERE id NOT IN (SELECT id FROM t2);
SELECT * FROM t1 WHERE NOT EXISTS (SELECT id FROM t2 WHERE t1.id=t2.id);
```

Can be rewritten as:

```
SELECT table1.*
  FROM table1 LEFT JOIN table2 ON table1.id=table2.id
  WHERE table2.id IS NULL;
```

A `LEFT [OUTER] JOIN` can be faster than an equivalent subquery because the server might be able to optimize it better—a fact that is not specific to MySQL Server alone. Prior to SQL-92, outer joins did not exist, so subqueries were the only way to do certain things. Today, MySQL Server and many other modern database systems offer a wide range of outer join types.

MySQL Server supports multiple-table `DELETE` statements that can be used to efficiently delete rows based on information from one table or even from many tables at the same time. Multiple-table `UPDATE` statements are also supported. See Section 13.2.2, “`DELETE` Syntax”, and Section 13.2.11, “`UPDATE` Syntax”.

13.2.11 `UPDATE` Syntax

Single-table syntax:

```
UPDATE [LOW_PRIORITY] [IGNORE] table_reference
    SET col_name1={expr1|DEFAULT} [, col_name2={expr2|DEFAULT}] ...
    [WHERE where_condition]
    [ORDER BY ...]
    [LIMIT row_count]
```

Multiple-table syntax:

```
UPDATE [LOW_PRIORITY] [IGNORE] table_references
```

```
    SET col_name1={expr1|DEFAULT} [, col_name2={expr2|DEFAULT}] ...
    [WHERE where_condition]
```

For the single-table syntax, the UPDATE statement updates columns of existing rows in the named table with new values. The SET clause indicates which columns to modify and the values they should be given. Each value can be given as an expression, or the keyword DEFAULT to set a column explicitly to its default value. The WHERE clause, if given, specifies the conditions that identify which rows to update. With no WHERE clause, all rows are updated. If the ORDER BY clause is specified, the rows are updated in the order that is specified. The LIMIT clause places a limit on the number of rows that can be updated.

For the multiple-table syntax, UPDATE updates rows in each table named in *table_references* that satisfy the conditions. Each matching row is updated once, even if it matches the conditions multiple times. For multiple-table syntax, ORDER BY and LIMIT cannot be used.

For partitioned tables, both the single-single and multiple-table forms of this statement support the use of a PARTITION option as part of a table reference. This option takes a list of one or more partitions or subpartitions (or both). Only the partitions (or subpartitions) listed are checked for matches, and a row that is not in any of these partitions or subpartitions is not updated, whether it satisfies the *where_condition* or not.

Note

Unlike the case when using PARTITION with an INSERT or REPLACE statement, an otherwise valid UPDATE ... PARTITION statement is considered successful even if no rows in the listed partitions (or subpartitions) match the *where_condition*.

See Section 17.5, “Partition Selection”, for more information and examples.

where_condition is an expression that evaluates to true for each row to be updated. For expression syntax, see Section 9.5, “Expression Syntax”.

table_references and *where_condition* are specified as described in Section 13.2.9, “SELECT Syntax”.

You need the UPDATE privilege only for columns referenced in an UPDATE that are actually updated. You need only the SELECT privilege for any columns that are read but not modified.

The UPDATE statement supports the following modifiers:

- With the LOW_PRIORITY keyword, execution of the UPDATE is delayed until no other clients are reading from the table. This affects only storage engines that use only table-level locking (such as MyISAM, MEMORY, and MERGE).

- With the IGNORE keyword, the update statement does not abort even if errors occur during the update. Rows for which duplicate-key conflicts occur are not updated. Rows for which columns are updated to values that would cause data conversion errors are updated to the closest valid values instead.

UPDATE IGNORE statements, including those having an ORDER BY clause, are flagged as unsafe for statement-based replication. (This is because the order in which the rows are updated determines which rows are ignored.) With this change, such statements produce a warning in the log when using statement-based mode and are logged using the row-based format when using MIXED mode. (Bug #11758262, Bug #50439) See Section 16.1.2.3, “Determination of Safe and Unsafe Statements in Binary Logging”, for more information.

If you access a column from the table to be updated in an expression, UPDATE uses the current value of the column. For example, the following statement sets col1 to one more than its current value:

```
UPDATE t1 SET col1 = col1 + 1;
```

The second assignment in the following statement sets `col2` to the current (updated) `col1` value, not the original `col1` value. The result is that `col1` and `col2` have the same value. This behavior differs from standard SQL.

```
UPDATE t1 SET col1 = col1 + 1, col2 = col1;
```

Single-table `UPDATE` assignments are generally evaluated from left to right. For multiple-table updates, there is no guarantee that assignments are carried out in any particular order.

If you set a column to the value it currently has, MySQL notices this and does not update it.

If you update a column that has been declared `NOT NULL` by setting to `NULL`, an error occurs if strict SQL mode is enabled; otherwise, the column is set to the implicit default value for the column data type and the warning count is incremented. The implicit default value is `0` for numeric types, the empty string (`''`) for string types, and the “zero” value for date and time types. See Section 11.5, “Data Type Default Values”.

`UPDATE` returns the number of rows that were actually changed. The `mysql_info()` C API function returns the number of rows that were matched and updated and the number of warnings that occurred during the `UPDATE`.

You can use `LIMIT` *`row_count`* to restrict the scope of the `UPDATE`. A `LIMIT` clause is a rows-matched restriction. The statement stops as soon as it has found *`row_count`* rows that satisfy the `WHERE` clause, whether or not they actually were changed.

If an `UPDATE` statement includes an `ORDER BY` clause, the rows are updated in the order specified by the clause. This can be useful in certain situations that might otherwise result in an error. Suppose that a table `t` contains a column `id` that has a unique index. The following statement could fail with a duplicate-key error, depending on the order in which rows are updated:

```
UPDATE t SET id = id + 1;
```

For example, if the table contains 1 and 2 in the `id` column and 1 is updated to 2 before 2 is updated to 3, an error occurs. To avoid this problem, add an `ORDER BY` clause to cause the rows with larger `id` values to be updated before those with smaller values:

```
UPDATE t SET id = id + 1 ORDER BY id DESC;
```

You can also perform `UPDATE` operations covering multiple tables. However, you cannot use `ORDER BY` or `LIMIT` with a multiple-table `UPDATE`. The *`table_references`* clause lists the tables involved in the join. Its syntax is described in Section 13.2.9.2, “`JOIN` Syntax”. Here is an example:

```
UPDATE items,month SET items.price=month.price
WHERE items.id=month.id;
```

The preceding example shows an inner join that uses the comma operator, but multiple-table `UPDATE` statements can use any type of join permitted in `SELECT` statements, such as `LEFT JOIN`.

If you use a multiple-table `UPDATE` statement involving `InnoDB` tables for which there are foreign key constraints, the MySQL optimizer might process tables in an order that differs from that of their parent/child relationship. In this case, the statement fails and rolls back. Instead, update a single table and rely on the

ON UPDATE capabilities that InnoDB provides to cause the other tables to be modified accordingly. See Section 14.2.6.6, "InnoDB and FOREIGN KEY Constraints".

Currently, you cannot update a table and select from the same table in a subquery.

Index hints (see Section 13.2.9.3, "Index Hint Syntax") are accepted but ignored for UPDATE statements.

In MySQL 5.7, an UPDATE on a partitioned table using a storage engine such as MyISAM that employs table-level locks locks only those partitions containing rows that match the UPDATE statement's WHERE clause, as long as none of the table's partitioning columns are updated. (For storage engines such as InnoDB that employ row-level locking, no locking of partitions takes place.) For more information, see Section 17.6.4, "Partitioning and Locking".

13.3 MySQL Transactional and Locking Statements

MySQL supports local transactions (within a given client session) through statements such as SET autocommit, START TRANSACTION, COMMIT, and ROLLBACK. See Section 13.3.1, "START TRANSACTION, COMMIT, and ROLLBACK Syntax". XA transaction support enables MySQL to participate in distributed transactions as well. See Section 13.3.7, "XA Transactions".

13.3.1 START TRANSACTION, COMMIT, and ROLLBACK Syntax

```
START TRANSACTION
    [transaction_characteristic [, transaction_characteristic] ...]

transaction_characteristic:
    WITH CONSISTENT SNAPSHOT
  | READ WRITE
  | READ ONLY

BEGIN [WORK]
COMMIT [WORK] [AND [NO] CHAIN] [[NO] RELEASE]
ROLLBACK [WORK] [AND [NO] CHAIN] [[NO] RELEASE]
SET autocommit = {0 | 1}
```

These statements provide control over use of transactions:

- START TRANSACTION or BEGIN start a new transaction.
- COMMIT commits the current transaction, making its changes permanent.
- ROLLBACK rolls back the current transaction, canceling its changes.
- SET autocommit disables or enables the default autocommit mode for the current session.

By default, MySQL runs with autocommit mode enabled. This means that as soon as you execute a statement that updates (modifies) a table, MySQL stores the update on disk to make it permanent. The change cannot be rolled back.

To disable autocommit mode implicitly for a single series of statements, use the START TRANSACTION statement:

```
START TRANSACTION;
SELECT @A:=SUM(salary) FROM table1 WHERE type=1;
UPDATE table2 SET summary=@A WHERE type=1;
COMMIT;
```

With `START TRANSACTION`, autocommit remains disabled until you end the transaction with `COMMIT` or `ROLLBACK`. The autocommit mode then reverts to its previous state.

`START TRANSACTION` permits several modifiers that control transaction characteristics. To specify multiple modifiers, separate them by commas.

- The `WITH CONSISTENT SNAPSHOT` modifier starts a consistent read for storage engines that are capable of it. This applies only to `InnoDB`. The effect is the same as issuing a `START TRANSACTION` followed by a `SELECT` from any `InnoDB` table. See Section 14.2.2.4, “Consistent Nonlocking Reads”. The `WITH CONSISTENT SNAPSHOT` modifier does not change the current transaction isolation level, so it provides a consistent snapshot only if the current isolation level is one that permits a consistent read. The only isolation level that permits a consistent read is `REPEATABLE READ`. For all other isolation levels, the `WITH CONSISTENT SNAPSHOT` clause is ignored. As of MySQL 5.7.2, a warning is generated when the `WITH CONSISTENT SNAPSHOT` clause is ignored.

- The `READ WRITE` and `READ ONLY` modifiers set the transaction access mode. They permit or prohibit changes to tables used in the transaction. The `READ ONLY` restriction prevents the transaction from modifying or locking both transactional and nontransactional tables that are visible to other transactions; the transaction can still modify or lock temporary tables.

  MySQL enables extra optimizations for queries on `InnoDB` tables when the transaction is known to be read-only. Specifying `READ ONLY` ensures these optimizations are applied in cases where the read-only status cannot be determined automatically. See Optimizations for Read-Only Transactions for more information.

  If no access mode is specified, the default mode applies. Unless the default has been changed, it is read/write. It is not permitted to specify both `READ WRITE` and `READ ONLY` in the same statement.

  In read-only mode, it remains possible to change tables created with the `TEMPORARY` keyword using DML statements. Changes made with DDL statements are not permitted, just as with permanent tables.

  For additional information about transaction access mode, including ways to change the default mode, see Section 13.3.6, “`SET TRANSACTION` Syntax”.

  If the `read_only` system variable is enabled, explicitly starting a transaction with `START TRANSACTION READ WRITE` requires the `SUPER` privilege.

> ⚠ **Important**
>
> Many APIs used for writing MySQL client applications (such as JDBC) provide their own methods for starting transactions that can (and sometimes should) be used instead of sending a `START TRANSACTION` statement from the client. See Chapter 21, *Connectors and APIs*, or the documentation for your API, for more information.

To disable autocommit mode explicitly, use the following statement:

```
SET autocommit=0;
```

After disabling autocommit mode by setting the `autocommit` variable to zero, changes to transaction-safe tables (such as those for `InnoDB` or `NDBCLUSTER`) are not made permanent immediately. You must use `COMMIT` to store your changes to disk or `ROLLBACK` to ignore the changes.

`autocommit` is a session variable and must be set for each session. To disable autocommit mode for each new connection, see the description of the `autocommit` system variable at Section 5.1.4, “Server System Variables”.

BEGIN and BEGIN WORK are supported as aliases of START TRANSACTION for initiating a transaction. START TRANSACTION is standard SQL syntax, is the recommended way to start an ad-hoc transaction, and permits modifiers that BEGIN does not.

The BEGIN statement differs from the use of the BEGIN keyword that starts a BEGIN ... END compound statement. The latter does not begin a transaction. See Section 13.6.1, “BEGIN ... END Compound-Statement Syntax”.

Note

Within all stored programs (stored procedures and functions, triggers, and events), the parser treats BEGIN [WORK] as the beginning of a BEGIN ... END block. Begin a transaction in this context with START TRANSACTION instead.

The optional WORK keyword is supported for COMMIT and ROLLBACK, as are the CHAIN and RELEASE clauses. CHAIN and RELEASE can be used for additional control over transaction completion. The value of the completion_type system variable determines the default completion behavior. See Section 5.1.4, “Server System Variables”.

The AND CHAIN clause causes a new transaction to begin as soon as the current one ends, and the new transaction has the same isolation level as the just-terminated transaction. The RELEASE clause causes the server to disconnect the current client session after terminating the current transaction. Including the NO keyword suppresses CHAIN or RELEASE completion, which can be useful if the completion_type system variable is set to cause chaining or release completion by default.

Beginning a transaction causes any pending transaction to be committed. See Section 13.3.3, “Statements That Cause an Implicit Commit”, for more information.

Beginning a transaction also causes table locks acquired with LOCK TABLES to be released, as though you had executed UNLOCK TABLES. Beginning a transaction does not release a global read lock acquired with FLUSH TABLES WITH READ LOCK.

For best results, transactions should be performed using only tables managed by a single transaction-safe storage engine. Otherwise, the following problems can occur:

- If you use tables from more than one transaction-safe storage engine (such as InnoDB), and the transaction isolation level is not SERIALIZABLE, it is possible that when one transaction commits, another ongoing transaction that uses the same tables will see only some of the changes made by the first transaction. That is, the atomicity of transactions is not guaranteed with mixed engines and inconsistencies can result. (If mixed-engine transactions are infrequent, you can use SET TRANSACTION ISOLATION LEVEL to set the isolation level to SERIALIZABLE on a per-transaction basis as necessary.)

- If you use tables that are not transaction-safe within a transaction, changes to those tables are stored at once, regardless of the status of autocommit mode.

- If you issue a ROLLBACK statement after updating a nontransactional table within a transaction, an ER_WARNING_NOT_COMPLETE_ROLLBACK warning occurs. Changes to transaction-safe tables are rolled back, but not changes to nontransaction-safe tables.

Each transaction is stored in the binary log in one chunk, upon COMMIT. Transactions that are rolled back are not logged. (**Exception**: Modifications to nontransactional tables cannot be rolled back. If a transaction that is rolled back includes modifications to nontransactional tables, the entire transaction is logged with a ROLLBACK statement at the end to ensure that modifications to the nontransactional tables are replicated.) See Section 5.2.4, “The Binary Log”.

You can change the isolation level or access mode for transactions with the `SET TRANSACTION` statement. See Section 13.3.6, "`SET TRANSACTION` Syntax".

Rolling back can be a slow operation that may occur implicitly without the user having explicitly asked for it (for example, when an error occurs). Because of this, `SHOW PROCESSLIST` displays `Rolling back` in the `State` column for the session, not only for explicit rollbacks performed with the `ROLLBACK` statement but also for implicit rollbacks.

> **Note**
>
> In MySQL 5.7, `BEGIN`, `COMMIT`, and `ROLLBACK` are not affected by `--replicate-do-db` or `--replicate-ignore-db` rules.

13.3.2 Statements That Cannot Be Rolled Back

Some statements cannot be rolled back. In general, these include data definition language (DDL) statements, such as those that create or drop databases, those that create, drop, or alter tables or stored routines.

You should design your transactions not to include such statements. If you issue a statement early in a transaction that cannot be rolled back, and then another statement later fails, the full effect of the transaction cannot be rolled back in such cases by issuing a `ROLLBACK` statement.

13.3.3 Statements That Cause an Implicit Commit

The statements listed in this section (and any synonyms for them) implicitly end any transaction active in the current session, as if you had done a `COMMIT` before executing the statement. As of MySQL 5.5.3, most of these statements also cause an implicit commit after executing; for additional details, see the end of this section.

- **Data definition language (DDL) statements that define or modify database objects.** `ALTER DATABASE ... UPGRADE DATA DIRECTORY NAME`, `ALTER EVENT`, `ALTER PROCEDURE`, `ALTER SERVER`, `ALTER TABLE`, `ALTER VIEW`, `CREATE DATABASE`, `CREATE EVENT`, `CREATE INDEX`, `CREATE PROCEDURE`, `CREATE SERVER`, `CREATE TABLE`, `CREATE TRIGGER`, `CREATE VIEW`, `DROP DATABASE`, `DROP EVENT`, `DROP INDEX`, `DROP PROCEDURE`, `DROP SERVER`, `DROP TABLE`, `DROP TRIGGER`, `DROP VIEW`, `RENAME TABLE`, `TRUNCATE TABLE`.

  `ALTER FUNCTION`, `CREATE FUNCTION` and `DROP FUNCTION` also cause an implicit commit when used with stored functions, but not with UDFs. (`ALTER FUNCTION` can only be used with stored functions.)

  `CREATE TABLE` and `DROP TABLE` statements do not commit a transaction if the `TEMPORARY` keyword is used. (This does not apply to other operations on temporary tables such as `ALTER TABLE` and `CREATE INDEX`, which do cause a commit.) However, although no implicit commit occurs, neither can the statement be rolled back, which means that the use of such statements causes transactional atomicity to be violated. For example, if you use `CREATE TEMPORARY TABLE` and then roll back the transaction, the table remains in existence.

  The `CREATE TABLE` statement in `InnoDB` is processed as a single transaction. This means that a `ROLLBACK` from the user does not undo `CREATE TABLE` statements the user made during that transaction.

  `CREATE TABLE ... SELECT` causes an implicit commit before and after the statement is executed when you are creating nontemporary tables. (No commit occurs for `CREATE TEMPORARY TABLE ... SELECT`.) This is to prevent an issue during replication where the table could be created on the master after a rollback, but fail to be recorded in the binary log, and therefore not replicated to the slave. For more information, see Bug #22865.

- **Statements that implicitly use or modify tables in the mysql database.** CREATE USER, DROP USER, GRANT, RENAME USER, REVOKE, SET PASSWORD.
- **Transaction-control and locking statements.** BEGIN, LOCK TABLES, SET autocommit = 1 (if the value is not already 1), START TRANSACTION, UNLOCK TABLES.

  UNLOCK TABLES commits a transaction only if any tables currently have been locked with LOCK TABLES to acquire nontransactional table locks. A commit does not occur for UNLOCK TABLES following FLUSH TABLES WITH READ LOCK because the latter statement does not acquire table-level locks.

  Transactions cannot be nested. This is a consequence of the implicit commit performed for any current transaction when you issue a START TRANSACTION statement or one of its synonyms.

  Statements that cause an implicit commit cannot be used in an XA transaction while the transaction is in an ACTIVE state.

  The BEGIN statement differs from the use of the BEGIN keyword that starts a BEGIN ... END compound statement. The latter does not cause an implicit commit. See Section 13.6.1, “BEGIN ... END Compound-Statement Syntax”.
- **Data loading statements.** LOAD DATA INFILE. LOAD DATA INFILE causes an implicit commit only for tables using the NDB storage engine. For more information, see Bug #11151.
- **Administrative statements.** ANALYZE TABLE, CACHE INDEX, CHECK TABLE, LOAD INDEX INTO CACHE, OPTIMIZE TABLE, REPAIR TABLE.
- **Replication control statements**. START SLAVE, STOP SLAVE, RESET SLAVE, CHANGE MASTER TO.

As of MySQL 5.5.3, most statements that previously caused an implicit commit before executing also do so after executing. The intent is to handle each such statement in its own special transaction because it cannot be rolled back anyway. The following list provides additional details pertaining to this change:

- The CREATE TABLE variants (CREATE TABLE for InnoDB tables and CREATE TABLE ... SELECT) that previously were special cases no longer are so because CREATE TABLE uniformly causes an implicit commit before and after executing.
- The FLUSH and RESET statements cause an implicit commit.
- Transaction-control and locking statements behave as before.

13.3.4 SAVEPOINT, ROLLBACK TO SAVEPOINT, and RELEASE SAVEPOINT Syntax

```
SAVEPOINT identifier
ROLLBACK [WORK] TO [SAVEPOINT] identifier
RELEASE SAVEPOINT identifier
```

InnoDB supports the SQL statements SAVEPOINT, ROLLBACK TO SAVEPOINT, RELEASE SAVEPOINT and the optional WORK keyword for ROLLBACK.

The SAVEPOINT statement sets a named transaction savepoint with a name of *identifier*. If the current transaction has a savepoint with the same name, the old savepoint is deleted and a new one is set.

The ROLLBACK TO SAVEPOINT statement rolls back a transaction to the named savepoint without terminating the transaction. Modifications that the current transaction made to rows after the savepoint was set are undone in the rollback, but InnoDB does *not* release the row locks that were stored in memory after the savepoint. (For a new inserted row, the lock information is carried by the transaction ID stored in the row; the lock is not separately stored in memory. In this case, the row lock is released in the undo.) Savepoints that were set at a later time than the named savepoint are deleted.

If the ROLLBACK TO SAVEPOINT statement returns the following error, it means that no savepoint with the specified name exists:

```
ERROR 1305 (42000): SAVEPOINT identifier does not exist
```

The RELEASE SAVEPOINT statement removes the named savepoint from the set of savepoints of the current transaction. No commit or rollback occurs. It is an error if the savepoint does not exist.

All savepoints of the current transaction are deleted if you execute a COMMIT, or a ROLLBACK that does not name a savepoint.

A new savepoint level is created when a stored function is invoked or a trigger is activated. The savepoints on previous levels become unavailable and thus do not conflict with savepoints on the new level. When the function or trigger terminates, any savepoints it created are released and the previous savepoint level is restored.

13.3.5 LOCK TABLES and UNLOCK TABLES Syntax

```
LOCK TABLES
    tbl_name [[AS] alias] lock_type
    [, tbl_name [[AS] alias] lock_type] ...

lock_type:
    READ [LOCAL]
  | [LOW_PRIORITY] WRITE

UNLOCK TABLES
```

MySQL enables client sessions to acquire table locks explicitly for the purpose of cooperating with other sessions for access to tables, or to prevent other sessions from modifying tables during periods when a session requires exclusive access to them. A session can acquire or release locks only for itself. One session cannot acquire locks for another session or release locks held by another session.

Locks may be used to emulate transactions or to get more speed when updating tables. This is explained in more detail later in this section.

LOCK TABLES explicitly acquires table locks for the current client session. Table locks can be acquired for base tables or views. You must have the LOCK TABLES privilege, and the SELECT privilege for each object to be locked.

For view locking, LOCK TABLES adds all base tables used in the view to the set of tables to be locked and locks them automatically. If you lock a table explicitly with LOCK TABLES, any tables used in triggers are also locked implicitly, as described in Section 13.3.5.2, “LOCK TABLES and Triggers”.

UNLOCK TABLES explicitly releases any table locks held by the current session. LOCK TABLES implicitly releases any table locks held by the current session before acquiring new locks.

Another use for UNLOCK TABLES is to release the global read lock acquired with the FLUSH TABLES WITH READ LOCK statement, which enables you to lock all tables in all databases. See Section 13.7.6.3, “FLUSH Syntax”. (This is a very convenient way to get backups if you have a file system such as Veritas that can take snapshots in time.)

A table lock protects only against inappropriate reads or writes by other sessions. The session holding the lock, even a read lock, can perform table-level operations such as DROP TABLE. Truncate operations are not transaction-safe, so an error occurs if the session attempts one during an active transaction or while holding a table lock.

The following discussion applies only to non-TEMPORARY tables. LOCK TABLES is permitted (but ignored) for a TEMPORARY table. The table can be accessed freely by the session within which it was created, regardless of what other locking may be in effect. No lock is necessary because no other session can see the table.

For information about other conditions on the use of LOCK TABLES and statements that cannot be used while LOCK TABLES is in effect, see Section 13.3.5.3, “Table-Locking Restrictions and Conditions”

Rules for Lock Acquisition

To acquire table locks within the current session, use the LOCK TABLES statement. The following lock types are available:

READ [LOCAL] lock:

- The session that holds the lock can read the table (but not write it).
- Multiple sessions can acquire a READ lock for the table at the same time.
- Other sessions can read the table without explicitly acquiring a READ lock.
- The LOCAL modifier enables nonconflicting INSERT statements (concurrent inserts) by other sessions to execute while the lock is held. (See Section 8.10.3, “Concurrent Inserts”.) However, READ LOCAL cannot be used if you are going to manipulate the database using processes external to the server while you hold the lock. For InnoDB tables, READ LOCAL is the same as READ.

[LOW_PRIORITY] WRITE lock:

- The session that holds the lock can read and write the table.
- Only the session that holds the lock can access the table. No other session can access it until the lock is released.
- Lock requests for the table by other sessions block while the WRITE lock is held.
- The LOW_PRIORITY modifier has no effect. In previous versions of MySQL, it affected locking behavior, but this is no longer true. It is now deprecated and its use produces a warning. Use WRITE without LOW_PRIORITY instead.

If the LOCK TABLES statement must wait due to locks held by other sessions on any of the tables, it blocks until all locks can be acquired.

A session that requires locks must acquire all the locks that it needs in a single LOCK TABLES statement. While the locks thus obtained are held, the session can access only the locked tables. For example, in the following sequence of statements, an error occurs for the attempt to access t2 because it was not locked in the LOCK TABLES statement:

```
mysql> LOCK TABLES t1 READ;
mysql> SELECT COUNT(*) FROM t1;
+----------+
| COUNT(*) |
+----------+
|        3 |
+----------+
mysql> SELECT COUNT(*) FROM t2;
ERROR 1100 (HY000): Table 't2' was not locked with LOCK TABLES
```

Tables in the INFORMATION_SCHEMA database are an exception. They can be accessed without being locked explicitly even while a session holds table locks obtained with LOCK TABLES.

You cannot refer to a locked table multiple times in a single query using the same name. Use aliases instead, and obtain a separate lock for the table and each alias:

```
mysql> LOCK TABLE t WRITE, t AS t1 READ;
mysql> INSERT INTO t SELECT * FROM t;
ERROR 1100: Table 't' was not locked with LOCK TABLES
mysql> INSERT INTO t SELECT * FROM t AS t1;
```

The error occurs for the first INSERT because there are two references to the same name for a locked table. The second INSERT succeeds because the references to the table use different names.

If your statements refer to a table by means of an alias, you must lock the table using that same alias. It does not work to lock the table without specifying the alias:

```
mysql> LOCK TABLE t READ;
mysql> SELECT * FROM t AS myalias;
ERROR 1100: Table 'myalias' was not locked with LOCK TABLES
```

Conversely, if you lock a table using an alias, you must refer to it in your statements using that alias:

```
mysql> LOCK TABLE t AS myalias READ;
mysql> SELECT * FROM t;
ERROR 1100: Table 't' was not locked with LOCK TABLES
mysql> SELECT * FROM t AS myalias;
```

WRITE locks normally have higher priority than READ locks to ensure that updates are processed as soon as possible. This means that if one session obtains a READ lock and then another session requests a WRITE lock, subsequent READ lock requests wait until the session that requested the WRITE lock has obtained the lock and released it.

LOCK TABLES acquires locks as follows:

1. Sort all tables to be locked in an internally defined order. From the user standpoint, this order is undefined.

2. If a table is to be locked with a read and a write lock, put the write lock request before the read lock request.

3. Lock one table at a time until the session gets all locks.

This policy ensures that table locking is deadlock free.

Note

LOCK TABLES or UNLOCK TABLES, when applied to a partitioned table, always locks or unlocks the entire table; these statements do not support partition lock pruning. See Section 17.6.4, “Partitioning and Locking”.

Rules for Lock Release

When the table locks held by a session are released, they are all released at the same time. A session can release its locks explicitly, or locks may be released implicitly under certain conditions.

- A session can release its locks explicitly with `UNLOCK TABLES`.
- If a session issues a `LOCK TABLES` statement to acquire a lock while already holding locks, its existing locks are released implicitly before the new locks are granted.
- If a session begins a transaction (for example, with `START TRANSACTION`), an implicit `UNLOCK TABLES` is performed, which causes existing locks to be released. (For additional information about the interaction between table locking and transactions, see Section 13.3.5.1, "Interaction of Table Locking and Transactions".)

If the connection for a client session terminates, whether normally or abnormally, the server implicitly releases all table locks held by the session (transactional and nontransactional). If the client reconnects, the locks will no longer be in effect. In addition, if the client had an active transaction, the server rolls back the transaction upon disconnect, and if reconnect occurs, the new session begins with autocommit enabled. For this reason, clients may wish to disable auto-reconnect. With auto-reconnect in effect, the client is not notified if reconnect occurs but any table locks or current transaction will have been lost. With auto-reconnect disabled, if the connection drops, an error occurs for the next statement issued. The client can detect the error and take appropriate action such as reacquiring the locks or redoing the transaction. See Section 21.8.16, "Controlling Automatic Reconnection Behavior".

Note

If you use `ALTER TABLE` on a locked table, it may become unlocked. For example, if you attempt a second `ALTER TABLE` operation, the result may be an error `Table '`*`tbl_name`*`' was not locked with LOCK TABLES`. To handle this, lock the table again prior to the second alteration. See also Section C.5.7.1, "Problems with `ALTER TABLE`".

13.3.5.1 Interaction of Table Locking and Transactions

`LOCK TABLES` and `UNLOCK TABLES` interact with the use of transactions as follows:

- `LOCK TABLES` is not transaction-safe and implicitly commits any active transaction before attempting to lock the tables.
- `UNLOCK TABLES` implicitly commits any active transaction, but only if `LOCK TABLES` has been used to acquire table locks. For example, in the following set of statements, `UNLOCK TABLES` releases the global read lock but does not commit the transaction because no table locks are in effect:

```
FLUSH TABLES WITH READ LOCK;
START TRANSACTION;
SELECT ... ;
UNLOCK TABLES;
```

- Beginning a transaction (for example, with `START TRANSACTION`) implicitly commits any current transaction and releases existing table locks.
- `FLUSH TABLES WITH READ LOCK` acquires a global read lock and not table locks, so it is not subject to the same behavior as `LOCK TABLES` and `UNLOCK TABLES` with respect to table locking and implicit commits. For example, `START TRANSACTION` does not release the global read lock. See Section 13.7.6.3, "`FLUSH` Syntax".

- Other statements that implicitly cause transactions to be committed do not release existing table locks. For a list of such statements, see Section 13.3.3, “Statements That Cause an Implicit Commit”.

- The correct way to use `LOCK TABLES` and `UNLOCK TABLES` with transactional tables, such as `InnoDB` tables, is to begin a transaction with `SET autocommit = 0` (not `START TRANSACTION`) followed by `LOCK TABLES`, and to not call `UNLOCK TABLES` until you commit the transaction explicitly. For example, if you need to write to table `t1` and read from table `t2`, you can do this:

  ```
  SET autocommit=0;
  LOCK TABLES t1 WRITE, t2 READ, ...;
  ... do something with tables t1 and t2 here ...
  COMMIT;
  UNLOCK TABLES;
  ```

  When you call `LOCK TABLES`, `InnoDB` internally takes its own table lock, and MySQL takes its own table lock. `InnoDB` releases its internal table lock at the next commit, but for MySQL to release its table lock, you have to call `UNLOCK TABLES`. You should not have `autocommit = 1`, because then `InnoDB` releases its internal table lock immediately after the call of `LOCK TABLES`, and deadlocks can very easily happen. `InnoDB` does not acquire the internal table lock at all if `autocommit = 1`, to help old applications avoid unnecessary deadlocks.

- `ROLLBACK` does not release table locks.

13.3.5.2 `LOCK TABLES` and Triggers

If you lock a table explicitly with `LOCK TABLES`, any tables used in triggers are also locked implicitly:

- The locks are taken as the same time as those acquired explicitly with the `LOCK TABLES` statement.

- The lock on a table used in a trigger depends on whether the table is used only for reading. If so, a read lock suffices. Otherwise, a write lock is used.

- If a table is locked explicitly for reading with `LOCK TABLES`, but needs to be locked for writing because it might be modified within a trigger, a write lock is taken rather than a read lock. (That is, an implicit write lock needed due to the table's appearance within a trigger causes an explicit read lock request for the table to be converted to a write lock request.)

Suppose that you lock two tables, `t1` and `t2`, using this statement:

```
LOCK TABLES t1 WRITE, t2 READ;
```

If `t1` or `t2` have any triggers, tables used within the triggers will also be locked. Suppose that `t1` has a trigger defined like this:

```
CREATE TRIGGER t1_a_ins AFTER INSERT ON t1 FOR EACH ROW
BEGIN
  UPDATE t4 SET count = count+1
      WHERE id = NEW.id AND EXISTS (SELECT a FROM t3);
  INSERT INTO t2 VALUES(1, 2);
END;
```

The result of the `LOCK TABLES` statement is that `t1` and `t2` are locked because they appear in the statement, and `t3` and `t4` are locked because they are used within the trigger:

- `t1` is locked for writing per the `WRITE` lock request.

- `t2` is locked for writing, even though the request is for a `READ` lock. This occurs because `t2` is inserted into within the trigger, so the `READ` request is converted to a `WRITE` request.

- t3 is locked for reading because it is only read from within the trigger.
- t4 is locked for writing because it might be updated within the trigger.

13.3.5.3 Table-Locking Restrictions and Conditions

You can safely use KILL to terminate a session that is waiting for a table lock. See Section 13.7.6.4, “KILL Syntax”.

LOCK TABLES and UNLOCK TABLES cannot be used within stored programs.

Tables in the performance_schema database cannot be locked with LOCK TABLES, except the setup_*xxx* tables.

The following statements are prohibited while a LOCK TABLES statement is in effect: CREATE TABLE, CREATE TABLE ... LIKE, CREATE VIEW, DROP VIEW, and DDL statements on stored functions and procedures and events.

For some operations, system tables in the mysql database must be accessed. For example, the HELP statement requires the contents of the server-side help tables, and CONVERT_TZ() might need to read the time zone tables. The server implicitly locks the system tables for reading as necessary so that you need not lock them explicitly. These tables are treated as just described:

```
mysql.help_category
mysql.help_keyword
mysql.help_relation
mysql.help_topic
mysql.proc
mysql.time_zone
mysql.time_zone_leap_second
mysql.time_zone_name
mysql.time_zone_transition
mysql.time_zone_transition_type
```

If you want to explicitly place a WRITE lock on any of those tables with a LOCK TABLES statement, the table must be the only one locked; no other table can be locked with the same statement.

Normally, you do not need to lock tables, because all single UPDATE statements are atomic; no other session can interfere with any other currently executing SQL statement. However, there are a few cases when locking tables may provide an advantage:

- If you are going to run many operations on a set of MyISAM tables, it is much faster to lock the tables you are going to use. Locking MyISAM tables speeds up inserting, updating, or deleting on them because MySQL does not flush the key cache for the locked tables until UNLOCK TABLES is called. Normally, the key cache is flushed after each SQL statement.

  The downside to locking the tables is that no session can update a READ-locked table (including the one holding the lock) and no session can access a WRITE-locked table other than the one holding the lock.

- If you are using tables for a nontransactional storage engine, you must use LOCK TABLES if you want to ensure that no other session modifies the tables between a SELECT and an UPDATE. The example shown here requires LOCK TABLES to execute safely:

  ```
  LOCK TABLES trans READ, customer WRITE;
  SELECT SUM(value) FROM trans WHERE customer_id=some_id;
  UPDATE customer
    SET total_value=sum_from_previous_statement
  ```

```
  WHERE customer_id=some_id;
UNLOCK TABLES;
```

Without LOCK TABLES, it is possible that another session might insert a new row in the trans table between execution of the SELECT and UPDATE statements.

You can avoid using LOCK TABLES in many cases by using relative updates (UPDATE customer SET value=value+new_value) or the LAST_INSERT_ID() function. See Section 1.8.2.3, "Transaction and Atomic Operation Differences".

You can also avoid locking tables in some cases by using the user-level advisory lock functions GET_LOCK() and RELEASE_LOCK(). These locks are saved in a hash table in the server and implemented with pthread_mutex_lock() and pthread_mutex_unlock() for high speed. See Section 12.16, "Miscellaneous Functions".

See Section 8.10.1, "Internal Locking Methods", for more information on locking policy.

13.3.6 SET TRANSACTION Syntax

```
SET [GLOBAL | SESSION] TRANSACTION
    transaction_characteristic [, transaction_characteristic] ...

transaction_characteristic:
    ISOLATION LEVEL level
  | READ WRITE
  | READ ONLY

level:
     REPEATABLE READ
   | READ COMMITTED
   | READ UNCOMMITTED
   | SERIALIZABLE
```

This statement specifies transaction characteristics. It takes a list of one or more characteristic values separated by commas. These characteristics set the transaction isolation level or access mode. The isolation level is used for operations on InnoDB tables. The access mode may be specified as to whether transactions operate in read/write or read-only mode.

In addition, SET TRANSACTION can include an optional GLOBAL or SESSION keyword to indicate the scope of the statement.

Scope of Transaction Characteristics

You can set transaction characteristics globally, for the current session, or for the next transaction:

- With the GLOBAL keyword, the statement applies globally for all subsequent sessions. Existing sessions are unaffected.
- With the SESSION keyword, the statement applies to all subsequent transactions performed within the current session.
- Without any SESSION or GLOBAL keyword, the statement applies to the next (not started) transaction performed within the current session.

A global change to transaction characteristics requires the SUPER privilege. Any session is free to change its session characteristics (even in the middle of a transaction), or the characteristics for its next transaction.

SET TRANSACTION without GLOBAL or SESSION is not permitted while there is an active transaction:

```
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.02 sec)

mysql> SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
ERROR 1568 (25001): Transaction characteristics can't be changed
while a transaction is in progress
```

To set the global default isolation level at server startup, use the --transaction-isolation=*level* option to mysqld on the command line or in an option file. Values of *level* for this option use dashes rather than spaces, so the permissible values are READ-UNCOMMITTED, READ-COMMITTED, REPEATABLE-READ, or SERIALIZABLE. For example, to set the default isolation level to REPEATABLE READ, use these lines in the [mysqld] section of an option file:

```
[mysqld]
transaction-isolation = REPEATABLE-READ
```

It is possible to check or set the global and session transaction isolation levels at runtime by using the tx_isolation system variable:

```
SELECT @@GLOBAL.tx_isolation, @@tx_isolation;
SET GLOBAL tx_isolation='REPEATABLE-READ';
SET SESSION tx_isolation='SERIALIZABLE';
```

Similarly, to set the transaction access mode at server startup or at runtime, use the --transaction-read-only option or tx_read_only system variable. By default, these are OFF (the mode is read/write) but can be set to ON for a default mode of read only.

Setting the global or session value of tx_isolation or tx_read_only is equivalent to setting the isolation level or access mode with SET GLOBAL TRANSACTION or SET SESSION TRANSACTION.

Details and Usage of Isolation Levels

InnoDB supports each of the transaction isolation levels described here using different locking strategies. You can enforce a high degree of consistency with the default REPEATABLE READ level, for operations on crucial data where ACID compliance is important. Or you can relax the consistency rules with READ COMMITTED or even READ UNCOMMITTED, in situations such as bulk reporting where precise consistency and repeatable results are less important than minimizing the amount of overhead for locking. SERIALIZABLE enforces even stricter rules than REPEATABLE READ, and is used mainly in specialized situations, such as with XA transactions and for troubleshooting issues with concurrency and deadlocks.

For full information about how these isolation levels work with InnoDB transactions, see Section 14.2.2.2, “The InnoDB Transaction Model and Locking”. In particular, for additional information about InnoDB record-level locks and how it uses them to execute various types of statements, see Section 14.2.2.6, “InnoDB Record, Gap, and Next-Key Locks” and Section 14.2.2.8, “Locks Set by Different SQL Statements in InnoDB”.

The following list describes how MySQL supports the different transaction levels. The list goes from the most commonly used level to the least used.

- REPEATABLE READ

  This is the default isolation level for InnoDB. For consistent reads, there is an important difference from the READ COMMITTED isolation level: All consistent reads within the same transaction read the snapshot established by the first read. This convention means that if you issue several plain (nonlocking) `SELECT` statements within the same transaction, these `SELECT` statements are consistent also with respect to each other. See Section 14.2.2.4, “Consistent Nonlocking Reads”.

For locking reads (`SELECT` with `FOR UPDATE` or `LOCK IN SHARE MODE`), `UPDATE`, and `DELETE` statements, locking depends on whether the statement uses a unique index with a unique search condition, or a range-type search condition. For a unique index with a unique search condition, `InnoDB` locks only the index record found, not the gap before it. For other search conditions, `InnoDB` locks the index range scanned, using gap locks or next-key locks to block insertions by other sessions into the gaps covered by the range.

- `READ COMMITTED`

  A somewhat Oracle-like isolation level with respect to consistent (nonlocking) reads: Each consistent read, even within the same transaction, sets and reads its own fresh snapshot. See Section 14.2.2.4, “Consistent Nonlocking Reads”.

  For locking reads (`SELECT` with `FOR UPDATE` or `LOCK IN SHARE MODE`), `UPDATE` statements, and `DELETE` statements, `InnoDB` locks only index records, not the gaps before them, and thus permits the free insertion of new records next to locked records.

  > **Note**
  >
  > In MySQL 5.7, when `READ COMMITTED` isolation level is used, or the deprecated `innodb_locks_unsafe_for_binlog` system variable is enabled, there is no `InnoDB` gap locking except for foreign-key constraint checking and duplicate-key checking. Also, record locks for nonmatching rows are released after MySQL has evaluated the `WHERE` condition.
  >
  > If you use `READ COMMITTED` or enable `innodb_locks_unsafe_for_binlog`, you *must* use row-based binary logging.

- `READ UNCOMMITTED`

  `SELECT` statements are performed in a nonlocking fashion, but a possible earlier version of a row might be used. Thus, using this isolation level, such reads are not consistent. This is also called a dirty read. Otherwise, this isolation level works like `READ COMMITTED`.

- `SERIALIZABLE`

  This level is like `REPEATABLE READ`, but `InnoDB` implicitly converts all plain `SELECT` statements to `SELECT ... LOCK IN SHARE MODE` if autocommit is disabled. If autocommit is enabled, the `SELECT` is its own transaction. It therefore is known to be read only and can be serialized if performed as a consistent (nonlocking) read and need not block for other transactions. (To force a plain `SELECT` to block if other transactions have modified the selected rows, disable autocommit.)

Transaction Access Mode

The transaction access mode may be specified with `SET TRANSACTION`. By default, a transaction takes place in read/write mode, with both reads and writes permitted to tables used in the transaction. This mode may be specified explicitly using an access mode of `READ WRITE`.

If the transaction access mode is set to `READ ONLY`, changes to tables are prohibited. This may enable storage engines to make performance improvements that are possible when writes are not permitted.

It is not permitted to specify both `READ WRITE` and `READ ONLY` in the same statement.

In read-only mode, it remains possible to change tables created with the TEMPORARY keyword using DML statements. Changes made with DDL statements are not permitted, just as with permanent tables.

The READ WRITE and READ ONLY access modes also may be specified for an individual transaction using the START TRANSACTION statement.

13.3.7 XA Transactions

Support for XA transactions is available for the InnoDB storage engine. The MySQL XA implementation is based on the X/Open CAE document *Distributed Transaction Processing: The XA Specification*. This document is published by The Open Group and available at http://www.opengroup.org/public/pubs/catalog/c193.htm. Limitations of the current XA implementation are described in Section E.6, “Restrictions on XA Transactions”.

On the client side, there are no special requirements. The XA interface to a MySQL server consists of SQL statements that begin with the XA keyword. MySQL client programs must be able to send SQL statements and to understand the semantics of the XA statement interface. They do not need be linked against a recent client library. Older client libraries also will work.

Currently, among the MySQL Connectors, MySQL Connector/J 5.0.0 and higher supports XA directly, by means of a class interface that handles the XA SQL statement interface for you.

XA supports distributed transactions, that is, the ability to permit multiple separate transactional resources to participate in a global transaction. Transactional resources often are RDBMSs but may be other kinds of resources.

A global transaction involves several actions that are transactional in themselves, but that all must either complete successfully as a group, or all be rolled back as a group. In essence, this extends ACID properties “up a level” so that multiple ACID transactions can be executed in concert as components of a global operation that also has ACID properties. (However, for a distributed transaction, you must use the SERIALIZABLE isolation level to achieve ACID properties. It is enough to use REPEATABLE READ for a nondistributed transaction, but not for a distributed transaction.)

Some examples of distributed transactions:

- An application may act as an integration tool that combines a messaging service with an RDBMS. The application makes sure that transactions dealing with message sending, retrieval, and processing that also involve a transactional database all happen in a global transaction. You can think of this as “transactional email.”

- An application performs actions that involve different database servers, such as a MySQL server and an Oracle server (or multiple MySQL servers), where actions that involve multiple servers must happen as part of a global transaction, rather than as separate transactions local to each server.

- A bank keeps account information in an RDBMS and distributes and receives money through automated teller machines (ATMs). It is necessary to ensure that ATM actions are correctly reflected in the accounts, but this cannot be done with the RDBMS alone. A global transaction manager integrates the ATM and database resources to ensure overall consistency of financial transactions.

Applications that use global transactions involve one or more Resource Managers and a Transaction Manager:

- A Resource Manager (RM) provides access to transactional resources. A database server is one kind of resource manager. It must be possible to either commit or roll back transactions managed by the RM.

- A Transaction Manager (TM) coordinates the transactions that are part of a global transaction. It communicates with the RMs that handle each of these transactions. The individual transactions within a global transaction are “branches” of the global transaction. Global transactions and their branches are identified by a naming scheme described later.

The MySQL implementation of XA MySQL enables a MySQL server to act as a Resource Manager that handles XA transactions within a global transaction. A client program that connects to the MySQL server acts as the Transaction Manager.

To carry out a global transaction, it is necessary to know which components are involved, and bring each component to a point when it can be committed or rolled back. Depending on what each component reports about its ability to succeed, they must all commit or roll back as an atomic group. That is, either all components must commit, or all components must roll back. To manage a global transaction, it is necessary to take into account that any component or the connecting network might fail.

The process for executing a global transaction uses two-phase commit (2PC). This takes place after the actions performed by the branches of the global transaction have been executed.

1. In the first phase, all branches are prepared. That is, they are told by the TM to get ready to commit. Typically, this means each RM that manages a branch records the actions for the branch in stable storage. The branches indicate whether they are able to do this, and these results are used for the second phase.

2. In the second phase, the TM tells the RMs whether to commit or roll back. If all branches indicated when they were prepared that they will be able to commit, all branches are told to commit. If any branch indicated when it was prepared that it will not be able to commit, all branches are told to roll back.

In some cases, a global transaction might use one-phase commit (1PC). For example, when a Transaction Manager finds that a global transaction consists of only one transactional resource (that is, a single branch), that resource can be told to prepare and commit at the same time.

13.3.7.1 XA Transaction SQL Syntax

To perform XA transactions in MySQL, use the following statements:

```
XA {START|BEGIN} xid [JOIN|RESUME]

XA END xid [SUSPEND [FOR MIGRATE]]

XA PREPARE xid

XA COMMIT xid [ONE PHASE]

XA ROLLBACK xid

XA RECOVER
```

For `XA START`, the `JOIN` and `RESUME` clauses are not supported.

For `XA END` the `SUSPEND [FOR MIGRATE]` clause is not supported.

Each XA statement begins with the `XA` keyword, and most of them require an *`xid`* value. An *`xid`* is an XA transaction identifier. It indicates which transaction the statement applies to. *`xid`* values are supplied by the client, or generated by the MySQL server. An *`xid`* value has from one to three parts:

```
xid: gtrid [, bqual [, formatID ]]
```

`gtrid` is a global transaction identifier, *`bqual`* is a branch qualifier, and *`formatID`* is a number that identifies the format used by the *`gtrid`* and *`bqual`* values. As indicated by the syntax, *`bqual`* and

formatID are optional. The default *bqual* value is '' if not given. The default *formatID* value is 1 if not given.

gtrid and *bqual* must be string literals, each up to 64 bytes (not characters) long. *gtrid* and *bqual* can be specified in several ways. You can use a quoted string ('ab'), hex string (0x6162, X'ab'), or bit value (b'*nnnn*').

formatID is an unsigned integer.

The *gtrid* and *bqual* values are interpreted in bytes by the MySQL server's underlying XA support routines. However, while an SQL statement containing an XA statement is being parsed, the server works with some specific character set. To be safe, write *gtrid* and *bqual* as hex strings.

xid values typically are generated by the Transaction Manager. Values generated by one TM must be different from values generated by other TMs. A given TM must be able to recognize its own *xid* values in a list of values returned by the XA RECOVER statement.

XA START *xid* starts an XA transaction with the given *xid* value. Each XA transaction must have a unique *xid* value, so the value must not currently be used by another XA transaction. Uniqueness is assessed using the *gtrid* and *bqual* values. All following XA statements for the XA transaction must be specified using the same *xid* value as that given in the XA START statement. If you use any of those statements but specify an *xid* value that does not correspond to some existing XA transaction, an error occurs.

One or more XA transactions can be part of the same global transaction. All XA transactions within a given global transaction must use the same *gtrid* value in the *xid* value. For this reason, *gtrid* values must be globally unique so that there is no ambiguity about which global transaction a given XA transaction is part of. The *bqual* part of the *xid* value must be different for each XA transaction within a global transaction. (The requirement that *bqual* values be different is a limitation of the current MySQL XA implementation. It is not part of the XA specification.)

The XA RECOVER statement returns information for those XA transactions on the MySQL server that are in the PREPARED state. (See Section 13.3.7.2, “XA Transaction States”.) The output includes a row for each such XA transaction on the server, regardless of which client started it.

XA RECOVER output rows look like this (for an example *xid* value consisting of the parts 'abc', 'def', and 7):

```
mysql> XA RECOVER;
+----------+--------------+--------------+--------+
| formatID | gtrid_length | bqual_length | data   |
+----------+--------------+--------------+--------+
|        7 |            3 |            3 | abcdef |
+----------+--------------+--------------+--------+
```

The output columns have the following meanings:

- formatID is the *formatID* part of the transaction *xid*
- gtrid_length is the length in bytes of the *gtrid* part of the *xid*
- bqual_length is the length in bytes of the *bqual* part of the *xid*
- data is the concatenation of the *gtrid* and *bqual* parts of the *xid*

13.3.7.2 XA Transaction States

An XA transaction progresses through the following states:

1. Use XA START to start an XA transaction and put it in the ACTIVE state.
2. For an ACTIVE XA transaction, issue the SQL statements that make up the transaction, and then issue an XA END statement. XA END puts the transaction in the IDLE state.
3. For an IDLE XA transaction, you can issue either an XA PREPARE statement or an XA COMMIT ... ONE PHASE statement:
   - XA PREPARE puts the transaction in the PREPARED state. An XA RECOVER statement at this point will include the transaction's *xid* value in its output, because XA RECOVER lists all XA transactions that are in the PREPARED state.
   - XA COMMIT ... ONE PHASE prepares and commits the transaction. The *xid* value will not be listed by XA RECOVER because the transaction terminates.
4. For a PREPARED XA transaction, you can issue an XA COMMIT statement to commit and terminate the transaction, or XA ROLLBACK to roll back and terminate the transaction.

Here is a simple XA transaction that inserts a row into a table as part of a global transaction:

```
mysql> XA START 'xatest';
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO mytable (i) VALUES(10);
Query OK, 1 row affected (0.04 sec)

mysql> XA END 'xatest';
Query OK, 0 rows affected (0.00 sec)

mysql> XA PREPARE 'xatest';
Query OK, 0 rows affected (0.00 sec)

mysql> XA COMMIT 'xatest';
Query OK, 0 rows affected (0.00 sec)
```

Within the context of a given client connection, XA transactions and local (non-XA) transactions are mutually exclusive. For example, if XA START has been issued to begin an XA transaction, a local transaction cannot be started until the XA transaction has been committed or rolled back. Conversely, if a local transaction has been started with START TRANSACTION, no XA statements can be used until the transaction has been committed or rolled back.

Note that if an XA transaction is in the ACTIVE state, you cannot issue any statements that cause an implicit commit. That would violate the XA contract because you could not roll back the XA transaction. You will receive the following error if you try to execute such a statement:

```
ERROR 1399 (XAE07): XAER_RMFAIL: The command cannot be executed
when global transaction is in the ACTIVE state
```

Statements to which the preceding remark applies are listed at Section 13.3.3, “Statements That Cause an Implicit Commit”.

13.4 Replication Statements

Replication can be controlled through the SQL interface using the statements described in this section. One group of statements controls master servers, the other controls slave servers.

13.4.1 SQL Statements for Controlling Master Servers

This section discusses statements for managing master replication servers. Section 13.4.2, "SQL Statements for Controlling Slave Servers", discusses statements for managing slave servers.

In addition to the statements described here, the following `SHOW` statements are used with master servers in replication. For information about these statements, see Section 13.7.5, "`SHOW` Syntax".

- `SHOW BINARY LOGS`
- `SHOW BINLOG EVENTS`
- `SHOW MASTER STATUS`
- `SHOW SLAVE HOSTS`

13.4.1.1 `PURGE BINARY LOGS` Syntax

```
PURGE { BINARY | MASTER } LOGS
    { TO 'log_name' | BEFORE datetime_expr }
```

The binary log is a set of files that contain information about data modifications made by the MySQL server. The log consists of a set of binary log files, plus an index file (see Section 5.2.4, "The Binary Log").

The `PURGE BINARY LOGS` statement deletes all the binary log files listed in the log index file prior to the specified log file name or date. `BINARY` and `MASTER` are synonyms. Deleted log files also are removed from the list recorded in the index file, so that the given log file becomes the first in the list.

This statement has no effect if the server was not started with the `--log-bin` option to enable binary logging.

Examples:

```
PURGE BINARY LOGS TO 'mysql-bin.010';
PURGE BINARY LOGS BEFORE '2008-04-02 22:46:26';
```

The `BEFORE` variant's *`datetime_expr`* argument should evaluate to a `DATETIME` value (a value in `'YYYY-MM-DD hh:mm:ss'` format).

This statement is safe to run while slaves are replicating. You need not stop them. If you have an active slave that currently is reading one of the log files you are trying to delete, this statement does nothing. In MySQL 5.7.2 and later, it fails with an error in such cases. (Bug #13727933) However, if a slave is not connected and you happen to purge one of the log files it has yet to read, the slave will be unable to replicate after it reconnects.

To safely purge binary log files, follow this procedure:

1. On each slave server, use `SHOW SLAVE STATUS` to check which log file it is reading.
2. Obtain a listing of the binary log files on the master server with `SHOW BINARY LOGS`.
3. Determine the earliest log file among all the slaves. This is the target file. If all the slaves are up to date, this is the last log file on the list.
4. Make a backup of all the log files you are about to delete. (This step is optional, but always advisable.)
5. Purge all log files up to but not including the target file.

You can also set the expire_logs_days system variable to expire binary log files automatically after a given number of days (see Section 5.1.4, “Server System Variables”). If you are using replication, you should set the variable no lower than the maximum number of days your slaves might lag behind the master.

PURGE BINARY LOGS TO and PURGE BINARY LOGS BEFORE both fail with an error when binary log files listed in the .index file had been removed from the system by some other means (such as using rm on Linux). (Bug #18199, Bug #18453) To handle such errors, edit the .index file (which is a simple text file) manually to ensure that it lists only the binary log files that are actually present, then run again the PURGE BINARY LOGS statement that failed.

13.4.1.2 RESET MASTER Syntax

```
RESET MASTER
```

Deletes all binary log files listed in the index file, resets the binary log index file to be empty, and creates a new binary log file.

RESET MASTER also clears the values of the gtid_purged system variable as well as the global value of the gtid_executed system variable (but not its session value); that is, executing this statement sets each of these values to an empty string ('').

This statement is intended to be used only when the master is started for the first time.

⚠

Important

The effects of RESET MASTER differ from those of PURGE BINARY LOGS in 2 key ways:

1. RESET MASTER removes *all* binary log files that are listed in the index file, leaving only a single, empty binary log file with a numeric suffix of .000001, whereas the numbering is not reset by PURGE BINARY LOGS.

2. RESET MASTER is *not* intended to be used while any replication slaves are running. The behavior of RESET MASTER when used while slaves are running is undefined (and thus unsupported), whereas PURGE BINARY LOGS may be safely used while replication slaves are running.

See also Section 13.4.1.1, “PURGE BINARY LOGS Syntax”.

RESET MASTER can prove useful when you first set up the master and the slave, so that you can verify the setup as follows:

1. Start the master and slave, and start replication (see Section 16.1.1, “How to Set Up Replication”).

2. Execute a few test queries on the master.

3. Check that the queries were replicated to the slave.

4. When replication is running correctly, issue STOP SLAVE followed by RESET SLAVE on the slave, then verify that any unwanted data no longer exists on the slave.

5. Issue RESET MASTER on the master to clean up the test queries.

After verifying the setup and getting rid of any unwanted and log files generated by testing, you can start the slave and begin replicating.

13.4.1.3 SET sql_log_bin Syntax

```
SET sql_log_bin = {0|1}
```

The sql_log_bin variable controls whether logging to the binary log is done. The default value is 1 (do logging). To change logging for the current session, change the session value of this variable. The session user must have the SUPER privilege to set this variable.

In MySQL 5.7, it is not possible to set @@session.sql_log_bin within a transaction or subquery. (Bug #53437)

13.4.2 SQL Statements for Controlling Slave Servers

This section discusses statements for managing slave replication servers. Section 13.4.1, "SQL Statements for Controlling Master Servers", discusses statements for managing master servers.

In addition to the statements described here, SHOW SLAVE STATUS and SHOW RELAYLOG EVENTS are also used with replication slaves. For information about these statements, see Section 13.7.5.33, "SHOW SLAVE STATUS Syntax", and Section 13.7.5.31, "SHOW RELAYLOG EVENTS Syntax".

13.4.2.1 CHANGE MASTER TO Syntax

```
CHANGE MASTER TO option [, option] ...

option:
    MASTER_BIND = 'interface_name'
  | MASTER_HOST = 'host_name'
  | MASTER_USER = 'user_name'
  | MASTER_PASSWORD = 'password'
  | MASTER_PORT = port_num
  | MASTER_CONNECT_RETRY = interval
  | MASTER_RETRY_COUNT = count
  | MASTER_DELAY = interval
  | MASTER_HEARTBEAT_PERIOD = interval
  | MASTER_LOG_FILE = 'master_log_name'
  | MASTER_LOG_POS = master_log_pos
  | MASTER_AUTO_POSITION = {0|1}
  | RELAY_LOG_FILE = 'relay_log_name'
  | RELAY_LOG_POS = relay_log_pos
  | MASTER_SSL = {0|1}
  | MASTER_SSL_CA = 'ca_file_name'
  | MASTER_SSL_CAPATH = 'ca_directory_name'
  | MASTER_SSL_CERT = 'cert_file_name'
  | MASTER_SSL_CRL = 'crl_file_name'
  | MASTER_SSL_CRLPATH = 'crl_directory_name'
  | MASTER_SSL_KEY = 'key_file_name'
  | MASTER_SSL_CIPHER = 'cipher_list'
  | MASTER_SSL_VERIFY_SERVER_CERT = {0|1}
  | IGNORE_SERVER_IDS = (server_id_list)

server_id_list:
    [server_id [, server_id] ... ]
```

CHANGE MASTER TO changes the parameters that the slave server uses for connecting to the master server, for reading the master binary log, and reading the slave relay log. It also updates the contents of the master info and relay log info repositories (see Section 16.2.2, "Replication Relay and Status Logs"). In MySQL 5.7.1 and later, gtid_next must be set to AUTOMATIC before you can use CHANGE MASTER TO (Bug #16062608).

Priot to MySQL 5.7.4, the slave replication threads must be stopped, using STOP SLAVE if necessary, before issuing this statement. In MySQL 5.7.4 and later, you can issue CHANGE MASTER TO statements on a running slave without doing this, depending on the states of the slave SQL thread and slave I/O thread. The rules governing such use are provided later in this section.

Options not specified retain their value, except as indicated in the following discussion. Thus, in most cases, there is no need to specify options that do not change. For example, in MySQL 5.7.3 and earlier, if the password to connect to your MySQL master has changed, you just need to issue these statements to tell the slave about the new password:

```
STOP SLAVE; -- if replication was running
CHANGE MASTER TO MASTER_PASSWORD='new3cret';
START SLAVE; -- if you want to restart replication
```

In MySQL 5.7.4 and later, the STOP SLAVE and START SLAVE statements are not needed with the CHANGE MASTER TO statement just shown if the I/O thread is stopped, as discussed later in this section.

MASTER_HOST, MASTER_USER, MASTER_PASSWORD, and MASTER_PORT provide information to the slave about how to connect to its master:

- MASTER_HOST and MASTER_PORT are the host name (or IP address) of the master host and its TCP/IP port.

  Note

  Replication cannot use Unix socket files. You must be able to connect to the master MySQL server using TCP/IP.

  If you specify the MASTER_HOST or MASTER_PORT option, the slave assumes that the master server is different from before (even if the option value is the same as its current value.) In this case, the old values for the master binary log file name and position are considered no longer applicable, so if you do not specify MASTER_LOG_FILE and MASTER_LOG_POS in the statement, MASTER_LOG_FILE='' and MASTER_LOG_POS=4 are silently appended to it.

  Setting MASTER_HOST='' (that is, setting its value explicitly to an empty string) is *not* the same as not setting MASTER_HOST at all. Beginning with MySQL 5.5, trying to set MASTER_HOST to an empty string fails with an error. Previously, setting MASTER_HOST to an empty string caused START SLAVE subsequently to fail. (Bug #28796)

  Values used for MASTER_HOST and other CHANGE MASTER TO options are checked for linefeed (\n or 0x0A) characters; the presence of such characters in these values causes the statement to fail with **ER_MASTER_INFO**. (Bug #11758581, Bug #50801)

- MASTER_USER and MASTER_PASSWORD are the user name and password of the account to use for connecting to the master.

  MASTER_USER cannot be made empty; setting MASTER_USER = '' or leaving it unset when setting a value for for MASTER_PASSWORD causes an error (Bug #13427949).

  Currently, a password used for a replication slave account is effectively limited to 32 characters in length; if the password is longer, any excess characters are truncated. This is not due to any limit imposed by the MySQL Server generally, but rather is an issue specific to MySQL Replication. (For more information, see Bug #43439.)

  The text of a running CHANGE MASTER TO statement, including values for MASTER_USER and MASTER_PASSWORD, can be seen in the output of a concurrent SHOW PROCESSLIST statement. (The complete text of a START SLAVE statement is also visible to SHOW PROCESSLIST.)

The MASTER_SSL_*xxx* options provide information about using SSL for the connection. They correspond to the --ssl-*xxx* options described in Section 6.3.11.4, “SSL Command Options”, and Section 16.3.7, “Setting Up Replication Using SSL”. These options can be changed even on slaves that are compiled without SSL support. They are saved to the master info repository, but are ignored if the slave does not have SSL support enabled.

As of MySQL 5.7.3, the MASTER_SSL=1 is prescriptive, not advisory. When given, the slave connection to the master must use SSL or the connection attempt fails. Before 5.7.3, an SSL connection is permitted but not required. This is analogous to the client-side meaning of the --ssl command-line option; see Section 6.3.11.4, “SSL Command Options”.

MASTER_CONNECT_RETRY specifies how many seconds to wait between connect retries. The default is 60.

MASTER_RETRY_COUNT limits the *number* of reconnection attempts and updates the value of the Master_Retry_Count column in the output of SHOW SLAVE STATUS. The default value is 24 * 3600 = 86400. MASTER_RETRY_COUNT is intended to replace the older --master-retry-count server option, and is now the preferred method for setting this limit. You are encouraged not to rely on --master-retry-count in new applications and, when upgrading to MySQL 5.7, to update any existing applications that rely on it, so that they use CHANGE MASTER TO ... MASTER_RETRY_COUNT instead.

MASTER_DELAY specifies how many seconds behind the master the slave must lag. An event received from the master is not executed until at least *interval* seconds later than its execution on the master. The default is 0. An error occurs if *interval* is not a nonnegative integer in the range from 0 to 2^{31}–1. For more information, see Section 16.3.9, “Delayed Replication”.

In MySQL 5.7.4 and later, a CHANGE MASTER TO statement employing the MASTER_DELAY option can be executed on a running slave when the slave SQL thread is stopped.

MASTER_BIND is for use on replication slaves having multiple network interfaces, and determines which of the slave's network interfaces is chosen for connecting to the master.

The address configured with this option, if any, can be seen in the Master_Bind column of the output from SHOW SLAVE STATUS. If you are using slave status log tables (server started with --master-info-repository=TABLE), the value can also be seen as the Master_bind column of the mysql.slave_master_info table.

MASTER_HEARTBEAT_PERIOD sets the interval in seconds between replication heartbeats. Whenever the master's binary log is updated with an event, the waiting period for the next heartbeat is reset. *interval* is a decimal value having the range 0 to 4294967 seconds and a resolution in milliseconds; the smallest nonzero value is 0.001. Heartbeats are sent by the master only if there are no unsent events in the binary log file for a period longer than *interval*.

Prior to MySQL 5.7.4, not including MASTER_HEARTBEAT_PERIOD caused CHANGE MASTER TO to reset the heartbeat period (Slave_heartbeat_period) to the default, and Slave_received_heartbeats to 0. (Bug #18185490)

If you are logging master connection information to tables, MASTER_HEARTBEAT_PERIOD can be seen as the value of the Heartbeat column of the mysql.slave_master_info table.

Setting *interval* to 0 disables heartbeats altogether. The default value for *interval* is equal to the value of slave_net_timeout divided by 2.

Setting @@global.slave_net_timeout to a value less than that of the current heartbeat interval results in a warning being issued. The effect of issuing RESET SLAVE on the heartbeat interval is to reset it to the default value.

MASTER_LOG_FILE and MASTER_LOG_POS are the coordinates at which the slave I/O thread should begin reading from the master the next time the thread starts. RELAY_LOG_FILE and RELAY_LOG_POS are the coordinates at which the slave SQL thread should begin reading from the relay log the next time the thread starts. If you specify either of MASTER_LOG_FILE or MASTER_LOG_POS, you cannot specify RELAY_LOG_FILE or RELAY_LOG_POS. If you specify either of MASTER_LOG_FILE or MASTER_LOG_POS, you also cannot specify MASTER_AUTO_POSITION = 1 (described later in this section). If neither of MASTER_LOG_FILE or MASTER_LOG_POS is specified, the slave uses the last coordinates of the *slave SQL thread* before CHANGE MASTER TO was issued. This ensures that there is no discontinuity in replication, even if the slave SQL thread was late compared to the slave I/O thread, when you merely want to change, say, the password to use.

In MySQL 5.7.4 and later, a CHANGE MASTER TO statement employing RELAY_LOG_FILE, RELAY_LOG_POS, or both options can be executed on a running slave when the slave SQL thread is stopped.

If MASTER_AUTO_POSITION = 1 is used with CHANGE MASTER TO, the slave attempts to connect to the master using the GTID-based replication protocol. In MySQL 5.7.4 and later, this option can be employed by CHANGE MASTER TO only if both the slave SQL and slave I/O threads are stopped.

When using GTIDs, the slave tells the master which transactions it has already received, executed, or both. To compute this set, it reads the global value of gtid_executed and the value of the Retrieved_gtid_set column from SHOW SLAVE STATUS. Since the GTID of the last transmitted transaction is included in Retrieved_gtid_set even if the transaction was only partially transmitted, the last received GTID is subtracted from this set. Thus, the slave computes the following set:

```
UNION(@@global.gtid_executed, Retrieved_gtid_set - last_received_GTID)
```

This set is sent to the master as part of the initial handshake, and the master sends back all transactions that it has executed which are not part of the set. If any of these transactions have been already purged from the master's binary log, the master sends the error **ER_MASTER_HAS_PURGED_REQUIRED_GTIDS** to the slave, and replication does not start.

When GTID-based replication is employed, the coordinates represented by MASTER_LOG_FILE and MASTER_LOG_POS are not used, and global transaction identifiers are used instead. Thus the use of either or both of these options together with MASTER_AUTO_POSITION causes an error.

Beginning with MySQL 5.7.1, you can see whether replication is running with autopositioning enabled by checking the output of SHOW SLAVE STATUS. (Bug #15992220)

gtid_mode must also be enabled before issuing CHANGE MASTER TO ... MASTER_AUTO_POSITION = 1. Otherwise, the statement fails with an error.

To revert to the older file-based replication protocol after using GTIDs, you can issue a new CHANGE MASTER TO statement that specifies MASTER_AUTO_POSITION = 0, as well as at least one of MASTER_LOG_FILE or MASTER_LOG_POSITION.

Prior to MySQL 5.7.4, CHANGE MASTER TO deletes all relay log files and starts a new one, unless you specify RELAY_LOG_FILE or RELAY_LOG_POS. In that case, relay log files are kept; the relay_log_purge global variable is set silently to 0. In MySQL 5.7.4 and later, relay logs are preserved when neither the slave SQL thread nor the slave I/O thread is stopped; if both threads are stopped, all relay log files are deleted unless you at least one of RELAY_LOG_FILE or RELAY_LOG_POS is specified.

RELAY_LOG_FILE can use either an absolute or relative path, and uses the same basename as MASTER_LOG_FILE. (Bug #12190)

IGNORE_SERVER_IDS takes a comma-separated list of 0 or more server IDs. Events originating from the corresponding servers are ignored, with the exception of log rotation and deletion events, which are still recorded in the relay log.

In circular replication, the originating server normally acts as the terminator of its own events, so that they are not applied more than once. Thus, this option is useful in circular replication when one of the servers in the circle is removed. Suppose that you have a circular replication setup with 4 servers, having server IDs 1, 2, 3, and 4, and server 3 fails. When bridging the gap by starting replication from server 2 to server 4, you can include IGNORE_SERVER_IDS = (3) in the CHANGE MASTER TO statement that you issue on server 4 to tell it to use server 2 as its master instead of server 3. Doing so causes it to ignore and not to propagate any statements that originated with the server that is no longer in use.

If a CHANGE MASTER TO statement is issued without any IGNORE_SERVER_IDS option, any existing list is preserved; RESET SLAVE also has no effect on the server ID list. To clear the list of ignored servers, it is necessary to use the option with an empty list:

```
CHANGE MASTER TO IGNORE_SERVER_IDS = ();
```

If IGNORE_SERVER_IDS contains the server's own ID and the server was started with the --replicate-same-server-id option enabled, an error results.

In MySQL 5.7, the master info repository and the output of SHOW SLAVE STATUS provide the list of servers that are currently ignored. For more information, see Section 16.2.2.2, “Slave Status Logs”, and Section 13.7.5.33, “SHOW SLAVE STATUS Syntax”.

In MySQL 5.7, invoking CHANGE MASTER TO causes the previous values for MASTER_HOST, MASTER_PORT, MASTER_LOG_FILE, and MASTER_LOG_POS to be written to the error log, along with other information about the slave's state prior to execution.

In MySQL 5.7, CHANGE MASTER TO causes an implicit commit of an ongoing transaction. See Section 13.3.3, “Statements That Cause an Implicit Commit”.

In MySQL 5.7.4 and later, the strict requirement to execute STOP SLAVE prior to issuing any CHANGE MASTER TO statement (and START SLAVE afterward) is removed. Instead of depending on whether the slave is stopped, the behavior of CHANGE MASTER TO depends (in MySL 5.7.4 and later) on the states of the slave SQL thread and slave I/O threads; which of these threads is stopped or running now determines the options that can or cannot be used with a CHANGE MASTER TO statement at a given point in time. The rules for making this determination are listed here:

- If the SQL thread is stopped, you can execute CHANGE MASTER TO using any combination that is otherwise allowed of RELAY_LOG_FILE, RELAY_LOG_POS, and MASTER_DELAY options, even if the slave I/O thread is running. No other options may be used with this statement when the I/O thread is running.
- If the I/O thread is stopped, you can execute CHANGE MASTER TO using any of the options for this statement (in any allowed combination) *except* RELAY_LOG_FILE, RELAY_LOG_POS, or MASTER_DELAY, even when the SQL thread is running. These three options may not be used when the I/O thread is running.
- Both the SQL thread and the I/O thread must be stopped before issuing a CHANGE MASTER TO statement that employs MASTER_AUTO_POSITION = 1.

You can check the current state of the slave SQL and I/O threads using SHOW SLAVE STATUS.

For more information, see Section 16.3.6, “Switching Masters During Failover”.

If you are using statement-based replication and temporary tables, it is possible for a CHANGE MASTER TO statement following a STOP SLAVE statement to leave behind temporary tables on the slave. In MySQL 5.7.4 and later, a warning (**ER_WARN_OPEN_TEMP_TABLES_MUST_BE_ZERO**) is now issued whenever this occurs. You can avoid this in such cases by making sure that the value of the Slave_open_temp_tables system status variable is equal to 0 prior to executing such a CHANGE MASTER TO statement.

CHANGE MASTER TO is useful for setting up a slave when you have the snapshot of the master and have recorded the master binary log coordinates corresponding to the time of the snapshot. After loading the snapshot into the slave to synchronize it with the master, you can run CHANGE MASTER TO MASTER_LOG_FILE='*log_name*', MASTER_LOG_POS=*log_pos* on the slave to specify the coordinates at which the slave should begin reading the master binary log.

The following example changes the master server the slave uses and establishes the master binary log coordinates from which the slave begins reading. This is used when you want to set up the slave to replicate the master:

```
CHANGE MASTER TO
  MASTER_HOST='master2.mycompany.com',
  MASTER_USER='replication',
  MASTER_PASSWORD='bigs3cret',
  MASTER_PORT=3306,
  MASTER_LOG_FILE='master2-bin.001',
  MASTER_LOG_POS=4,
  MASTER_CONNECT_RETRY=10;
```

The next example shows an operation that is less frequently employed. It is used when the slave has relay log files that you want it to execute again for some reason. To do this, the master need not be reachable. You need only use CHANGE MASTER TO and start the SQL thread (START SLAVE SQL_THREAD):

```
CHANGE MASTER TO
  RELAY_LOG_FILE='slave-relay-bin.006',
  RELAY_LOG_POS=4025;
```

You can even use the second operation in a nonreplication setup with a standalone, nonslave server for recovery following a crash. Suppose that your server has crashed and you have restored it from a backup. You want to replay the server's own binary log files (not relay log files, but regular binary log files), named (for example) myhost-bin.*. First, make a backup copy of these binary log files in some safe place, in case you don't exactly follow the procedure below and accidentally have the server purge the binary log. Use SET GLOBAL relay_log_purge=0 for additional safety. Then start the server without the --log-bin option, Instead, use the --replicate-same-server-id, --relay-log=myhost-bin (to make the server believe that these regular binary log files are relay log files) and --skip-slave-start options. After the server starts, issue these statements:

```
CHANGE MASTER TO
  RELAY_LOG_FILE='myhost-bin.153',
  RELAY_LOG_POS=410,
  MASTER_HOST='some_dummy_string';
START SLAVE SQL_THREAD;
```

The server reads and executes its own binary log files, thus achieving crash recovery. Once the recovery is finished, run STOP SLAVE, shut down the server, clear the master info and relay log info repositories, and restart the server with its original options.

Specifying the MASTER_HOST option (even with a dummy value) is required to make the server think it is a slave.

The following table shows the maximum permissible length for the string-valued options.

Option	Maximum Length
MASTER_HOST	60
MASTER_USER	16
MASTER_PASSWORD	32
MASTER_LOG_FILE	255
RELAY_LOG_FILE	255
MASTER_SSL_CA	255
MASTER_SSL_CAPATH	255
MASTER_SSL_CERT	255
MASTER_SSL_CRL	255
MASTER_SSL_CRLPATH	255
MASTER_SSL_KEY	255
MASTER_SSL_CIPHER	511

13.4.2.2 CHANGE REPLICATION FILTER Syntax

```
CHANGE REPLICATION FILTER filter[, filter][, ...]

filter:
    REPLICATE_DO_DB = (db_list)
  | REPLICATE_IGNORE_DB = (db_list)
  | REPLICATE_DO_TABLE = (tbl_list)
  | REPLICATE_IGNORE_TABLE = (tbl_list)
  | REPLICATE_WILD_DO_TABLE = (wild_tbl_list)
  | REPLICATE_WILD_IGNORE_TABLE = (wild_tbl_list)
  | REPLICATE_REWRITE_DB = (db_pair_list)

db_list:
    db_name[, db_name][, ...]

tbl_list:
    tbl_name[, tbl_name][, ...]

wild_tbl_list:
    'pattern'[, 'pattern'][, ...]

db_pair_list:
    (db_pair)[, (db_pair)][, ...]

db_pair:
    from_db, to_db
```

In MySQL 5.7.3 and later, CHANGE REPLICATION FILTER sets one or more replication filtering rules on the slave in the same way as starting the slave mysqld with replication filtering options such as --replicate-do-db or --replicate-wild-ignore-table. Unlike the case with the server options, this statement does not require restarting the server to take effect, only that the slave SQL thread be stopped using STOP SLAVE SQL_THREAD first (and restarted with START SLAVE SQL_THREAD afterwards).

CHANGE REPLICATION FILTER options have the effects described and relate to --replicate-* server options shown in the following list:

- REPLICATE_DO_DB: Include updates based on database name. Equivalent to --replicate-do-db.
- REPLICATE_IGNORE_DB: Exclude updates based on database name. Equivalent to --replicate-ignore-db.
- REPLICATE_DO_TABLE: Include updates based on table name. Equivalent to --replicate-do-table.
- REPLICATE_IGNORE_TABLE: Exclude updates based on table name. Equivalent to --replicate-ignore-table.
- REPLICATE_WILD_DO_TABLE: Include updates based on wildcard pattern matching table name. Equivalent to --replicate-wild-do-table.
- REPLICATE_WILD_IGNORE_TABLE: Exclude updates based on wildcard pattern matching table name. Equivalent to --replicate-wild-ignore-table.
- REPLICATE_REWRITE_DB: Perform updates on slave after substituting new name on slave for specified database on master. Equivalent to --replicate-rewrite-db.

The precise effects of REPLICATE_DO_DB and REPLICATE_IGNORE_DB filters are dependent on whether statement-based or row-based replication is in effect. See Section 16.2.3, “How Servers Evaluate Replication Filtering Rules”, for more information.

Mutliple replication filtering rules can be created in a single CHANGE REPLICATION FILTER statement by separating the rules with commas, as shown here:

```
CHANGE REPLICATION FILTER
    REPLICATE_DO_DB = (d1), REPLICATE_IGNORE_DB = (d2);
```

Issuing the statement just shown is equivalent to starting the slave mysqld with the options --replicate-do-db=d1 --replicate-ignore-db=d2.

If the same filtering rule is specified multiple times, only the *last* such rule is actually used. For example, the two statements shown here have exactly the same effect, because the first REPLICATE_DO_DB rule in the first statement is ignored:

```
CHANGE REPLICATION FILTER
    REPLICATION_DO_DB = (db1, db2), REPLICATE_DO_DB = (db3, db4);

CHANGE REPLICATION FILTER
    REPLICATE_DO_DB = (db3,db4);
```

Caution

This behavior differs from that of the --replicate-* filter options where specifying the same option multiple times causes the creation of multiple filter rules.

Names of tables and database not containing any special characters need not be quoted. Values used with REPLICATION_WILD_TABLE and REPLICATION_WILD_IGNORE_TABLE are string expressions, possibly containing (special) wildcard characters, and so must be quoted. This is shown in the following example statements:

```
CHANGE REPLICATION FILTER
    REPLICATE_WILD_DO_TABLE = ('db1.old%');
```

```
CHANGE REPLICATION FILTER
    REPLICATE_WILD_IGNORE_TABLE = ('db1.new%', 'db2.new*');
```

Values used with REPLICATE_REWRITE_DB represent *pairs* of database names; each such value must be enclosed in parentheses. The following statement rewrites statements occurring on database dbA on the master to database dbB on the slave:

```
CHANGE REPLICATION FILTER REPLICATE_REWRITE_DB = ((db1, db2));
```

The statement just shown contains two sets of parentheses, one enclosing the pair of database names, and the other enclosing the entire list. This is perhap more easily seen in the following example, which creates two rewrite-db rules, one rewriting database dbA to dbB, and one rewriting database dbC to dbD:

```
CHANGE REPLICATION FILTER
  REPLICATE_REWRITE_DB = ((dbA, dbB), (dbC, dbD));
```

This statement leaves any existing replication filtering rules unchanged; to unset all filters of a given type, set the filter's value to an explicitly empty list, as shown in this example, which removes all existing REPLICATE_DO_DB and REPLICATE_IGNORE_DB rules:

```
CHANGE REPLICATION FILTER
    REPLICATE_DO_DB = (), REPLICATE_IGNORE_DB = ();
```

Setting a filter to empty in this way removes all existing rules, does not create any new ones, and does not restore any rules set at mysqld startup using --replicate-* options on the command line or in the configuration file.

For more information, see Section 16.2.3, “How Servers Evaluate Replication Filtering Rules”.

13.4.2.3 MASTER_POS_WAIT() Syntax

```
SELECT MASTER_POS_WAIT('master_log_file', master_log_pos [, timeout])
```

This is actually a function, not a statement. It is used to ensure that the slave has read and executed events up to a given position in the master's binary log. See Section 12.16, “Miscellaneous Functions”, for a full description.

13.4.2.4 RESET SLAVE Syntax

```
RESET SLAVE [ALL]
```

RESET SLAVE makes the slave forget its replication position in the master's binary log. This statement is meant to be used for a clean start: It clears the master info and relay log info repositories, deletes all the relay log files, and starts a new relay log file. It also resets to 0 the replication delay specified with the MASTER_DELAY option to CHANGE MASTER TO. To use RESET SLAVE, the slave replication threads must be stopped (use STOP SLAVE if necessary).

Note

All relay log files are deleted, even if they have not been completely executed by the slave SQL thread. (This is a condition likely to exist on a replication slave if you have issued a STOP SLAVE statement or if the slave is highly loaded.)

In MySQL 5.7 (unlike the case in MySQL 5.1 and earlier), `RESET SLAVE` does not change any replication connection parameters such as master host, master port, master user, or master password, which are retained in memory. This means that `START SLAVE` can be issued without requiring a `CHANGE MASTER TO` statement following `RESET SLAVE`.

Connection parameters are reset by `RESET SLAVE ALL`. (`RESET SLAVE` followed by a restart of the slave `mysqld` also does this.)

In MySQL 5.7, `RESET SLAVE` causes an implicit commit of an ongoing transaction. See Section 13.3.3, "Statements That Cause an Implicit Commit".

If the slave SQL thread was in the middle of replicating temporary tables when it was stopped, and `RESET SLAVE` is issued, these replicated temporary tables are deleted on the slave.

13.4.2.5 `SET GLOBAL sql_slave_skip_counter` Syntax

```
SET GLOBAL sql_slave_skip_counter = N
```

This statement skips the next *`N`* events from the master. This is useful for recovering from replication stops caused by a statement.

This statement is valid only when the slave threads are not running. Otherwise, it produces an error.

When using this statement, it is important to understand that the binary log is actually organized as a sequence of groups known as *event groups*. Each event group consists of a sequence of events.

- For transactional tables, an event group corresponds to a transaction.
- For nontransactional tables, an event group corresponds to a single SQL statement.

> **Note**
>
> A single transaction can contain changes to both transactional and nontransactional tables.

When you use `SET GLOBAL sql_slave_skip_counter` to skip events and the result is in the middle of a group, the slave continues to skip events until it reaches the end of the group. Execution then starts with the next event group.

In MySQL 5.7, issuing this statement causes the previous values of `RELAY_LOG_FILE`, `RELAY_LOG_POS`, and `sql_slave_skip_counter` to be written to the error log.

13.4.2.6 `START SLAVE` Syntax

```
START SLAVE [thread_types] [until_option] [connection_options]

thread_types:
    [thread_type [, thread_type] ... ]

thread_type:
    IO_THREAD | SQL_THREAD

until_option:
    UNTIL {   {SQL_BEFORE_GTIDS | SQL_AFTER_GTIDS} = gtid_set
          |   MASTER_LOG_FILE = 'log_name', MASTER_LOG_POS = log_pos
          |   RELAY_LOG_FILE = 'log_name', RELAY_LOG_POS = log_pos
          |   SQL_AFTER_MTS_GAPS  }
```

```
connection_options:
    [USER='user_name'] [PASSWORD='user_pass'] [DEFAULT_AUTH='plugin_name'] [PLUGIN_DIR='plugin_dir']


gtid_set:
    uuid_set [, uuid_set] ...
    | ''

uuid_set:
    uuid:interval[:interval]...

uuid:
    hhhhhhhh-hhhh-hhhh-hhhh-hhhhhhhhhhhh

h:
    [0-9,A-F]

interval:
    n[-n]

    (n >= 1)
```

START SLAVE with no *thread_type* options starts both of the slave threads. The I/O thread reads events from the master server and stores them in the relay log. The SQL thread reads events from the relay log and executes them. START SLAVE requires the SUPER privilege.

If START SLAVE succeeds in starting the slave threads, it returns without any error. However, even in that case, it might be that the slave threads start and then later stop (for example, because they do not manage to connect to the master or read its binary log, or some other problem). START SLAVE does not warn you about this. You must check the slave's error log for error messages generated by the slave threads, or check that they are running satisfactorily with SHOW SLAVE STATUS.

In MySQL 5.7, START SLAVE causes an implicit commit of an ongoing transaction. See Section 13.3.3, “Statements That Cause an Implicit Commit”.

Beginning with MySQL 5.7.1, gtid_next must be set to AUTOMATIC before issuing this statement (Bug #16062608).

MySQL 5.7 supports pluggable user-password authentication with START SLAVE with the USER, PASSWORD, DEFAULT_AUTH and PLUGIN_DIR options, as described in the following list:

- USER: User name. Cannot be set to an empty or null string, or left unset if PASSWORD is used.
- PASSWORD: Password.
- DEFAULT_AUTH: Name of plugin; default is MySQL native authentication.
- PLUGIN_DIR: Location of plugin.

You cannot use the SQL_THREAD option when specifying any of USER, PASSWORD, DEFAULT_AUTH, or PLUGIN_DIR, unless the IO_THREAD option is also provided.

See Section 6.3.8, “Pluggable Authentication”, for more information.

If an insecure connection is used with any these options, the server issues the warning Sending passwords in plain text without SSL/TLS is extremely insecure.

START SLAVE ... UNTIL supports two additional options for use with global transaction identifiers (GTIDs) (see Section 16.1.3, “Replication with Global Transaction Identifiers”). Each of these takes a set of one or more global transaction identifiers *gtid_set* as an argument (see GTID sets, for more information).

When no *thread_type* is specified, START SLAVE UNTIL SQL_BEFORE_GTIDS causes the slave SQL thread to process transactions until it has reached the *first* transaction whose GTID is listed in the *gtid_set*. START SLAVE UNTIL SQL_AFTER_GTIDS causes the slave threads to process all transactions until the *last* transaction in the *gtid_set* has been processed by both threads. In other words, START SLAVE UNTIL SQL_BEFORE_GTIDS causes the slave SQL thread to process all transactions occurring before the first GTID in the *gtid_set* is reached, and START SLAVE UNTIL SQL_AFTER_GTIDS causes the slave threads to handle all transactions, including those whose GTIDs are found in *gtid_set*, until each has encountered a transaction whose GTID is not part of the set. SQL_BEFORE_GTIDS and SQL_AFTER_GTIDS each support the SQL_THREAD and IO_THREAD options, although using IO_THREAD with them currently has no effect.

For example, START SLAVE SQL_THREAD UNTIL SQL_BEFORE_GTIDS = 3E11FA47-71CA-11E1-9E33-C80AA9429562:11-56 causes the slave SQL thread to process all transactions originating from the master whose server_uuid [2162] is 3E11FA47-71CA-11E1-9E33-C80AA9429562 until it encounters the transaction having sequence number 11; it then stops without processing this transaction. In other words, all transactions up to and including the transaction with sequence number 10 are processed. Executing START SLAVE SQL_THREAD UNTIL SQL_AFTER_GTIDS = 3E11FA47-71CA-11E1-9E33-C80AA9429562:11-56, on the other hand, would cause the slave SQL thread to obtain all transactions just mentioned from the master, including all of the transactions having the sequence numbers 11 through 56, and then to stop without processing any additional transactions; that is, the transaction having sequence number 56 would be the last transaction fetched by the slave SQL thread.

Prior to MySQL 5.7.3, SQL_AFTER_GTIDS did not stop the slave once the indicated transaction waa completed, but waited until another GTID event was received (Bug #14767986).

START SLAVE UNTIL SQL_AFTER_MTS_GAPS causes a multi-threaded slave's SQL threads to run until no more gaps are found in the relay log, and then to stop. This statement can take an SQL_THREAD option, but the effects of the statement remain unchanged. It has no effect on the slave I/O thread (and cannot be used with the IO_THREAD option). START SLAVE UNTIL SQL_AFTER_MTS_GAPS should be used before switching the slave from multi-threaded mode to single-threaded mode (that is, when resetting slave_parallel_workers back to 0 from a positive, nonzero value) after slave has failed with errors in multi-threaded mode.

To change a failed multi-threaded slave to single-threaded mode, you can issue the following series of statements, in the order shown:

```
START SLAVE UNTIL SQL_AFTER_MTS_GAPS;

SET @@GLOBAL.slave_parallel_workers = 0;

START SLAVE SQL_THREAD;
```

If you were running the failed multi-threaded slave with relay_log_recovery enabled, then you must issue START SLAVE UNTIL SQL_AFTER_MTS_GAPS prior to executing CHANGE MASTER TO. Otherwise the latter statement fails.

Note

It is possible to view the entire text of a running START SLAVE ... statement, including any USER or PASSWORD values used, in the output of SHOW PROCESSLIST. This is also true for the text of a running CHANGE

> MASTER TO statement, including any values it employs for MASTER_USER or MASTER_PASSWORD.

START SLAVE sends an acknowledgment to the user after both the I/O thread and the SQL thread have started. However, the I/O thread may not yet have connected. For this reason, a successful START SLAVE causes SHOW SLAVE STATUS to show Slave_SQL_Running=Yes, but this does not guarantee that Slave_IO_Running=Yes (because Slave_IO_Running=Yes only if the I/O thread is running *and connected*). For more information, see Section 13.7.5.33, "SHOW SLAVE STATUS Syntax", and Section 16.1.5.1, "Checking Replication Status".

You can add IO_THREAD and SQL_THREAD options to the statement to name which of the threads to start. The SQL_THREAD option is disallowed when specifying any of USER, PASSWORD, DEFAULT_AUTH, or PLUGIN_DIR, unless the IO_THREAD option is also provided.

An UNTIL clause (*until_option*, in the preceding grammar) may be added to specify that the slave should start and run until the SQL thread reaches a given point in the master binary log, specified by the MASTER_LOG_POS and MASTER_LOG_FILE options, or a given point in the slave relay log, indicated with the RELAY_LOG_POS and RELAY_LOG_FILE options. When the SQL thread reaches the point specified, it stops. If the SQL_THREAD option is specified in the statement, it starts only the SQL thread. Otherwise, it starts both slave threads. If the SQL thread is running, the UNTIL clause is ignored and a warning is issued. You cannot use an UNTIL clause with the IO_THREAD option.

It is also possible with START SLAVE UNTIL to specify a stop point relative to a given GTID or set of GTIDs using one of the options SQL_BEFORE_GTIDS or SQL_AFTER_GTIDS, as explained previously in this section. When using one of these options, you can specify SQL_THREAD, IO_THREAD, both of these, or neither of them. If you specify only SQL_THREAD, then only the slave SQL thread is affected by the statement; if only IO_THREAD is used, then only the slave I/O is affected. If both SQL_THREAD and IO_THREAD are used, or if neither of them is used, then both the SQL and I/O threads are affected by the statement.

The UNTIL clause is not supported for multi-threaded slaves except when also using SQL_AFTER_MTS_GAPS.

For an UNTIL clause, you must specify any one of the following:

- *Both* a log file name and a position in that file
- *Either* of SQL_BEFORE_GTIDS or SQL_AFTER_GTIDS
- SQL_AFTER_MTS_GAPS

Do not mix master and relay log options. Do not mix log file options with GTID options.

Any UNTIL condition is reset by a subsequent STOP SLAVE statement, a START SLAVE statement that includes no UNTIL clause, or a server restart.

When specifying a log file and position, you can use the IO_THREAD option with START SLAVE ... UNTIL even though only the SQL thread is affected by this statement. The IO_THREAD option is ignored in such cases. The preceding restriction does not apply when using one of the GTID options (SQL_BEFORE_GTIDS and SQL_AFTER_GTIDS); the GTID options support both SQL_THREAD and IO_THREAD, as explained previously in this section.

The UNTIL clause can be useful for debugging replication, or to cause replication to proceed until just before the point where you want to avoid having the slave replicate an event. For example, if an unwise DROP TABLE statement was executed on the master, you can use UNTIL to tell the slave to execute up to that point but no farther. To find what the event is, use mysqlbinlog with the master binary log or slave relay log, or by using a SHOW BINLOG EVENTS statement.

If you are using UNTIL to have the slave process replicated queries in sections, it is recommended that you start the slave with the --skip-slave-start option to prevent the SQL thread from running when the slave server starts. It is probably best to use this option in an option file rather than on the command line, so that an unexpected server restart does not cause it to be forgotten.

The SHOW SLAVE STATUS statement includes output fields that display the current values of the UNTIL condition.

In very old versions of MySQL (before 4.0.5), this statement was called SLAVE START. In MySQL 5.7, that syntax produces an error.

13.4.2.7 STOP SLAVE Syntax

```
STOP SLAVE [thread_types]

thread_types:
    [thread_type [, thread_type] ... ]

thread_type: IO_THREAD | SQL_THREAD
```

Stops the slave threads. STOP SLAVE requires the SUPER privilege. Recommended best practice is to execute STOP SLAVE on the slave before stopping the slave server (see Section 5.1.12, “The Shutdown Process”, for more information).

When using the row-based logging format: You should execute STOP SLAVE or STOP SLAVE SQL_THREAD on the slave prior to shutting down the slave server if you are replicating any tables that use a nontransactional storage engine (see the *Note* later in this section).

Like START SLAVE, this statement may be used with the IO_THREAD and SQL_THREAD options to name the thread or threads to be stopped.

In MySQL 5.7, STOP SLAVE causes an implicit commit of an ongoing transaction. See Section 13.3.3, “Statements That Cause an Implicit Commit”.

Beginning with MySQL 5.7.1, gtid_next must be set to AUTOMATIC before issuing this statement (Bug #16062608).

In MySQL 5.7.2 and later, you can control how long STOP SLAVE waits before timing out by setting the rpl_stop_slave_timeout system variable. This can be used to avoid deadlocks between STOP SLAVE and other slave SQL statements using different client connections to the slave. (Bug #16856735)

Prior to MySQL 5.7.4, it was necessary to issue this statement on a running slave prior to executing CHANGE MASTER TO. In MySQL 5.7.4 and later, this is no longer always the case; some CHANGE MASTER TO statements are now allowed while the slave is running, depending on the states of the slave SQL and I/O threads. However, using STOP SLAVE prior to executing CHANGE MASTER TO in such cases is still supported. See Section 13.4.2.1, “CHANGE MASTER TO Syntax”, and Section 16.3.6, “Switching Masters During Failover”, for more information.

When using statement-based replication, changing the master while it has open temporary tables is potentially unsafe. (This is one of the reasons why statement-based replication of temporary tables is not recommended.) In MySQL 5.7.4 and later, CHANGE MASTER TO following STOP SLAVE causes a warning (**ER_WARN_OPEN_TEMP_TABLES_MUST_BE_ZERO**) if there are any temporary tables open on the slave. You can find out whether there are any temporary tables on the slave by checking the value of `Slave_open_temp_tables`; when using statement-based replication, this value should be 0 before executing `CHANGE MASTER TO`.

> **Note**
>
> In MySQL 5.7, `STOP SLAVE` waits until the current replication event group affecting one or more nontransactional tables has finished executing (if there is any such replication group), or until the user issues a `KILL QUERY` or `KILL CONNECTION` statement. (Bug #319, Bug #38205)

In very old versions of MySQL (before 4.0.5), this statement was called `SLAVE STOP`. In MySQL 5.7, that syntax produces an error.

13.5 SQL Syntax for Prepared Statements

MySQL 5.7 provides support for server-side prepared statements. This support takes advantage of the efficient client/server binary protocol available since MySQL 4.1. Using prepared statements with placeholders for parameter values has the following benefits:

- Less overhead for parsing the statement each time it is executed. Typically, database applications process large volumes of almost-identical statements, with only changes to literal or variable values in clauses such as `WHERE` for queries and deletes, `SET` for updates, and `VALUES` for inserts.

- Protection against SQL injection attacks. The parameter values can contain unescaped SQL quote and delimiter characters.

Prepared Statements in Application Programs

You can use server-side prepared statements through client programming interfaces, including the MySQL C API client library or MySQL Connector/C for C programs, MySQL Connector/J for Java programs, and MySQL Connector/Net for programs using .NET technologies. For example, the C API provides a set of function calls that make up its prepared statement API. See Section 21.8.8, “C API Prepared Statements”. Other language interfaces can provide support for prepared statements that use the binary protocol by linking in the C client library, one example being the `mysqli` extension, available in PHP 5.0 and later.

Prepared Statements in SQL Scripts

An alternative SQL interface to prepared statements is available. This interface is not as efficient as using the binary protocol through a prepared statement API, but requires no programming because it is available directly at the SQL level:

- You can use it when no programming interface is available to you.

- You can use it from any program that can send SQL statements to the server to be executed, such as the `mysql` client program.

- You can use it even if the client is using an old version of the client library, as long as you connect to a server running MySQL 4.1 or higher.

SQL syntax for prepared statements is intended to be used for situations such as these:

- To test how prepared statements work in your application before coding it.

- To use prepared statements when you do not have access to a programming API that supports them.

- To interactively troubleshoot application issues with prepared statements.
- To create a test case that reproduces a problem with prepared statements, so that you can file a bug report.

PREPARE, EXECUTE, and DEALLOCATE PREPARE Statements

SQL syntax for prepared statements is based on three SQL statements:

- PREPARE prepares a statement for execution (see Section 13.5.1, "PREPARE Syntax").
- EXECUTE executes a prepared statement (see Section 13.5.2, "EXECUTE Syntax").
- DEALLOCATE PREPARE releases a prepared statement (see Section 13.5.3, "DEALLOCATE PREPARE Syntax").

The following examples show two equivalent ways of preparing a statement that computes the hypotenuse of a triangle given the lengths of the two sides.

The first example shows how to create a prepared statement by using a string literal to supply the text of the statement:

```
mysql> PREPARE stmt1 FROM 'SELECT SQRT(POW(?,2) + POW(?,2)) AS hypotenuse';
mysql> SET @a = 3;
mysql> SET @b = 4;
mysql> EXECUTE stmt1 USING @a, @b;
+------------+
| hypotenuse |
+------------+
|          5 |
+------------+
mysql> DEALLOCATE PREPARE stmt1;
```

The second example is similar, but supplies the text of the statement as a user variable:

```
mysql> SET @s = 'SELECT SQRT(POW(?,2) + POW(?,2)) AS hypotenuse';
mysql> PREPARE stmt2 FROM @s;
mysql> SET @a = 6;
mysql> SET @b = 8;
mysql> EXECUTE stmt2 USING @a, @b;
+------------+
| hypotenuse |
+------------+
|         10 |
+------------+
mysql> DEALLOCATE PREPARE stmt2;
```

Here is an additional example that demonstrates how to choose the table on which to perform a query at runtime, by storing the name of the table as a user variable:

```
mysql> USE test;
mysql> CREATE TABLE t1 (a INT NOT NULL);
mysql> INSERT INTO t1 VALUES (4), (8), (11), (32), (80);

mysql> SET @table = 't1';
mysql> SET @s = CONCAT('SELECT * FROM ', @table);

mysql> PREPARE stmt3 FROM @s;
mysql> EXECUTE stmt3;
+----+
| a  |
+----+
|  4 |
|  8 |
| 11 |
| 32 |
| 80 |
+----+

mysql> DEALLOCATE PREPARE stmt3;
```

A prepared statement is specific to the session in which it was created. If you terminate a session without deallocating a previously prepared statement, the server deallocates it automatically.

A prepared statement is also global to the session. If you create a prepared statement within a stored routine, it is not deallocated when the stored routine ends.

To guard against too many prepared statements being created simultaneously, set the `max_prepared_stmt_count` system variable. To prevent the use of prepared statements, set the value to 0.

SQL Syntax Allowed in Prepared Statements

The following SQL statements can be used as prepared statements:

```
ALTER TABLE
ALTER USER
ANALYZE TABLE
CACHE INDEX
CALL
CHANGE MASTER
CHECKSUM {TABLE | TABLES}
COMMIT
{CREATE | DROP} INDEX
{CREATE | RENAME | DROP} DATABASE
{CREATE | DROP} TABLE
{CREATE | RENAME | DROP} USER
{CREATE | DROP} VIEW
DELETE
DO
FLUSH {TABLE | TABLES | TABLES WITH READ LOCK | HOSTS | PRIVILEGES
  | LOGS | STATUS | MASTER | SLAVE | DES_KEY_FILE | USER_RESOURCES}
GRANT
INSERT
INSTALL PLUGIN
KILL
LOAD INDEX INTO CACHE
OPTIMIZE TABLE
RENAME TABLE
REPAIR TABLE
REPLACE
RESET {MASTER | SLAVE | QUERY CACHE}
REVOKE
SELECT
SET
SHOW {WARNINGS | ERRORS}
SHOW BINLOG EVENTS
SHOW CREATE {PROCEDURE | FUNCTION | EVENT | TABLE | VIEW}
SHOW {MASTER | BINARY} LOGS
SHOW {MASTER | SLAVE} STATUS
SLAVE {START | STOP}
TRUNCATE TABLE
UNINSTALL PLUGIN
```

```
UPDATE
```

As of MySQL 5.7.2, for compliance with the SQL standard, which states that diagnostics statements are not preparable, MySQL does not support the following as prepared statements:

- SHOW WARNINGS, SHOW COUNT(*) WARNINGS
- SHOW ERRORS, SHOW COUNT(*) ERRORS
- Statements containing any reference to the warning_count or error_count system variable.

Other statements are not supported in MySQL 5.7.

Generally, statements not permitted in SQL prepared statements are also not permitted in stored programs. Exceptions are noted in Section E.1, “Restrictions on Stored Programs”.

Metadata changes to tables or views referred to by prepared statements are detected and cause automatic repreparation of the statement when it is next executed. For more information, see Section 8.9.4, “Caching of Prepared Statements and Stored Programs”.

Placeholders can be used for the arguments of the LIMIT clause when using prepared statements. See Section 13.2.9, “SELECT Syntax”.

In prepared CALL statements used with PREPARE and EXECUTE, placeholder support for OUT and INOUT parameters is available beginning with MySQL 5.7. See Section 13.2.1, “CALL Syntax”, for an example and a workaround for earlier versions. Placeholders can be used for IN parameters regardless of version.

SQL syntax for prepared statements cannot be used in nested fashion. That is, a statement passed to PREPARE cannot itself be a PREPARE, EXECUTE, or DEALLOCATE PREPARE statement.

SQL syntax for prepared statements is distinct from using prepared statement API calls. For example, you cannot use the mysql_stmt_prepare() C API function to prepare a PREPARE, EXECUTE, or DEALLOCATE PREPARE statement.

SQL syntax for prepared statements can be used within stored procedures, but not in stored functions or triggers. However, a cursor cannot be used for a dynamic statement that is prepared and executed with PREPARE and EXECUTE. The statement for a cursor is checked at cursor creation time, so the statement cannot be dynamic.

SQL syntax for prepared statements does not support multi-statements (that is, multiple statements within a single string separated by “;” characters).

Prepared statements use the query cache under the conditions described in Section 8.9.3.1, “How the Query Cache Operates”.

To write C programs that use the CALL SQL statement to execute stored procedures that contain prepared statements, the CLIENT_MULTI_RESULTS flag must be enabled. This is because each CALL returns a result to indicate the call status, in addition to any result sets that might be returned by statements executed within the procedure.

CLIENT_MULTI_RESULTS can be enabled when you call mysql_real_connect(), either explicitly by passing the CLIENT_MULTI_RESULTS flag itself, or implicitly by passing CLIENT_MULTI_STATEMENTS (which also enables CLIENT_MULTI_RESULTS). For additional information, see Section 13.2.1, “CALL Syntax”.

13.5.1 PREPARE Syntax

```
PREPARE stmt_name FROM preparable_stmt
```

The PREPARE statement prepares a SQL statement and assigns it a name, *stmt_name*, by which to refer to the statement later. The prepared statement is executed with EXECUTE and released with DEALLOCATE PREPARE. For examples, see Section 13.5, “SQL Syntax for Prepared Statements”.

Statement names are not case sensitive. *preparable_stmt* is either a string literal or a user variable that contains the text of the SQL statement. The text must represent a single statement, not multiple statements. Within the statement, ? characters can be used as parameter markers to indicate where data values are to be bound to the query later when you execute it. The ? characters should not be enclosed within quotation marks, even if you intend to bind them to string values. Parameter markers can be used only where data values should appear, not for SQL keywords, identifiers, and so forth.

If a prepared statement with the given name already exists, it is deallocated implicitly before the new statement is prepared. This means that if the new statement contains an error and cannot be prepared, an error is returned and no statement with the given name exists.

The scope of a prepared statement is the session within which it is created, which as several implications:

- A prepared statement created in one session is not available to other sessions.
- When a session ends, whether normally or abnormally, its prepared statements no longer exist. If auto-reconnect is enabled, the client is not notified that the connection was lost. For this reason, clients may wish to disable auto-reconnect. See Section 21.8.16, “Controlling Automatic Reconnection Behavior”.
- A prepared statement created within a stored program continues to exist after the program finishes executing and can be executed outside the program later.
- A statement prepared in stored program context cannot refer to stored procedure or function parameters or local variables because they go out of scope when the program ends and would be unavailable were the statement to be executed later outside the program. As a workaround, refer instead to user-defined variables, which also have session scope; see Section 9.4, “User-Defined Variables”.

13.5.2 EXECUTE Syntax

```
EXECUTE stmt_name
    [USING @var_name [, @var_name] ...]
```

After preparing a statement with PREPARE, you execute it with an EXECUTE statement that refers to the prepared statement name. If the prepared statement contains any parameter markers, you must supply a USING clause that lists user variables containing the values to be bound to the parameters. Parameter values can be supplied only by user variables, and the USING clause must name exactly as many variables as the number of parameter markers in the statement.

You can execute a given prepared statement multiple times, passing different variables to it or setting the variables to different values before each execution.

For examples, see Section 13.5, “SQL Syntax for Prepared Statements”.

13.5.3 DEALLOCATE PREPARE Syntax

```
{DEALLOCATE | DROP} PREPARE stmt_name
```

To deallocate a prepared statement produced with PREPARE, use a DEALLOCATE PREPARE statement that refers to the prepared statement name. Attempting to execute a prepared statement after deallocating it results in an error. If too many prepared statements are created and not deallocated by either the DEALLOCATE PREPARE statement or the end of the session, you might encounter the upper limit enforced by the max_prepared_stmt_count system variable.

For examples, see Section 13.5, "SQL Syntax for Prepared Statements".

13.6 MySQL Compound-Statement Syntax

This section describes the syntax for the BEGIN ... END compound statement and other statements that can be used in the body of stored programs: Stored procedures and functions, triggers, and events. These objects are defined in terms of SQL code that is stored on the server for later invocation (see Chapter 18, *Stored Programs and Views*).

A compound statement is a block that can contain other blocks; declarations for variables, condition handlers, and cursors; and flow control constructs such as loops and conditional tests.

13.6.1 BEGIN ... END Compound-Statement Syntax

```
[begin_label:] BEGIN
    [statement_list]
END [end_label]
```

BEGIN ... END syntax is used for writing compound statements, which can appear within stored programs (stored procedures and functions, triggers, and events). A compound statement can contain multiple statements, enclosed by the BEGIN and END keywords. *statement_list* represents a list of one or more statements, each terminated by a semicolon (;) statement delimiter. The *statement_list* itself is optional, so the empty compound statement (BEGIN END) is legal.

BEGIN ... END blocks can be nested.

Use of multiple statements requires that a client is able to send statement strings containing the ; statement delimiter. In the mysql command-line client, this is handled with the delimiter command. Changing the ; end-of-statement delimiter (for example, to //) permit ; to be used in a program body. For an example, see Section 18.1, "Defining Stored Programs".

A BEGIN ... END block can be labeled. See Section 13.6.2, "Statement Label Syntax".

The optional [NOT] ATOMIC clause is not supported. This means that no transactional savepoint is set at the start of the instruction block and the BEGIN clause used in this context has no effect on the current transaction.

Note

Within all stored programs, the parser treats BEGIN [WORK] as the beginning of a BEGIN ... END block. To begin a transaction in this context, use START TRANSACTION instead.

13.6.2 Statement Label Syntax

```
[begin_label:] BEGIN
    [statement_list]
```

```
END [end_label]

[begin_label:] LOOP
    statement_list
END LOOP [end_label]

[begin_label:] REPEAT
    statement_list
UNTIL search_condition
END REPEAT [end_label]

[begin_label:] WHILE search_condition DO
    statement_list
END WHILE [end_label]
```

Labels are permitted for `BEGIN ... END` blocks and for the `LOOP`, `REPEAT`, and `WHILE` statements. Label use for those statements follows these rules:

- *`begin_label`* must be followed by a colon.
- *`begin_label`* can be given without *`end_label`*. If *`end_label`* is present, it must be the same as *`begin_label`*.
- *`end_label`* cannot be given without *`begin_label`*.
- Labels at the same nesting level must be distinct.
- Labels can be up to 16 characters long.

To refer to a label within the labeled construct, use an `ITERATE` or `LEAVE` statement. The following example uses those statements to continue iterating or terminate the loop:

```
CREATE PROCEDURE doiterate(p1 INT)
BEGIN
  label1: LOOP
    SET p1 = p1 + 1;
    IF p1 < 10 THEN ITERATE label1; END IF;
    LEAVE label1;
  END LOOP label1;
END;
```

The scope of a block label does not include the code for handlers declared within the block. For details, see Section 13.6.7.2, “`DECLARE ... HANDLER` Syntax”.

13.6.3 `DECLARE` Syntax

The `DECLARE` statement is used to define various items local to a program:

- Local variables. See Section 13.6.4, “Variables in Stored Programs”.
- Conditions and handlers. See Section 13.6.7, “Condition Handling”.
- Cursors. See Section 13.6.6, “Cursors”.

`DECLARE` is permitted only inside a `BEGIN ... END` compound statement and must be at its start, before any other statements.

Declarations must follow a certain order. Cursor declarations must appear before handler declarations. Variable and condition declarations must appear before cursor or handler declarations.

13.6.4 Variables in Stored Programs

System variables and user-defined variables can be used in stored programs, just as they can be used outside stored-program context. In addition, stored programs can use DECLARE to define local variables, and stored routines (procedures and functions) can be declared to take parameters that communicate values between the routine and its caller.

- To declare local variables, use the DECLARE statement, as described in Section 13.6.4.1, “Local Variable DECLARE Syntax”.
- Variables can be set directly with the SET statement. See Section 13.7.4, “SET Syntax”.
- Results from queries can be retrieved into local variables using SELECT ... INTO *var_list* or by opening a cursor and using FETCH ... INTO *var_list*. See Section 13.2.9.1, “SELECT ... INTO Syntax”, and Section 13.6.6, “Cursors”.

For information about the scope of local variables and how MySQL resolves ambiguous names, see Section 13.6.4.2, “Local Variable Scope and Resolution”.

It is not permitted to assign the value DEFAULT to stored procedure or function parameters or stored program local variables (for example with a SET *var_name* = DEFAULT statement). In MySQL 5.7, this results in a syntax error.

13.6.4.1 Local Variable DECLARE Syntax

```
DECLARE var_name [, var_name] ... type [DEFAULT value]
```

This statement declares local variables within stored programs. To provide a default value for a variable, include a DEFAULT clause. The value can be specified as an expression; it need not be a constant. If the DEFAULT clause is missing, the initial value is NULL.

Local variables are treated like stored routine parameters with respect to data type and overflow checking. See Section 13.1.12, “CREATE PROCEDURE and CREATE FUNCTION Syntax”.

Variable declarations must appear before cursor or handler declarations.

Local variable names are not case sensitive. Permissible characters and quoting rules are the same as for other identifiers, as described in Section 9.2, “Schema Object Names”.

The scope of a local variable is the BEGIN ... END block within which it is declared. The variable can be referred to in blocks nested within the declaring block, except those blocks that declare a variable with the same name.

13.6.4.2 Local Variable Scope and Resolution

The scope of a local variable is the BEGIN ... END block within which it is declared. The variable can be referred to in blocks nested within the declaring block, except those blocks that declare a variable with the same name.

Because local variables are in scope only during stored program execution, references to them are not permitted in prepared statements created within a stored program. Prepared statement scope is the current session, not the stored program, so the statement could be executed after the program ends, at which point the variables would no longer be in scope. For example, SELECT ... INTO *local_var* cannot be used as a prepared statement. This restriction also applies to stored procedure and function parameters. See Section 13.5.1, “PREPARE Syntax”.

A local variable should not have the same name as a table column. If an SQL statement, such as a SELECT ... INTO statement, contains a reference to a column and a declared local variable with the same name, MySQL currently interprets the reference as the name of a variable. Consider the following procedure definition:

```
CREATE PROCEDURE sp1 (x VARCHAR(5))
BEGIN
  DECLARE xname VARCHAR(5) DEFAULT 'bob';
  DECLARE newname VARCHAR(5);
  DECLARE xid INT;

  SELECT xname, id INTO newname, xid
    FROM table1 WHERE xname = xname;
  SELECT newname;
END;
```

MySQL interprets xname in the SELECT statement as a reference to the xname *variable* rather than the xname *column*. Consequently, when the procedure sp1()is called, the newname variable returns the value 'bob' regardless of the value of the table1.xname column.

Similarly, the cursor definition in the following procedure contains a SELECT statement that refers to xname. MySQL interprets this as a reference to the variable of that name rather than a column reference.

```
CREATE PROCEDURE sp2 (x VARCHAR(5))
BEGIN
  DECLARE xname VARCHAR(5) DEFAULT 'bob';
  DECLARE newname VARCHAR(5);
  DECLARE xid INT;
  DECLARE done TINYINT DEFAULT 0;
  DECLARE cur1 CURSOR FOR SELECT xname, id FROM table1;
  DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

  OPEN cur1;
  read_loop: LOOP
    FETCH FROM cur1 INTO newname, xid;
    IF done THEN LEAVE read_loop; END IF;
    SELECT newname;
  END LOOP;
  CLOSE cur1;
END;
```

See also Section E.1, “Restrictions on Stored Programs”.

13.6.5 Flow Control Statements

MySQL supports the IF, CASE, ITERATE, LEAVE LOOP, WHILE, and REPEAT constructs for flow control within stored programs. It also supports RETURN within stored functions.

Many of these constructs contain other statements, as indicated by the grammar specifications in the following sections. Such constructs may be nested. For example, an IF statement might contain a WHILE loop, which itself contains a CASE statement.

MySQL does not support FOR loops.

13.6.5.1 CASE Syntax

```
CASE case_value
    WHEN when_value THEN statement_list
    [WHEN when_value THEN statement_list] ...
    [ELSE statement_list]
END CASE
```

Or:

```
CASE
    WHEN search_condition THEN statement_list
    [WHEN search_condition THEN statement_list] ...
    [ELSE statement_list]
END CASE
```

The CASE statement for stored programs implements a complex conditional construct.

> **Note**
>
> There is also a CASE *expression*, which differs from the CASE *statement* described here. See Section 12.4, "Control Flow Functions". The CASE statement cannot have an ELSE NULL clause, and it is terminated with END CASE instead of END.

For the first syntax, *case_value* is an expression. This value is compared to the *when_value* expression in each WHEN clause until one of them is equal. When an equal *when_value* is found, the corresponding THEN clause *statement_list* executes. If no *when_value* is equal, the ELSE clause *statement_list* executes, if there is one.

This syntax cannot be used to test for equality with NULL because NULL = NULL is false. See Section 3.3.4.6, "Working with NULL Values".

For the second syntax, each WHEN clause *search_condition* expression is evaluated until one is true, at which point its corresponding THEN clause *statement_list* executes. If no *search_condition* is equal, the ELSE clause *statement_list* executes, if there is one.

If no *when_value* or *search_condition* matches the value tested and the CASE statement contains no ELSE clause, a Case not found for CASE statement error results.

Each *statement_list* consists of one or more SQL statements; an empty *statement_list* is not permitted.

To handle situations where no value is matched by any WHEN clause, use an ELSE containing an empty BEGIN ... END block, as shown in this example. (The indentation used here in the ELSE clause is for purposes of clarity only, and is not otherwise significant.)

```
DELIMITER |

CREATE PROCEDURE p()
  BEGIN
    DECLARE v INT DEFAULT 1;

    CASE v
      WHEN 2 THEN SELECT v;
      WHEN 3 THEN SELECT 0;
      ELSE
        BEGIN
        END;
    END CASE;
  END;
  |
```

13.6.5.2 IF Syntax

```
IF search_condition THEN statement_list
    [ELSEIF search_condition THEN statement_list] ...
    [ELSE statement_list]
END IF
```

The IF statement for stored programs implements a basic conditional construct.

> **Note**
>
> There is also an IF() *function*, which differs from the IF *statement* described here. See Section 12.4, “Control Flow Functions”. The IF statement can have THEN, ELSE, and ELSEIF clauses, and it is terminated with END IF.

If the *search_condition* evaluates to true, the corresponding THEN or ELSEIF clause *statement_list* executes. If no *search_condition* matches, the ELSE clause *statement_list* executes.

Each *statement_list* consists of one or more SQL statements; an empty *statement_list* is not permitted.

An IF ... END IF block, like all other flow-control blocks used within stored programs, must be terminated with a semicolon, as shown in this example:

```
DELIMITER //

CREATE FUNCTION SimpleCompare(n INT, m INT)
  RETURNS VARCHAR(20)

  BEGIN
    DECLARE s VARCHAR(20);

    IF n > m THEN SET s = '>';
    ELSEIF n = m THEN SET s = '=';
    ELSE SET s = '<';
    END IF;

    SET s = CONCAT(n, ' ', s, ' ', m);

    RETURN s;
  END //

DELIMITER ;
```

As with other flow-control constructs, IF ... END IF blocks may be nested within other flow-control constructs, including other IF statements. Each IF must be terminated by its own END IF followed by a semicolon. You can use indentation to make nested flow-control blocks more easily readable by humans (although this is not required by MySQL), as shown here:

```
DELIMITER //

CREATE FUNCTION VerboseCompare (n INT, m INT)
  RETURNS VARCHAR(50)

  BEGIN
    DECLARE s VARCHAR(50);

    IF n = m THEN SET s = 'equals';
    ELSE
      IF n > m THEN SET s = 'greater';
      ELSE SET s = 'less';
      END IF;

      SET s = CONCAT('is ', s, ' than');
    END IF;

    SET s = CONCAT(n, ' ', s, ' ', m, '.');

    RETURN s;
  END //

DELIMITER ;
```

In this example, the inner `IF` is evaluated only if `n` is not equal to `m`.

13.6.5.3 `ITERATE` Syntax

```
ITERATE label
```

`ITERATE` can appear only within `LOOP`, `REPEAT`, and `WHILE` statements. `ITERATE` means "start the loop again."

For an example, see Section 13.6.5.5, "`LOOP` Syntax".

13.6.5.4 `LEAVE` Syntax

```
LEAVE label
```

This statement is used to exit the flow control construct that has the given label. If the label is for the outermost stored program block, `LEAVE` exits the program.

`LEAVE` can be used within `BEGIN ... END` or loop constructs (`LOOP`, `REPEAT`, `WHILE`).

For an example, see Section 13.6.5.5, "`LOOP` Syntax".

13.6.5.5 `LOOP` Syntax

```
[begin_label:] LOOP
    statement_list
END LOOP [end_label]
```

`LOOP` implements a simple loop construct, enabling repeated execution of the statement list, which consists of one or more statements, each terminated by a semicolon (`;`) statement delimiter. The statements within the loop are repeated until the loop is terminated. Usually, this is accomplished with a `LEAVE` statement. Within a stored function, `RETURN` can also be used, which exits the function entirely.

Neglecting to include a loop-termination statement results in an infinite loop.

A `LOOP` statement can be labeled. For the rules regarding label use, see Section 13.6.2, "Statement Label Syntax".

Example:

```
CREATE PROCEDURE doiterate(p1 INT)
BEGIN
```

```
  label1: LOOP
    SET p1 = p1 + 1;
    IF p1 < 10 THEN
      ITERATE label1;
    END IF;
    LEAVE label1;
  END LOOP label1;
  SET @x = p1;
END;
```

13.6.5.6 REPEAT Syntax

```
[begin_label:] REPEAT
    statement_list
UNTIL search_condition
END REPEAT [end_label]
```

The statement list within a REPEAT statement is repeated until the *search_condition* expression is true. Thus, a REPEAT always enters the loop at least once. *statement_list* consists of one or more statements, each terminated by a semicolon (;) statement delimiter.

A REPEAT statement can be labeled. For the rules regarding label use, see Section 13.6.2, “Statement Label Syntax”.

Example:

```
mysql> delimiter //

mysql> CREATE PROCEDURE dorepeat(p1 INT)
    -> BEGIN
    ->   SET @x = 0;
    ->   REPEAT
    ->     SET @x = @x + 1;
    ->   UNTIL @x > p1 END REPEAT;
    -> END
    -> //
Query OK, 0 rows affected (0.00 sec)

mysql> CALL dorepeat(1000)//
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @x//
+------+
| @x   |
+------+
| 1001 |
+------+
1 row in set (0.00 sec)
```

13.6.5.7 RETURN Syntax

```
RETURN expr
```

The RETURN statement terminates execution of a stored function and returns the value *expr* to the function caller. There must be at least one RETURN statement in a stored function. There may be more than one if the function has multiple exit points.

This statement is not used in stored procedures, triggers, or events. The LEAVE statement can be used to exit a stored program of those types.

13.6.5.8 WHILE Syntax

```
[begin_label:] WHILE search_condition DO
    statement_list
END WHILE [end_label]
```

The statement list within a `WHILE` statement is repeated as long as the *`search_condition`* expression is true. *`statement_list`* consists of one or more SQL statements, each terminated by a semicolon (`;`) statement delimiter.

A `WHILE` statement can be labeled. For the rules regarding label use, see Section 13.6.2, “Statement Label Syntax”.

Example:

```
CREATE PROCEDURE dowhile()
BEGIN
  DECLARE v1 INT DEFAULT 5;

  WHILE v1 > 0 DO
    ...
    SET v1 = v1 - 1;
  END WHILE;
END;
```

13.6.6 Cursors

MySQL supports cursors inside stored programs. The syntax is as in embedded SQL. Cursors have these properties:

- Asensitive: The server may or may not make a copy of its result table
- Read only: Not updatable
- Nonscrollable: Can be traversed only in one direction and cannot skip rows

Cursor declarations must appear before handler declarations and after variable and condition declarations.

Example:

```
CREATE PROCEDURE curdemo()
BEGIN
  DECLARE done INT DEFAULT FALSE;
  DECLARE a CHAR(16);
  DECLARE b, c INT;
  DECLARE cur1 CURSOR FOR SELECT id,data FROM test.t1;
  DECLARE cur2 CURSOR FOR SELECT i FROM test.t2;
  DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

  OPEN cur1;
  OPEN cur2;

  read_loop: LOOP
    FETCH cur1 INTO a, b;
    FETCH cur2 INTO c;
    IF done THEN
      LEAVE read_loop;
    END IF;
    IF b < c THEN
```

```
      INSERT INTO test.t3 VALUES (a,b);
    ELSE
      INSERT INTO test.t3 VALUES (a,c);
    END IF;
  END LOOP;

  CLOSE cur1;
  CLOSE cur2;
END;
```

13.6.6.1 Cursor `CLOSE` Syntax

```
CLOSE cursor_name
```

This statement closes a previously opened cursor. For an example, see Section 13.6.6, “Cursors”.

An error occurs if the cursor is not open.

If not closed explicitly, a cursor is closed at the end of the `BEGIN ... END` block in which it was declared.

13.6.6.2 Cursor `DECLARE` Syntax

```
DECLARE cursor_name CURSOR FOR select_statement
```

This statement declares a cursor and associates it with a `SELECT` statement that retrieves the rows to be traversed by the cursor. To fetch the rows later, use a `FETCH` statement. The number of columns retrieved by the `SELECT` statement must match the number of output variables specified in the `FETCH` statement.

The `SELECT` statement cannot have an `INTO` clause.

Cursor declarations must appear before handler declarations and after variable and condition declarations.

A stored program may contain multiple cursor declarations, but each cursor declared in a given block must have a unique name. For an example, see Section 13.6.6, “Cursors”.

For information available through `SHOW` statements, it is possible in many cases to obtain equivalent information by using a cursor with an `INFORMATION_SCHEMA` table.

13.6.6.3 Cursor `FETCH` Syntax

```
FETCH [[NEXT] FROM] cursor_name INTO var_name [, var_name] ...
```

This statement fetches the next row for the `SELECT` statement associated with the specified cursor (which must be open), and advances the cursor pointer. If a row exists, the fetched columns are stored in the named variables. The number of columns retrieved by the `SELECT` statement must match the number of output variables specified in the `FETCH` statement.

If no more rows are available, a No Data condition occurs with SQLSTATE value `'02000'`. To detect this condition, you can set up a handler for it (or for a `NOT FOUND` condition). For an example, see Section 13.6.6, “Cursors”.

13.6.6.4 Cursor `OPEN` Syntax

```
OPEN cursor_name
```

This statement opens a previously declared cursor. For an example, see Section 13.6.6, “Cursors”.

13.6.7 Condition Handling

Conditions may arise during stored program execution that require special handling, such as exiting the current program block or continuing execution. Handlers can be defined for general conditions such as warnings or exceptions, or for specific conditions such as a particular error code. Specific conditions can be assigned names and referred to that way in handlers.

To name a condition, use the `DECLARE ... CONDITION` statement. To declare a handler, use the `DECLARE ... HANDLER` statement. See Section 13.6.7.1, “`DECLARE ... CONDITION` Syntax”, and Section 13.6.7.2, “`DECLARE ... HANDLER` Syntax”. For information about how the server chooses handlers when a condition occurs, see Section 13.6.7.6, “Scope Rules for Handlers”.

To raise a condition, use the `SIGNAL` statement. To modify condition information within a condition handler, use `RESIGNAL`. See Section 13.6.7.1, “`DECLARE ... CONDITION` Syntax”, and Section 13.6.7.2, “`DECLARE ... HANDLER` Syntax”.

To retrieve information from the diagnostics area, use the `GET DIAGNOSTICS` statement (see Section 13.6.7.3, “`GET DIAGNOSTICS` Syntax”). For information about the diagnostics area, see Section 13.6.7.7, “The MySQL Diagnostics Area”.

13.6.7.1 `DECLARE ... CONDITION` Syntax

```
DECLARE condition_name CONDITION FOR condition_value

condition_value:
    mysql_error_code
  | SQLSTATE [VALUE] sqlstate_value
```

The `DECLARE ... CONDITION` statement declares a named error condition, associating a name with a condition that needs specific handling. The name can be referred to in a subsequent `DECLARE ... HANDLER` statement (see Section 13.6.7.2, “`DECLARE ... HANDLER` Syntax”).

Condition declarations must appear before cursor or handler declarations.

The *`condition_value`* for `DECLARE ... CONDITION` can be a MySQL error code (a number) or an SQLSTATE value (a 5-character string literal). You should not use MySQL error code 0 or SQLSTATE values that begin with `'00'`, because those indicate success rather than an error condition. For a list of MySQL error codes and SQLSTATE values, see Section C.3, “Server Error Codes and Messages”.

Using names for conditions can help make stored program code clearer. For example, this handler applies to attempts to drop a nonexistent table, but that is apparent only if you know the meaning of MySQL error code 1051:

```
DECLARE CONTINUE HANDLER FOR 1051
  BEGIN
    -- body of handler
  END;
```

By declaring a name for the condition, the purpose of the handler is more readily seen:

```
DECLARE no_such_table CONDITION FOR 1051;
DECLARE CONTINUE HANDLER FOR no_such_table
  BEGIN
```

```
    -- body of handler
  END;
```

Here is a named condition for the same condition, but based on the corresponding SQLSTATE value rather than the MySQL error code:

```
DECLARE no_such_table CONDITION FOR SQLSTATE '42S02';
DECLARE CONTINUE HANDLER FOR no_such_table
  BEGIN
    -- body of handler
  END;
```

Condition names referred to in `SIGNAL` or use `RESIGNAL` statements must be associated with SQLSTATE values, not MySQL error codes.

13.6.7.2 `DECLARE ... HANDLER` Syntax

```
DECLARE handler_action HANDLER
    FOR condition_value [, condition_value] ...
    statement

handler_action:
    CONTINUE
  | EXIT
  | UNDO

condition_value:
    mysql_error_code
  | SQLSTATE [VALUE] sqlstate_value
  | condition_name
  | SQLWARNING
  | NOT FOUND
  | SQLEXCEPTION
```

The `DECLARE ... HANDLER` statement specifies a handler that deals with one or more conditions. If one of these conditions occurs, the specified *`statement`* executes. *`statement`* can be a simple statement such as `SET var_name = value`, or a compound statement written using `BEGIN` and `END` (see Section 13.6.1, “`BEGIN ... END` Compound-Statement Syntax”).

Handler declarations must appear after variable or condition declarations.

The *`handler_action`* value indicates what action the handler takes after execution of the handler statement:

- `CONTINUE`: Execution of the current program continues.
- `EXIT`: Execution terminates for the `BEGIN ... END` compound statement in which the handler is declared. This is true even if the condition occurs in an inner block.
- `UNDO`: Not supported.

The *`condition_value`* for `DECLARE ... HANDLER` indicates the specific condition or class of conditions that activates the handler:

- A MySQL error code (a number) or an SQLSTATE value (a 5-character string literal). You should not use MySQL error code 0 or SQLSTATE values that begin with `'00'`, because those indicate success rather than an error condition. For a list of MySQL error codes and SQLSTATE values, see Section C.3, “Server Error Codes and Messages”.

- A condition name previously specified with DECLARE ... CONDITION. A condition name can be associated with a MySQL error code or SQLSTATE value. See Section 13.6.7.1, "DECLARE ... CONDITION Syntax".

- SQLWARNING is shorthand for the class of SQLSTATE values that begin with '01'.

- NOT FOUND is shorthand for the class of SQLSTATE values that begin with '02'. This is relevant within the context of cursors and is used to control what happens when a cursor reaches the end of a data set. If no more rows are available, a No Data condition occurs with SQLSTATE value '02000'. To detect this condition, you can set up a handler for it (or for a NOT FOUND condition). For an example, see Section 13.6.6, "Cursors". This condition also occurs for SELECT ... INTO *var_list* statements that retrieve no rows.

- SQLEXCEPTION is shorthand for the class of SQLSTATE values that do not begin with '00', '01', or '02'.

For information about how the server chooses handlers when a condition occurs, see Section 13.6.7.6, "Scope Rules for Handlers".

If a condition occurs for which no handler has been declared, the action taken depends on the condition class:

- For SQLEXCEPTION conditions, the stored program terminates at the statement that raised the condition, as if there were an EXIT handler. If the program was called by another stored program, the calling program handles the condition using the handler selection rules applied to its own handlers.

- For SQLWARNING conditions, the program continues executing, as if there were a CONTINUE handler.

- For NOT FOUND conditions, if the condition was raised normally, the action is CONTINUE. If it was raised by SIGNAL or RESIGNAL, the action is EXIT.

The following example uses a handler for SQLSTATE '23000', which occurs for a duplicate-key error:

```
mysql> CREATE TABLE test.t (s1 INT, PRIMARY KEY (s1));
Query OK, 0 rows affected (0.00 sec)

mysql> delimiter //

mysql> CREATE PROCEDURE handlerdemo ()
    -> BEGIN
    ->   DECLARE CONTINUE HANDLER FOR SQLSTATE '23000' SET @x2 = 1;
    ->   SET @x = 1;
    ->   INSERT INTO test.t VALUES (1);
    ->   SET @x = 2;
    ->   INSERT INTO test.t VALUES (1);
    ->   SET @x = 3;
    -> END;
    -> //
Query OK, 0 rows affected (0.00 sec)

mysql> CALL handlerdemo()//
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @x//
    +------+
    | @x   |
    +------+
    | 3    |
    +------+
    1 row in set (0.00 sec)
```

Notice that @x is 3 after the procedure executes, which shows that execution continued to the end of the procedure after the error occurred. If the DECLARE ... HANDLER statement had not been present, MySQL would have taken the default action (EXIT) after the second INSERT failed due to the PRIMARY KEY constraint, and SELECT @x would have returned 2.

To ignore a condition, declare a CONTINUE handler for it and associate it with an empty block. For example:

```
DECLARE CONTINUE HANDLER FOR SQLWARNING BEGIN END;
```

The scope of a block label does not include the code for handlers declared within the block. Therefore, the statement associated with a handler cannot use ITERATE or LEAVE to refer to labels for blocks that enclose the handler declaration. Consider the following example, where the REPEAT block has a label of retry:

```
CREATE PROCEDURE p ()
BEGIN
  DECLARE i INT DEFAULT 3;
  retry:
    REPEAT
      BEGIN
        DECLARE CONTINUE HANDLER FOR SQLWARNING
          BEGIN
            ITERATE retry;    # illegal
          END;
        IF i < 0 THEN
          LEAVE retry;        # legal
        END IF;
        SET i = i - 1;
      END;
    UNTIL FALSE END REPEAT;
END;
```

The retry label is in scope for the IF statement within the block. It is not in scope for the CONTINUE handler, so the reference there is invalid and results in an error:

```
ERROR 1308 (42000): LEAVE with no matching label: retry
```

To avoid references to outer labels in handlers, use one of these strategies:

- To leave the block, use an EXIT handler. If no block cleanup is required, the BEGIN ... END handler body can be empty:

  ```
  DECLARE EXIT HANDLER FOR SQLWARNING BEGIN END;
  ```

  Otherwise, put the cleanup statements in the handler body:

  ```
  DECLARE EXIT HANDLER FOR SQLWARNING
    BEGIN
      block cleanup statements
    END;
  ```

- To continue execution, set a status variable in a CONTINUE handler that can be checked in the enclosing block to determine whether the handler was invoked. The following example uses the variable done for this purpose:

  ```
  CREATE PROCEDURE p ()
  ```

```
BEGIN
  DECLARE i INT DEFAULT 3;
  DECLARE done INT DEFAULT FALSE;
  retry:
    REPEAT
      BEGIN
        DECLARE CONTINUE HANDLER FOR SQLWARNING
          BEGIN
            SET done = TRUE;
          END;
        IF done OR i < 0 THEN
          LEAVE retry;
        END IF;
        SET i = i - 1;
      END;
    UNTIL FALSE END REPEAT;
END;
```

13.6.7.3 GET DIAGNOSTICS Syntax

```
GET [CURRENT | STACKED] DIAGNOSTICS
{
    statement_information_item
    [, statement_information_item] ...
  | CONDITION condition_number
    condition_information_item
    [, condition_information_item] ...
}

statement_information_item:
    target = statement_information_item_name

condition_information_item:
    target = condition_information_item_name

statement_information_item_name:
    NUMBER
  | ROW_COUNT

condition_information_item_name:
    CLASS_ORIGIN
  | SUBCLASS_ORIGIN
  | RETURNED_SQLSTATE
  | MESSAGE_TEXT
  | MYSQL_ERRNO
  | CONSTRAINT_CATALOG
  | CONSTRAINT_SCHEMA
  | CONSTRAINT_NAME
  | CATALOG_NAME
  | SCHEMA_NAME
  | TABLE_NAME
  | COLUMN_NAME
  | CURSOR_NAME

condition_number, target:
    (see following discussion)
```

SQL statements produce diagnostic information that populates the diagnostics area. The `GET DIAGNOSTICS` statement enables applications to inspect this information. (You can also use `SHOW WARNINGS` or `SHOW ERRORS` to see conditions or errors.)

No special privileges are required to execute `GET DIAGNOSTICS`.

The keyword `CURRENT` means to retrieve information from the current diagnostics area. The keyword `STACKED` means to retrieve information from the second diagnostics area, which is available only if the current context is a condition handler. If neither keyword is given, the default is to use the current diagnostics area.

The GET DIAGNOSTICS statement is typically used in a handler within a stored program. It is a MySQL extension that GET [CURRENT] DIAGNOSTICS is permitted outside handler context to check the execution of any SQL statement. For example, if you invoke the mysql client program, you can enter these statements at the prompt:

```
mysql> DROP TABLE test.no_such_table;
ERROR 1051 (42S02): Unknown table 'test.no_such_table'
mysql> GET DIAGNOSTICS CONDITION 1
    ->   @p1 = RETURNED_SQLSTATE, @p2 = MESSAGE_TEXT;
mysql> SELECT @p1, @p2;
+-------+------------------------------------+
| @p1   | @p2                                |
+-------+------------------------------------+
| 42S02 | Unknown table 'test.no_such_table' |
+-------+------------------------------------+
```

This extension applies only to the current diagnostics area. It does not apply to the second diagnostics area because GET STACKED DIAGNOSTICS is permitted only if the current context is a condition handler. If that is not the case, a GET STACKED DIAGNOSTICS when handler not active error occurs.

For a description of the diagnostics area, see Section 13.6.7.7, “The MySQL Diagnostics Area”. Briefly, it contains two kinds of information:

- Statement information, such as the number of conditions that occurred or the affected-rows count.
- Condition information, such as the error code and message. If a statement raises multiple conditions, this part of the diagnostics area has a condition area for each one. If a statement raises no conditions, this part of the diagnostics area is empty.

For a statement that produces three conditions, the diagnostics area contains statement and condition information like this:

```
Statement information:
  row count
  ... other statement information items ...
Condition area list:
  Condition area 1:
    error code for condition 1
    error message for condition 1
    ... other condition information items ...
  Condition area 2:
    error code for condition 2:
    error message for condition 2
    ... other condition information items ...
  Condition area 3:
    error code for condition 3
    error message for condition 3
    ... other condition information items ...
```

GET DIAGNOSTICS can obtain either statement or condition information, but not both in the same statement:

- To obtain statement information, retrieve the desired statement items into target variables. This instance of GET DIAGNOSTICS assigns the number of available conditions and the rows-affected count to the user variables @p1 and @p2:

```
GET DIAGNOSTICS @p1 = NUMBER, @p2 = ROW_COUNT;
```

- To obtain condition information, specify the condition number and retrieve the desired condition items into target variables. This instance of `GET DIAGNOSTICS` assigns the SQLSTATE value and error message to the user variables `@p3` and `@p4`:

```
GET DIAGNOSTICS CONDITION 1
  @p3 = RETURNED_SQLSTATE, @p4 = MESSAGE_TEXT;
```

The retrieval list specifies one or more *`target = item_name`* assignments, separated by commas. Each assignment names a target variable and either a *`statement_information_item_name`* or *`condition_information_item_name`* designator, depending on whether the statement retrieves statement or condition information.

Valid *`target`* designators for storing item information can be stored procedure or function parameters, stored program local variables declared with `DECLARE`, or user-defined variables.

Valid *`condition_number`* designators can be stored procedure or function parameters, stored program local variables declared with `DECLARE`, user-defined variables, system variables, or literals. A character literal may include a *`_charset`* introducer. A warning occurs if the condition number is not in the range from 1 to the number of condition areas that have information. In this case, the warning is added to the diagnostics area without clearing it.

Currently, when a condition occurs, MySQL does not populate all condition items recognized by `GET DIAGNOSTICS`. For example:

```
mysql> GET DIAGNOSTICS CONDITION 1
    ->   @p5 = SCHEMA_NAME, @p6 = TABLE_NAME;
mysql> SELECT @p5, @p6;
+------+------+
| @p5  | @p6  |
+------+------+
|      |      |
+------+------+
```

In standard SQL, if there are multiple conditions, the first condition relates to the `SQLSTATE` value returned for the previous SQL statement. In MySQL, this is not guaranteed. To get the main error, you cannot do this:

```
GET DIAGNOSTICS CONDITION 1 @errno = MYSQL_ERRNO;
```

Instead, retrieve the condition count first, then use it to specify which condition number to inspect:

```
GET DIAGNOSTICS @cno = NUMBER;
GET DIAGNOSTICS CONDITION @cno @errno = MYSQL_ERRNO;
```

For information about permissible statement and condition information items, and which ones are populated when a condition occurs, see Diagnostics Area Information Items.

Here is an example that uses `GET DIAGNOSTICS` and an exception handler in stored procedure context to assess the outcome of an insert operation. If the insert was successful, the procedure uses `GET DIAGNOSTICS` to get the rows-affected count. This shows that you can use `GET DIAGNOSTICS` multiple times to retrieve information about a statement as long as the current diagnostics area has not been cleared.

```
CREATE PROCEDURE do_insert(value INT)
BEGIN
  -- Declare variables to hold diagnostics area information
  DECLARE code CHAR(5) DEFAULT '00000';
  DECLARE msg TEXT;
  DECLARE rows INT;
  DECLARE result TEXT;
  -- Declare exception handler for failed insert
  DECLARE CONTINUE HANDLER FOR SQLEXCEPTION
    BEGIN
      GET DIAGNOSTICS CONDITION 1
        code = RETURNED_SQLSTATE, msg = MESSAGE_TEXT;
    END;

  -- Perform the insert
  INSERT INTO t1 (int_col) VALUES(value);
  -- Check whether the insert was successful
  IF code = '00000' THEN
    GET DIAGNOSTICS rows = ROW_COUNT;
    SET result = CONCAT('insert succeeded, row count = ',rows);
  ELSE
    SET result = CONCAT('insert failed, error = ',code,', message = ',msg);
  END IF;
  -- Say what happened
  SELECT result;
END;
```

Suppose that `t1.int_col` is an integer column that is declared as `NOT NULL`. The procedure produces these results when invoked to insert non-`NULL` and `NULL` values:

```
mysql> CALL do_insert(1);
+---------------------------------+
| result                          |
+---------------------------------+
| insert succeeded, row count = 1 |
+---------------------------------+

mysql> CALL do_insert(NULL);
+-------------------------------------------------------------------------+
| result                                                                  |
+-------------------------------------------------------------------------+
| insert failed, error = 23000, message = Column 'int_col' cannot be null |
+-------------------------------------------------------------------------+
```

When a condition handler activates, a push to the diagnostics area stack occurs. The first (current) diagnostics area becomes the second (stacked) diagnostics area and a new current diagnostics area is created as a copy of it. `GET [CURRENT] DIAGNOSTICS` and `GET STACKED DIAGNOSTICS` can be used within the handler to access the contents of the current and stacked diagnostics areas. Initially, they return the same result, so it is possible to get information from the current diagnostics area about the condition that activated the handler, *as long as* you execute no statements within the handler that change its current diagnostics area. However, statements executing within the handler can modify the current diagnostics area, clearing and setting its contents according to the normal rules (see How the Diagnostics Area is Populated).

A more reliable way to obtain information about the handler-activating condition is to use the stacked diagnostics area, which cannot be modified by statements executing within the handler except `RESIGNAL`. For information about when the current diagnostics area is set and cleared, see Section 13.6.7.7, “The MySQL Diagnostics Area”.

The next example shows how `GET STACKED DIAGNOSTICS` can be used within a handler to obtain information about the handled exception, even after the current diagnostics area has been modified by handler statements.

Within a stored procedure `p()`, we attempt to insert two values into a table that contains a `TEXT NOT NULL` column. The first value is a non-`NULL` string and the second is `NULL`. The column prohibits `NULL` values, so the first insert succeeds but the second causes an exception. The procedure includes an exception handler that maps attempts to insert `NULL` into inserts of the empty string:

```
DROP TABLE IF EXISTS t1;
CREATE TABLE t1 (c1 TEXT NOT NULL);
DROP PROCEDURE IF EXISTS p;
delimiter //
END//
CREATE PROCEDURE p ()
BEGIN
  DECLARE EXIT HANDLER FOR SQLEXCEPTION
  BEGIN
    DECLARE num INT;
    DECLARE errno INT;
    DECLARE msg TEXT;

    -- Here the current DA is nonempty because the handler was invoked
    GET CURRENT DIAGNOSTICS CONDITION 1
      errno = MYSQL_ERRNO, msg = MESSAGE_TEXT;
    SELECT 'current DA before mapped insert' AS op, errno, msg;
    GET STACKED DIAGNOSTICS CONDITION 1
      errno = MYSQL_ERRNO, msg = MESSAGE_TEXT;
    SELECT 'stacked DA before mapped insert' AS op, errno, msg;

    -- Map attempted NULL insert to empty string insert
    INSERT INTO t1 (c1) VALUES('');

    -- Here the current DA should be empty (if the INSERT succeeded),
    -- so check whether there are conditions before attempting to
    -- obtain condition information
    GET CURRENT DIAGNOSTICS num = NUMBER;
    IF num = 0
    THEN
      SELECT 'INSERT succeeded, current DA is empty' AS op;
    ELSE
      GET CURRENT DIAGNOSTICS CONDITION 1
        errno = MYSQL_ERRNO, msg = MESSAGE_TEXT;
      SELECT 'current DA after mapped insert' AS op, errno, msg;
    END IF ;
    GET STACKED DIAGNOSTICS CONDITION 1
      errno = MYSQL_ERRNO, msg = MESSAGE_TEXT;
    SELECT 'stacked DA after mapped insert' AS op, errno, msg;
  END;
  INSERT INTO t1 (c1) VALUES('string 1');
  INSERT INTO t1 (c1) VALUES(NULL);
END;
//
delimiter ;
CALL p();
SELECT * FROM t1;
```

When the handler activates, a copy of the current diagnostics area is pushed to the diagnostics area stack. The handler first displays the contents of the current and stacked diagnostics areas, which are both the same initially:

```
+---------------------------------+-------+----------------------------+
| op                              | errno | msg                        |
+---------------------------------+-------+----------------------------+
| current DA before mapped insert |  1048 | Column 'c1' cannot be null |
+---------------------------------+-------+----------------------------+

+---------------------------------+-------+----------------------------+
| op                              | errno | msg                        |
+---------------------------------+-------+----------------------------+
| stacked DA before mapped insert |  1048 | Column 'c1' cannot be null |
+---------------------------------+-------+----------------------------+
```

Then the handler maps the `NULL` insert to an empty-string insert and displays the result. The new insert succeeds and clears the current diagnostics area, but the stacked diagnostics area remains unchanged and still contains information about the condition that activated the handler:

```
+---------------------------------------+
| op                                    |
+---------------------------------------+
| INSERT succeeded, current DA is empty |
+---------------------------------------+

+--------------------------------+-------+----------------------------+
| op                             | errno | msg                        |
+--------------------------------+-------+----------------------------+
| stacked DA after mapped insert |  1048 | Column 'c1' cannot be null |
+--------------------------------+-------+----------------------------+
```

When the condition handler ends, its current diagnostics area is popped from the stack and the stacked diagnostics area becomes the current diagnostics area in the stored procedure.

After the procedure returns, the table contains two rows. The empty row results from the attempt to insert `NULL` that was mapped to an empty-string insert:

```
+----------+
| c1       |
+----------+
| string 1 |
|          |
+----------+
```

13.6.7.4 RESIGNAL Syntax

```
RESIGNAL [condition_value]
    [SET signal_information_item
    [, signal_information_item] ...]

condition_value:
    SQLSTATE [VALUE] sqlstate_value
  | condition_name

signal_information_item:
    condition_information_item_name = simple_value_specification

condition_information_item_name:
    CLASS_ORIGIN
  | SUBCLASS_ORIGIN
  | MESSAGE_TEXT
  | MYSQL_ERRNO
  | CONSTRAINT_CATALOG
  | CONSTRAINT_SCHEMA
  | CONSTRAINT_NAME
  | CATALOG_NAME
  | SCHEMA_NAME
  | TABLE_NAME
  | COLUMN_NAME
  | CURSOR_NAME

condition_name, simple_value_specification:
    (see following discussion)
```

RESIGNAL passes on the error condition information that is available during execution of a condition handler within a compound statement inside a stored procedure or function, trigger, or event. RESIGNAL may change some or all information before passing it on. RESIGNAL is related to SIGNAL, but instead of originating a condition as SIGNAL does, RESIGNAL relays existing condition information, possibly after modifying it.

RESIGNAL makes it possible to both handle an error and return the error information. Otherwise, by executing an SQL statement within the handler, information that caused the handler's activation is destroyed. RESIGNAL also can make some procedures shorter if a given handler can handle part of a situation, then pass the condition “up the line” to another handler.

No special privileges are required to execute the RESIGNAL statement.

All forms of RESIGNAL require that the current context be a condition handler. Otherwise, RESIGNAL is illegal and a RESIGNAL when handler not active error occurs.

To retrieve information from the diagnostics area, use the GET DIAGNOSTICS statement (see Section 13.6.7.3, “GET DIAGNOSTICS Syntax”). For information about the diagnostics area, see Section 13.6.7.7, “The MySQL Diagnostics Area”.

For *condition_value* and *signal_information_item*, the definitions and rules are the same for RESIGNAL as for SIGNAL. For example, the *condition_value* can be an SQLSTATE value, and the value can indicate errors, warnings, or “not found.” For additional information, see Section 13.6.7.5, “SIGNAL Syntax”.

The RESIGNAL statement takes *condition_value* and SET clauses, both of which are optional. This leads to several possible uses:

- RESIGNAL alone:

  ```
  RESIGNAL;
  ```

- RESIGNAL with new signal information:

  ```
  RESIGNAL SET signal_information_item [, signal_information_item] ...;
  ```

- RESIGNAL with a condition value and possibly new signal information:

  ```
  RESIGNAL condition_value
      [SET signal_information_item [, signal_information_item] ...];
  ```

These use cases all cause changes to the diagnostics and condition areas:

- A diagnostics area contains one or more condition areas.
- A condition area contains condition information items, such as the SQLSTATE value, MYSQL_ERRNO, or MESSAGE_TEXT.

There is a stack of diagnostics areas. When a handler takes control, it pushes a diagnostics area to the top of the stack, so there are two diagnostics areas during handler execution:

- The first (current) diagnostics area, which starts as a copy of the last diagnostics area, but will be overwritten by the first statement in the handler that changes the current diagnostics area.

- The last (stacked) diagnostics area, which has the condition areas that were set up before the handler took control.

The maximum number of condition areas in a diagnostics area is determined by the value of the `max_error_count` system variable. See Diagnostics Area-Related System Variables.

`RESIGNAL` Alone

A simple `RESIGNAL` alone means “pass on the error with no change.” It restores the last diagnostics area and makes it the current diagnostics area. That is, it “pops” the diagnostics area stack.

Within a condition handler that catches a condition, one use for `RESIGNAL` alone is to perform some other actions, and then pass on without change the original condition information (the information that existed before entry into the handler).

Example:

```
DROP TABLE IF EXISTS xx;
delimiter //
CREATE PROCEDURE p ()
BEGIN
  DECLARE EXIT HANDLER FOR SQLEXCEPTION
  BEGIN
    SET @error_count = @error_count + 1;
    IF @a = 0 THEN RESIGNAL; END IF;
  END;
  DROP TABLE xx;
END//
delimiter ;
SET @error_count = 0;
SET @a = 0;
CALL p();
```

Suppose that the `DROP TABLE xx` statement fails. The diagnostics area stack looks like this:

```
DA 1. ERROR 1051 (42S02): Unknown table 'xx'
```

Then execution enters the `EXIT` handler. It starts by pushing a diagnostics area to the top of the stack, which now looks like this:

```
DA 1. ERROR 1051 (42S02): Unknown table 'xx'
DA 2. ERROR 1051 (42S02): Unknown table 'xx'
```

At this point, the contents of the first (current) and second (stacked) diagnostics areas are the same. The first diagnostics area may be modified by statements executing subsequently within the handler.

Usually a procedure statement clears the first diagnostics area. `BEGIN` is an exception, it does not clear, it does nothing. `SET` is not an exception, it clears, performs the operation, and produces a result of “success.” The diagnostics area stack now looks like this:

```
DA 1. ERROR 0000 (00000): Successful operation
DA 2. ERROR 1051 (42S02): Unknown table 'xx'
```

At this point, if `@a = 0`, `RESIGNAL` pops the diagnostics area stack, which now looks like this:

```
DA 1. ERROR 1051 (42S02): Unknown table 'xx'
```

And that is what the caller sees.

If `@a` is not 0, the handler simply ends, which means that there is no more use for the current diagnostics area (it has been “handled”), so it can be thrown away, causing the stacked diagnostics area to become the current diagnostics area again. The diagnostics area stack looks like this:

```
DA 1. ERROR 0000 (00000): Successful operation
```

The details make it look complex, but the end result is quite useful: Handlers can execute without destroying information about the condition that caused activation of the handler.

`RESIGNAL` with New Signal Information

`RESIGNAL` with a `SET` clause provides new signal information, so the statement means “pass on the error with changes”:

```
RESIGNAL SET signal_information_item [, signal_information_item] ...;
```

As with `RESIGNAL` alone, the idea is to pop the diagnostics area stack so that the original information will go out. Unlike `RESIGNAL` alone, anything specified in the `SET` clause changes.

Example:

```
DROP TABLE IF EXISTS xx;
delimiter //
CREATE PROCEDURE p ()
BEGIN
  DECLARE EXIT HANDLER FOR SQLEXCEPTION
  BEGIN
    SET @error_count = @error_count + 1;
    IF @a = 0 THEN RESIGNAL SET MYSQL_ERRNO = 5; END IF;
  END;
  DROP TABLE xx;
END//
delimiter ;
SET @error_count = 0;
SET @a = 0;
CALL p();
```

Remember from the previous discussion that `RESIGNAL` alone results in a diagnostics area stack like this:

```
DA 1. ERROR 1051 (42S02): Unknown table 'xx'
```

The `RESIGNAL SET MYSQL_ERRNO = 5` statement results in this stack instead, which is what the caller sees:

```
DA 1. ERROR 5 (42S02): Unknown table 'xx'
```

In other words, it changes the error number, and nothing else.

The `RESIGNAL` statement can change any or all of the signal information items, making the first condition area of the diagnostics area look quite different.

`RESIGNAL` with a Condition Value and Optional New Signal Information

`RESIGNAL` with a condition value means “push a condition into the current diagnostics area.” If the `SET` clause is present, it also changes the error information.

```
RESIGNAL condition_value
```

```
    [SET signal_information_item [, signal_information_item] ...];
```

This form of RESIGNAL restores the last diagnostics area and makes it the current diagnostics area. That is, it “pops” the diagnostics area stack, which is the same as what a simple RESIGNAL alone would do. However, it also changes the diagnostics area depending on the condition value or signal information.

Example:

```
DROP TABLE IF EXISTS xx;
delimiter //
CREATE PROCEDURE p ()
BEGIN
  DECLARE EXIT HANDLER FOR SQLEXCEPTION
  BEGIN
    SET @error_count = @error_count + 1;
    IF @a = 0 THEN RESIGNAL SQLSTATE '45000' SET MYSQL_ERRNO=5; END IF;
  END;
  DROP TABLE xx;
END//
delimiter ;
SET @error_count = 0;
SET @a = 0;
SET @@max_error_count = 2;
CALL p();
SHOW ERRORS;
```

This is similar to the previous example, and the effects are the same, except that if RESIGNAL happens, the current condition area looks different at the end. (The reason the condition adds to rather than replaces the existing condition is the use of a condition value.)

The RESIGNAL statement includes a condition value (SQLSTATE '45000'), so it adds a new condition area, resulting in a diagnostics area stack that looks like this:

```
DA 1. (condition 2) ERROR 1051 (42S02): Unknown table 'xx'
      (condition 1) ERROR 5 (45000) Unknown table 'xx'
```

The result of CALL p() and SHOW ERRORS for this example is:

```
mysql> CALL p();
ERROR 5 (45000): Unknown table 'xx'
mysql> SHOW ERRORS;
+-------+------+----------------------------------+
| Level | Code | Message                          |
+-------+------+----------------------------------+
| Error | 1051 | Unknown table 'xx'               |
| Error |    5 | Unknown table 'xx'               |
+-------+------+----------------------------------+
```

RESIGNAL Requires Condition Handler Context

All forms of RESIGNAL require that the current context be a condition handler. Otherwise, RESIGNAL is illegal and a RESIGNAL when handler not active error occurs. For example:

```
mysql> CREATE PROCEDURE p () RESIGNAL;
Query OK, 0 rows affected (0.00 sec)

mysql> CALL p();
ERROR 1645 (0K000): RESIGNAL when handler not active
```

Here is a more difficult example:

```
delimiter //
CREATE FUNCTION f () RETURNS INT
BEGIN
  RESIGNAL;
  RETURN 5;
END//
CREATE PROCEDURE p ()
BEGIN
  DECLARE EXIT HANDLER FOR SQLEXCEPTION SET @a=f();
  SIGNAL SQLSTATE '55555';
END//
delimiter ;
CALL p();
```

RESIGNAL occurs within the stored function f(). Although f() itself is invoked within the context of the EXIT handler, execution within f() has its own context, which is not handler context. Thus, RESIGNAL within f() results in a “handler not active” error.

13.6.7.5 SIGNAL Syntax

```
SIGNAL condition_value
    [SET signal_information_item
    [, signal_information_item] ...]

condition_value:
    SQLSTATE [VALUE] sqlstate_value
  | condition_name

signal_information_item:
    condition_information_item_name = simple_value_specification

condition_information_item_name:
    CLASS_ORIGIN
  | SUBCLASS_ORIGIN
  | MESSAGE_TEXT
  | MYSQL_ERRNO
  | CONSTRAINT_CATALOG
  | CONSTRAINT_SCHEMA
  | CONSTRAINT_NAME
  | CATALOG_NAME
  | SCHEMA_NAME
  | TABLE_NAME
  | COLUMN_NAME
  | CURSOR_NAME

condition_name, simple_value_specification:
    (see following discussion)
```

SIGNAL is the way to “return” an error. SIGNAL provides error information to a handler, to an outer portion of the application, or to the client. Also, it provides control over the error's characteristics (error number, SQLSTATE value, message). Without SIGNAL, it is necessary to resort to workarounds such as deliberately referring to a nonexistent table to cause a routine to return an error.

No special privileges are required to execute the SIGNAL statement.

To retrieve information from the diagnostics area, use the GET DIAGNOSTICS statement (see Section 13.6.7.3, “GET DIAGNOSTICS Syntax”). For information about the diagnostics area, see Section 13.6.7.7, “The MySQL Diagnostics Area”.

The *condition_value* in a SIGNAL statement indicates the error value to be returned. It can be an SQLSTATE value (a 5-character string literal) or a *condition_name* that refers to a named condition previously defined with `DECLARE ... CONDITION` (see Section 13.6.7.1, “`DECLARE ... CONDITION` Syntax”).

An `SQLSTATE` value can indicate errors, warnings, or “not found.” The first two characters of the value indicate its error class, as discussed in Signal Condition Information Items. Some signal values cause statement termination; see Effect of Signals on Handlers, Cursors, and Statements.

The `SQLSTATE` value for a `SIGNAL` statement should not start with `'00'` because such values indicate success and are not valid for signaling an error. This is true whether the `SQLSTATE` value is specified directly in the `SIGNAL` statement or in a named condition referred to in the statement. If the value is invalid, a `Bad SQLSTATE` error occurs.

To signal a generic `SQLSTATE` value, use `'45000'`, which means “unhandled user-defined exception.”

The `SIGNAL` statement optionally includes a `SET` clause that contains multiple signal items, in a comma-separated list of *`condition_information_item_name`* = *`simple_value_specification`* assignments.

Each *`condition_information_item_name`* may be specified only once in the `SET` clause. Otherwise, a `Duplicate condition information item` error occurs.

Valid *`simple_value_specification`* designators can be specified using stored procedure or function parameters, stored program local variables declared with `DECLARE`, user-defined variables, system variables, or literals. A character literal may include a *`_charset`* introducer.

For information about permissible *`condition_information_item_name`* values, see Signal Condition Information Items.

The following procedure signals an error or warning depending on the value of `pval`, its input parameter:

```
CREATE PROCEDURE p (pval INT)
BEGIN
  DECLARE specialty CONDITION FOR SQLSTATE '45000';
  IF pval = 0 THEN
    SIGNAL SQLSTATE '01000';
  ELSEIF pval = 1 THEN
    SIGNAL SQLSTATE '45000'
      SET MESSAGE_TEXT = 'An error occurred';
  ELSEIF pval = 2 THEN
    SIGNAL specialty
      SET MESSAGE_TEXT = 'An error occurred';
  ELSE
    SIGNAL SQLSTATE '01000'
      SET MESSAGE_TEXT = 'A warning occurred', MYSQL_ERRNO = 1000;
    SIGNAL SQLSTATE '45000'
      SET MESSAGE_TEXT = 'An error occurred', MYSQL_ERRNO = 1001;
  END IF;
END;
```

If `pval` is 0, `p()` signals a warning because `SQLSTATE` values that begin with `'01'` are signals in the warning class. The warning does not terminate the procedure, and can be seen with `SHOW WARNINGS` after the procedure returns.

If `pval` is 1, `p()` signals an error and sets the `MESSAGE_TEXT` condition information item. The error terminates the procedure, and the text is returned with the error information.

If `pval` is 2, the same error is signaled, although the `SQLSTATE` value is specified using a named condition in this case.

If pval is anything else, p() first signals a warning and sets the message text and error number condition information items. This warning does not terminate the procedure, so execution continues and p() then signals an error. The error does terminate the procedure. The message text and error number set by the warning are replaced by the values set by the error, which are returned with the error information.

SIGNAL is typically used within stored programs, but it is a MySQL extension that it is permitted outside handler context. For example, if you invoke the mysql client program, you can enter any of these statements at the prompt:

```
mysql> SIGNAL SQLSTATE '77777';
mysql> CREATE TRIGGER t_bi BEFORE INSERT ON t
    -> FOR EACH ROW SIGNAL SQLSTATE '77777';
mysql> CREATE EVENT e ON SCHEDULE EVERY 1 SECOND
    -> DO SIGNAL SQLSTATE '77777';
```

SIGNAL executes according to the following rules:

If the SIGNAL statement indicates a particular SQLSTATE value, that value is used to signal the condition specified. Example:

```
CREATE PROCEDURE p (divisor INT)
BEGIN
  IF divisor = 0 THEN
    SIGNAL SQLSTATE '22012';
  END IF;
END;
```

If the SIGNAL statement uses a named condition, the condition must be declared in some scope that applies to the SIGNAL statement, and must be defined using an SQLSTATE value, not a MySQL error number. Example:

```
CREATE PROCEDURE p (divisor INT)
BEGIN
  DECLARE divide_by_zero CONDITION FOR SQLSTATE '22012';
  IF divisor = 0 THEN
    SIGNAL divide_by_zero;
  END IF;
END;
```

If the named condition does not exist in the scope of the SIGNAL statement, an Undefined CONDITION error occurs.

If SIGNAL refers to a named condition that is defined with a MySQL error number rather than an SQLSTATE value, a SIGNAL/RESIGNAL can only use a CONDITION defined with SQLSTATE error occurs. The following statements cause that error because the named condition is associated with a MySQL error number:

```
DECLARE no_such_table CONDITION FOR 1051;
SIGNAL no_such_table;
```

If a condition with a given name is declared multiple times in different scopes, the declaration with the most local scope applies. Consider the following procedure:

```
CREATE PROCEDURE p (divisor INT)
BEGIN
  DECLARE my_error CONDITION FOR SQLSTATE '45000';
  IF divisor = 0 THEN
```

```
    BEGIN
      DECLARE my_error CONDITION FOR SQLSTATE '22012';
      SIGNAL my_error;
    END;
  END IF;
  SIGNAL my_error;
END;
```

If divisor is 0, the first SIGNAL statement executes. The innermost my_error condition declaration applies, raising SQLSTATE '22012'.

If divisor is not 0, the second SIGNAL statement executes. The outermost my_error condition declaration applies, raising SQLSTATE '45000'.

For information about how the server chooses handlers when a condition occurs, see Section 13.6.7.6, “Scope Rules for Handlers”.

Signals can be raised within exception handlers:

```
CREATE PROCEDURE p ()
BEGIN
  DECLARE EXIT HANDLER FOR SQLEXCEPTION
  BEGIN
    SIGNAL SQLSTATE VALUE '99999'
      SET MESSAGE_TEXT = 'An error occurred';
  END;
  DROP TABLE no_such_table;
END;
```

CALL p() reaches the DROP TABLE statement. There is no table named no_such_table, so the error handler is activated. The error handler destroys the original error (“no such table”) and makes a new error with SQLSTATE '99999' and message An error occurred.

Signal Condition Information Items

The following table lists the names of diagnostics area condition information items that can be set in a SIGNAL (or RESIGNAL) statement. All items are standard SQL except MYSQL_ERRNO, which is a MySQL extension. For more information about these items see Section 13.6.7.7, “The MySQL Diagnostics Area”.

```
Item Name             Definition
---------             ----------
CLASS_ORIGIN          VARCHAR(64)
SUBCLASS_ORIGIN       VARCHAR(64)
CONSTRAINT_CATALOG    VARCHAR(64)
CONSTRAINT_SCHEMA     VARCHAR(64)
CONSTRAINT_NAME       VARCHAR(64)
CATALOG_NAME          VARCHAR(64)
SCHEMA_NAME           VARCHAR(64)
TABLE_NAME            VARCHAR(64)
COLUMN_NAME           VARCHAR(64)
CURSOR_NAME           VARCHAR(64)
MESSAGE_TEXT          VARCHAR(128)
MYSQL_ERRNO           SMALLINT UNSIGNED
```

The character set for character items is UTF-8.

It is illegal to assign NULL to a condition information item in a SIGNAL statement.

A SIGNAL statement always specifies an SQLSTATE value, either directly, or indirectly by referring to a named condition defined with an SQLSTATE value. The first two characters of an SQLSTATE value are its class, and the class determines the default value for the condition information items:

- Class = '00' (success)

  Illegal. SQLSTATE values that begin with '00' indicate success and are not valid for SIGNAL.

- Class = '01' (warning)

  ```
  MESSAGE_TEXT = 'Unhandled user-defined warning condition';
  MYSQL_ERRNO = ER_SIGNAL_WARN
  ```

- Class = '02' (not found)

  ```
  MESSAGE_TEXT = 'Unhandled user-defined not found condition';
  MYSQL_ERRNO = ER_SIGNAL_NOT_FOUND
  ```

- Class > '02' (exception)

  ```
  MESSAGE_TEXT = 'Unhandled user-defined exception condition';
  MYSQL_ERRNO = ER_SIGNAL_EXCEPTION
  ```

For legal classes, the other condition information items are set as follows:

```
CLASS_ORIGIN = SUBCLASS_ORIGIN = '';
CONSTRAINT_CATALOG = CONSTRAINT_SCHEMA = CONSTRAINT_NAME = '';
CATALOG_NAME = SCHEMA_NAME = TABLE_NAME = COLUMN_NAME = '';
CURSOR_NAME = '';
```

The error values that are accessible after SIGNAL executes are the SQLSTATE value raised by the SIGNAL statement and the MESSAGE_TEXT and MYSQL_ERRNO items. These values are available from the C API:

- SQLSTATE value: Call mysql_sqlstate()
- MYSQL_ERRNO value: Call mysql_errno()
- MESSAGE_TEXT value: Call mysql_error()

From SQL, the output from SHOW WARNINGS and SHOW ERRORS indicates the MYSQL_ERRNO and MESSAGE_TEXT values in the Code and Message columns.

To retrieve information from the diagnostics area, use the GET DIAGNOSTICS statement (see Section 13.6.7.3, “GET DIAGNOSTICS Syntax”). For information about the diagnostics area, see Section 13.6.7.7, “The MySQL Diagnostics Area”.

Effect of Signals on Handlers, Cursors, and Statements

Signals have different effects on statement execution depending on the signal class. The class determines how severe an error is. MySQL ignores the value of the sql_mode system variable; in particular, strict SQL mode does not matter. MySQL also ignores IGNORE: The intent of SIGNAL is to raise a user-generated error explicitly, so a signal is never ignored.

In the following descriptions, “unhandled” means that no handler for the signaled SQLSTATE value has been defined with DECLARE ... HANDLER.

- Class = '00' (success)

  Illegal. SQLSTATE values that begin with '00' indicate success and are not valid for SIGNAL.

- Class = '01' (warning)

  The value of the warning_count system variable goes up. SHOW WARNINGS shows the signal. SQLWARNING handlers catch the signal. If the signal is unhandled in a function, statements do not end.

- Class = '02' (not found)

  NOT FOUND handlers catch the signal. There is no effect on cursors. If the signal is unhandled in a function, statements end.

- Class > '02' (exception)

  SQLEXCEPTION handlers catch the signal. If the signal is unhandled in a function, statements end.

- Class = '40'

  Treated as an ordinary exception.

Example:

```
mysql> delimiter //
mysql> CREATE FUNCTION f () RETURNS INT
    -> BEGIN
    ->   SIGNAL SQLSTATE '01234';  -- signal a warning
    ->   RETURN 5;
    -> END//
mysql> delimiter ;
mysql> CREATE TABLE t (s1 INT);
mysql> INSERT INTO t VALUES (f());
```

The result is that a row containing 5 is inserted into table t. The warning that is signaled can be viewed with SHOW WARNINGS.

13.6.7.6 Scope Rules for Handlers

A stored program may include handlers to be invoked when certain conditions occur within the program. The applicability of each handler depends on its location within the program definition and on the condition or conditions that it handles:

- A handler declared in a BEGIN ... END block is in scope only for the SQL statements following the handler declarations in the block. If the handler itself raises a condition, it cannot handle that condition, nor can any other handlers declared in the block. In the following example, handlers H1 and H2 are in scope for conditions raised by statements *stmt1* and *stmt2*. But neither H1 nor H2 are in scope for conditions raised in the body of H1 or H2.

  ```
  BEGIN -- outer block
    DECLARE EXIT HANDLER FOR ...;  -- handler H1
    DECLARE EXIT HANDLER FOR ...;  -- handler H2
    stmt1;
    stmt2;
  END;
  ```

- A handler is in scope only for the block in which it is declared, and cannot be activated for conditions occurring outside that block. In the following example, handler H1 is in scope for *stmt1* in the inner block, but not for *stmt2* in the outer block:

  ```
  BEGIN -- outer block
  ```

```
  BEGIN -- inner block
    DECLARE EXIT HANDLER FOR ...;  -- handler H1
    stmt1;
  END;
  stmt2;
END;
```

- A handler can be specific or general. A specific handler is for a MySQL error code, SQLSTATE value, or condition name. A general handler is for a condition in the SQLWARNING, SQLEXCEPTION, or NOT FOUND class. Condition specificity is related to condition precedence, as described later.

Multiple handlers can be declared in different scopes and with different specificities. For example, there might be a specific MySQL error code handler in an outer block, and a general SQLWARNING handler in an inner block. Or there might be handlers for a specific MySQL error code and the general SQLWARNING class in the same block.

Whether a handler is activated depends not only on its own scope and condition value, but on what other handlers are present. When a condition occurs in a stored program, the server searches for applicable handlers in the current scope (current BEGIN ... END block). If there are no applicable handlers, the search continues outward with the handlers in each successive containing scope (block). When the server finds one or more applicable handlers at a given scope, it chooses among them based on condition precedence:

- A MySQL error code handler takes precedence over an SQLSTATE value handler.
- An SQLSTATE value handler takes precedence over general SQLWARNING, SQLEXCEPTION, or NOT FOUND handlers.
- An SQLEXCEPTION handler takes precedence over an SQLWARNING handler.
- The precedence of NOT FOUND depends on how the condition is raised:
  - Normally, a condition in the NOT FOUND class can be handled by an SQLWARNING or NOT FOUND handler, with the SQLWARNING handler taking precedence if both are present. Normal occurrence of NOT FOUND takes place when a cursor used to fetch a set of rows reaches the end of the data set, or for instances of SELECT ... INTO *var_list* such that the WHERE clause finds no rows.
  - If a NOT FOUND condition is raised by a SIGNAL (or RESIGNAL) statement, the condition can be handled by a NOT FOUND handler but not an SQLWARNING handler.
- It is possible to have several applicable handlers with the same precedence. For example, a statement could generate multiple warnings with different error codes, for each of which an error-specific handler exists. In this case, the choice of which handler the server activates is indeterminate, and may change depending on the circumstances under which the condition occurs.

One implication of the handler selection rules is that if multiple applicable handlers occur in different scopes, handlers with the most local scope take precedence over handlers in outer scopes, even over those for more specific conditions.

If there is no appropriate handler when a condition occurs, the action taken depends on the class of the condition:

- For SQLEXCEPTION conditions, the stored program terminates at the statement that raised the condition, as if there were an EXIT handler. If the program was called by another stored program, the calling program handles the condition using the handler selection rules applied to its own handlers.
- For SQLWARNING conditions, the program continues executing, as if there were a CONTINUE handler.

- For NOT FOUND conditions, if the condition was raised normally, the action is CONTINUE. If it was raised by SIGNAL or RESIGNAL, the action is EXIT.

The following examples demonstrate how MySQL applies the handler selection rules.

This procedure contains two handlers, one for the specific SQLSTATE value ('42S02') that occurs for attempts to drop a nonexistent table, and one for the general SQLEXCEPTION class:

```
CREATE PROCEDURE p1()
BEGIN
  DECLARE CONTINUE HANDLER FOR SQLSTATE '42S02'
    SELECT 'SQLSTATE handler was activated' AS msg;
  DECLARE CONTINUE HANDLER FOR SQLEXCEPTION
    SELECT 'SQLEXCEPTION handler was activated' AS msg;

  DROP TABLE test.t;
END;
```

Both handlers are declared in the same block and have the same scope. However, SQLSTATE handlers take precedence over SQLEXCEPTION handlers, so if the table t is nonexistent, the DROP TABLE statement raises a condition that activates the SQLSTATE handler:

```
mysql> CALL p1();
+--------------------------------+
| msg                            |
+--------------------------------+
| SQLSTATE handler was activated |
+--------------------------------+
```

This procedure contains the same two handlers. But this time, the DROP TABLE statement and SQLEXCEPTION handler are in an inner block relative to the SQLSTATE handler:

```
CREATE PROCEDURE p2()
BEGIN -- outer block
    DECLARE CONTINUE HANDLER FOR SQLSTATE '42S02'
      SELECT 'SQLSTATE handler was activated' AS msg;
  BEGIN -- inner block
    DECLARE CONTINUE HANDLER FOR SQLEXCEPTION
      SELECT 'SQLEXCEPTION handler was activated' AS msg;

    DROP TABLE test.t; -- occurs within inner block
  END;
END;
```

In this case, the handler that is more local to where the condition occurs takes precedence. The SQLEXCEPTION handler activates, even though it is more general than the SQLSTATE handler:

```
mysql> CALL p2();
+------------------------------------+
| msg                                |
+------------------------------------+
| SQLEXCEPTION handler was activated |
+------------------------------------+
```

In this procedure, one of the handlers is declared in a block inner to the scope of the DROP TABLE statement:

```
CREATE PROCEDURE p3()
```

```
BEGIN -- outer block
  DECLARE CONTINUE HANDLER FOR SQLEXCEPTION
    SELECT 'SQLEXCEPTION handler was activated' AS msg;
  BEGIN -- inner block
    DECLARE CONTINUE HANDLER FOR SQLSTATE '42S02'
      SELECT 'SQLSTATE handler was activated' AS msg;
  END;

  DROP TABLE test.t; -- occurs within outer block
END;
```

Only the `SQLEXCEPTION` handler applies because the other one is not in scope for the condition raised by the `DROP TABLE`:

```
mysql> CALL p3();
+------------------------------------+
| msg                                |
+------------------------------------+
| SQLEXCEPTION handler was activated |
+------------------------------------+
```

In this procedure, both handlers are declared in a block inner to the scope of the `DROP TABLE` statement:

```
CREATE PROCEDURE p4()
BEGIN -- outer block
  BEGIN -- inner block
    DECLARE CONTINUE HANDLER FOR SQLEXCEPTION
      SELECT 'SQLEXCEPTION handler was activated' AS msg;
    DECLARE CONTINUE HANDLER FOR SQLSTATE '42S02'
      SELECT 'SQLSTATE handler was activated' AS msg;
  END;

  DROP TABLE test.t; -- occurs within outer block
END;
```

Neither handler applies because they are not in scope for the `DROP TABLE`. The condition raised by the statement goes unhandled and terminates the procedure with an error:

```
mysql> CALL p4();
ERROR 1051 (42S02): Unknown table 'test.t'
```

13.6.7.7 The MySQL Diagnostics Area

SQL statements produce diagnostic information that populates the diagnostics area. Standard SQL has a diagnostics area stack, containing a diagnostics area for each nested execution context. Standard SQL also supports `GET STACKED DIAGNOSTICS` syntax for referring to the second diagnostics area during condition handler execution. MySQL supports the `STACKED` keyword as of MySQL 5.7. Before that, MySQL does not support `STACKED`; there is a single diagnostics area containing information from the most recent statement that wrote to it.

This section describes the structure of the diagnostics area in MySQL, the information items recognized by MySQL, how statements clear and set the diagnostics area, and how diagnostics areas are pushed to and popped from the stack.

Diagnostics Area Structure

The diagnostics area contains two kinds of information:

- Statement information, such as the number of conditions that occurred or the affected-rows count.

- Condition information, such as the error code and message. If a statement raises multiple conditions, this part of the diagnostics area has a condition area for each one. If a statement raises no conditions, this part of the diagnostics area is empty.

For a statement that produces three conditions, the diagnostics area contains statement and condition information like this:

```
Statement information:
  row count
  ... other statement information items ...
Condition area list:
  Condition area 1:
    error code for condition 1
    error message for condition 1
    ... other condition information items ...
  Condition area 2:
    error code for condition 2:
    error message for condition 2
    ... other condition information items ...
  Condition area 3:
    error code for condition 3
    error message for condition 3
    ... other condition information items ...
```

Diagnostics Area Information Items

The diagnostics area contains statement and condition information items. Numeric items are integers. The character set for character items is UTF-8. No item can be NULL. If a statement or condition item is not set by a statement that populates the diagnostics area, its value is 0 or the empty string, depending on the item data type.

The statement information part of the diagnostics area contains these items:

- NUMBER: An integer indicating the number of condition areas that have information.

- ROW_COUNT: An integer indicating the number of rows affected by the statement. ROW_COUNT has the same value as the ROW_COUNT() function (see Section 12.14, “Information Functions”).

The condition information part of the diagnostics area contains a condition area for each condition. Condition areas are numbered from 1 to the value of the NUMBER statement condition item. If NUMBER is 0, there are no condition areas.

Each condition area contains the items in the following list. All items are standard SQL except MYSQL_ERRNO, which is a MySQL extension. The definitions apply for conditions generated other than by a signal (that is, by a SIGNAL or RESIGNAL statement). For nonsignal conditions, MySQL populates only those condition items not described as always empty. The effects of signals on the condition area are described later.

- CLASS_ORIGIN: A string containing the class of the RETURNED_SQLSTATE value. If the RETURNED_SQLSTATE value begins with a class value defined in SQL standards document ISO 9075-2 (section 24.1, SQLSTATE), CLASS_ORIGIN is 'ISO 9075'. Otherwise, CLASS_ORIGIN is 'MySQL'.

- SUBCLASS_ORIGIN: A string containing the subclass of the RETURNED_SQLSTATE value. If CLASS_ORIGIN is 'ISO 9075' or RETURNED_SQLSTATE ends with '000', SUBCLASS_ORIGIN is 'ISO 9075'. Otherwise, SUBCLASS_ORIGIN is 'MySQL'.

- RETURNED_SQLSTATE: A string that indicates the SQLSTATE value for the condition.

- MESSAGE_TEXT: A string that indicates the error message for the condition.

- MYSQL_ERRNO: An integer that indicates the MySQL error code for the condition.

- CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, CONSTRAINT_NAME: Strings that indicate the catalog, schema, and name for a violated constraint. They are always empty.

- CATALOG_NAME, SCHEMA_NAME, TABLE_NAME, COLUMN_NAME: Strings that indicate the catalog, schema, table, and column related to the condition. They are always empty.

- CURSOR_NAME: A string that indicates the cursor name. This is always empty.

For the RETURNED_SQLSTATE, MESSAGE_TEXT, and MYSQL_ERRNO values for particular errors, see Section C.3, “Server Error Codes and Messages”.

If a SIGNAL (or RESIGNAL) statement populates the diagnostics area, its SET clause can assign to any condition information item except RETURNED_SQLSTATE any value that is legal for the item data type. SIGNAL also sets the RETURNED_SQLSTATE value, but not directly in its SET clause. That value comes from the SIGNAL statement SQLSTATE argument.

SIGNAL also sets statement information items. It sets NUMBER to 1. It sets ROW_COUNT to –1 for errors and 0 otherwise.

How the Diagnostics Area is Populated

Nondiagnostic SQL statements populate the diagnostics area automatically, and its contents can be set explicitly with the SIGNAL and RESIGNAL statements. The diagnostics area can be examined with GET DIAGNOSTICS to extract specific items, or with SHOW WARNINGS or SHOW ERRORS to see conditions or errors.

SQL statements clear and set the diagnostics area as follows:

- When the server starts executing a statement after parsing it, it clears the diagnostics area for nondiagnostic statements. (Before MySQL 5.7.2, the server clears the diagnostics area for nondiagnostic statements that use tables.) Diagnostic statements do not clear the diagnostics area (SHOW WARNINGS, SHOW ERRORS, GET DIAGNOSTICS).

- If a statement raises a condition, the diagnostics area is cleared of conditions that belong to earlier statements. The exception is that conditions raised by GET DIAGNOSTICS and RESIGNAL are added to the diagnostics area without clearing it.

Thus, even a statement that does not normally clear the diagnostics area when it begins executing clears it if the statement raises a condition.

The following example shows the effect of various statements on the diagnostics area, using SHOW WARNINGS to display information about conditions stored there.

This DROP TABLE statement clears the diagnostics area and populates it when the condition occurs:

```
mysql> DROP TABLE IF EXISTS test.no_such_table;
Query OK, 0 rows affected, 1 warning (0.01 sec)

mysql> SHOW WARNINGS;
+-------+------+------------------------------------+
| Level | Code | Message                            |
+-------+------+------------------------------------+
| Note  | 1051 | Unknown table 'test.no_such_table' |
+-------+------+------------------------------------+
1 row in set (0.00 sec)
```

This SET statement generates an error, so it clears and populates the diagnostics area:

```
mysql> SET @x = @@x;
ERROR 1193 (HY000): Unknown system variable 'x'

mysql> SHOW WARNINGS;
+-------+------+-----------------------------+
| Level | Code | Message                     |
+-------+------+-----------------------------+
| Error | 1193 | Unknown system variable 'x' |
+-------+------+-----------------------------+
1 row in set (0.00 sec)
```

The previous SET statement produced a single condition, so 1 is the only valid condition number for GET DIAGNOSTICS at this point. The following statement uses a condition number of 2, which produces a warning that is added to the diagnostics area without clearing it:

```
mysql> GET DIAGNOSTICS CONDITION 2 @p = MESSAGE_TEXT;
Query OK, 0 rows affected, 1 warning (0.00 sec)

mysql> SHOW WARNINGS;
+-------+------+------------------------------+
| Level | Code | Message                      |
+-------+------+------------------------------+
| Error | 1193 | Unknown system variable 'xx' |
| Error | 1753 | Invalid condition number     |
+-------+------+------------------------------+
2 rows in set (0.00 sec)
```

Now there are two conditions in the diagnostics area, so the same GET DIAGNOSTICS statement succeeds:

```
mysql> GET DIAGNOSTICS CONDITION 2 @p = MESSAGE_TEXT;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @p;
+--------------------------+
| @p                       |
+--------------------------+
| Invalid condition number |
+--------------------------+
1 row in set (0.01 sec)
```

How the Diagnostics Area Stack Works

When a push to the diagnostics area stack occurs, the first (current) diagnostics area becomes the second (stacked) diagnostics area and a new current diagnostics area is created as a copy of it. Diagnostics areas are pushed to and popped from the stack under the following circumstances:

- Execution of a stored program

  A push occurs before the program executes and a pop occurs afterward. If the stored program ends while handlers are executing, there can be more than one diagnostics area to pop; this occurs due to an exception for which there are no appropriate handlers or due to RETURN in the handler.

  Any warning or error conditions occurring during stored program execution then are added to the current diagnostics area, except that, for triggers, only errors are added. When the stored program ends, the caller sees these conditions in its current diagonstics area.

- Execution of a condition handler within a stored program

When a push occurs as a result of condition handler activation, the stacked diagnostics area is the area that was current within the stored program prior to the push. The new now-current diagnostics area is the handler's current diagnostics area. GET [CURRENT] DIAGNOSTICS and GET STACKED DIAGNOSTICS can be used within the handler to access the contents of the current (handler) and stacked (stored program) diagnostics areas. Initially, they return the same result, but statements executing within the handler modify the current diagnostics area, clearing and setting its contents according to the normal rules (see How the Diagnostics Area is Populated). The stacked diagnostics area cannot be modified by statements executing within the handler except RESIGNAL.

If the handler executes successfully, the current (handler) diagnostics area is popped and the stacked (stored program) diagnostics area again becomes the current diagnostics area. Conditions added to the handler diagnostics area during handler execution are added to the current diagnostics area.

- Execution of RESIGNAL

  The RESIGNAL statement passes on the error condition information that is available during execution of a condition handler within a compound statement inside a stored program. RESIGNAL may change some or all information before passing it on, modifying the diagnostics stack as described in Section 13.6.7.4, “RESIGNAL Syntax”.

Diagnostics Area-Related System Variables

Certain system variables control or are related to some aspects of the diagnostics area:

- max_error_count controls the number of condition areas in the diagnostics area. If more conditions than this occur, MySQL silently discards information for the excess conditions. (Conditions added by RESIGNAL are always added, with older conditions being discarded as necessary to make room.)
- warning_count indicates the number of conditions that occurred. This includes errors, warnings, and notes. Normally, NUMBER and warning_count are the same. However, as the number of conditions generated exceeds max_error_count, the value of warning_count continues to rise whereas NUMBER remains capped at max_error_count because no additional conditions are stored in the diagnostics area.
- error_count indicates the number of errors that occurred. This value includes “not found” and exception conditions, but excludes warnings and notes. Like warning_count, its value can exceed max_error_count.
- If the sql_notes system variable is set to 0, notes are not stored and do not increment warning_count.

Example: If max_error_count is 10, the diagnostics area can contain a maximum of 10 condition areas. Suppose that a statement raises 20 conditions, 12 of which are errors. In that case, the diagnostics area contains the first 10 conditions, NUMBER is 10, warning_count is 20, and error_count is 12.

Changes to the value of max_error_count have no effect until the next attempt to modify the diagnostics area. If the diagnostics area contains 10 condition areas and max_error_count is set to 5, that has no immediate effect on the size or content of the diagnostics area.

13.7 Database Administration Statements

13.7.1 Account Management Statements

MySQL account information is stored in the tables of the mysql database. This database and the access control system are discussed extensively in Chapter 5, *MySQL Server Administration*, which you should consult for additional details.

> **Important**
>
> Some releases of MySQL introduce changes to the structure of the grant tables to add new privileges or features. To make sure that you can take advantage of any new capabilities, update your grant tables to have the current structure whenever you update to a new version of MySQL. See Section 4.4.7, “`mysql_upgrade` — Check and Upgrade MySQL Tables”.

13.7.1.1 `ALTER USER` Syntax

```
ALTER USER user_specification [, user_specification] ...

user_specification:
    user alter_option

alter_option: {
    PASSWORD EXPIRE
  | PASSWORD EXPIRE DEFAULT
  | PASSWORD EXPIRE NEVER
  | PASSWORD EXPIRE INTERVAL N DAY
}
```

The `ALTER USER` statement alters MySQL accounts. To use it, you must have the global `CREATE USER` privilege or the `INSERT` privilege for the `mysql` database.

Each *`user_specification`* clause consists of an account name and an option specifying the action to take for that account regarding its password.

Each account name uses the format described in Section 6.2.3, “Specifying Account Names”. If you specify only the user name part of the account name, a host name part of `'%'` is used.

`ALTER USER` permits the following *`alter_option`* values. The first option manually expires an account password. The others establish password expiration *policy* for the account. They do not expire the password but determine how the server applies automatic expiration to the account (see Section 6.3.6, “Password Expiration Policy”).

- `PASSWORD EXPIRE`

  This option expires the account password. For example:

  ```
  ALTER USER 'jeffrey'@'localhost' PASSWORD EXPIRE;
  ```

- `PASSWORD EXPIRE DEFAULT`

  This option sets the account so that the global expiration policy applies, as specified by the `default_password_lifetime` system variable. For example:

  ```
  ALTER USER 'jeffrey'@'localhost' PASSWORD EXPIRE DEFAULT;
  ```

  This option was added in MySQL 5.7.4.

- `PASSWORD EXPIRE NEVER`

  This option disables password expiration for the account so that its password never expires. For example:

```
ALTER USER 'jeffrey'@'localhost' PASSWORD EXPIRE NEVER;
```

This option was added in MySQL 5.7.4.

- `PASSWORD EXPIRE INTERVAL N DAY`

  This option sets the account password lifetime to *`N`* days. For example, this statement requires the password to be changed every 180 days:

  ```
  ALTER USER 'jeffrey'@'localhost' PASSWORD EXPIRE INTERVAL 180 DAY;
  ```

  This option was added in MySQL 5.7.4.

A client session operates in restricted mode if the account password was expired manually or if the password is considered past its lifetime per the automatic expiration policy. In restricted mode, operations performed in the session result in an error until the user issues a `SET PASSWORD` statement to establish a new account password:

```
mysql> SELECT 1;
ERROR 1820 (HY000): You must SET PASSWORD before executing this statement

mysql> SET PASSWORD = PASSWORD('new_password');
Query OK, 0 rows affected (0.01 sec)

mysql> SELECT 1;
+---+
| 1 |
+---+
| 1 |
+---+
1 row in set (0.00 sec)
```

This restricted mode of operation permits `SET` statements, which is useful if the account password uses a hashing format that requires `old_passwords` to be set to a value different from its default.

It is also possible for an administrative user to reset the account password, but any existing sessions for the account remain restricted. Clients using the account must disconnect and reconnect before statements can be executed successfully.

Note

It is possible to "reset" a password with `SET PASSWORD` by setting it to its current value. As a matter of good policy, it is preferable to choose a different password.

Note

As of MySQL 5.7.3, it is not possible to use `ALTER USER` for anonymous-user accounts. This constraint is imposed because an anonymous user cannot execute `SET PASSWORD` to reset the account password.

13.7.1.2 `CREATE USER` Syntax

```
CREATE USER user_specification [, user_specification] ...

user_specification:
    user
    [
```

```
      | IDENTIFIED WITH auth_plugin [AS 'auth_string']
        IDENTIFIED BY [PASSWORD] 'password'
    ]
```

The CREATE USER statement creates new MySQL accounts. An error occurs for accounts that already exist. To use this statement, you must have the global CREATE USER privilege or the INSERT privilege for the mysql database. For each account, CREATE USER creates a new row in the mysql.user table with no privileges and assigns the account an authentication plugin. Depending on the syntax used, CREATE USER may also assign the account a password.

Each *user_specification* clause consists of an account name and information about how authentication occurs for clients that use the account. This part of CREATE USER syntax is shared with GRANT, so the description here applies to GRANT as well.

Each account name uses the format described in Section 6.2.3, “Specifying Account Names”. For example:

```
CREATE USER 'jeffrey'@'localhost' IDENTIFIED BY 'mypass';
```

If you specify only the user name part of the account name, a host name part of '%' is used.

The server assigns an authentication plugin and password to each account as follows, depending on whether the user specification clause includes IDENTIFIED WITH to specify a plugin or IDENTIFIED BY to specify a password:

- With IDENTIFIED WITH, the server assigns the specified plugin and the account has no password.

- With IDENTIFIED BY, the server assigns the plugin implicitly and assigns the specified password.

- With neither IDENTIFIED WITH nor IDENTIFIED BY, the server assigns the plugin implicitly and the account has no password.

If the account has no password, the Password column in the account's mysql.user table row remains empty, which is insecure. To set the password, use SET PASSWORD. See Section 13.7.1.7, “SET PASSWORD Syntax”.

For implicit plugin assignment, the default plugin becomes the value of the plugin column in the account's mysql.user table row. The default plugin is mysql_native_password unless the default_authentication_plugin system variable is set otherwise.

For client connections that use a given account, the server invokes the authentication plugin assigned to the account and the client must provide credentials as required by the authentication method that the plugin implements. If the server cannot find the plugin, either at account-creation time or connect time, an error occurs

If an account's mysql.user table row has a nonempty plugin column:

- The server authenticates client connection attempts using the named plugin.

- Changes to the account password using SET PASSWORD with PASSWORD() must be made with the old_passwords system variable set to the value required by the authentication plugin, so that PASSWORD() uses the appropriate password hashing method. If the plugin is mysql_old_password, the password can also be changed using SET PASSWORD with OLD_PASSWORD(), which uses pre-4.1 password hashing regardless of the value of old_passwords.

If an account's mysql.user table row has an empty plugin column:

- As of MySQL 5.7.2, the server disables any account with an empty plugin until the DBA assigns a nonempty one. Before MySQL 5.7.2, the server authenticates client connection attempts using the `mysql_native_password` or `mysql_old_password` authentication plugin, depending on the hash format of the password stored in the `Password` column.

- Changes to the account password using `SET PASSWORD` can be made with `PASSWORD()`, with `old_passwords` set to 0 or 1 for 4.1 or pre-4.1 password hashing, respectively, or with `OLD_PASSWORD()`, which uses pre-4.1 password hashing regardless of the value of `old_passwords`.

`CREATE USER` examples:

- To specify an authentication plugin for an account, use `IDENTIFIED WITH` *`auth_plugin`*. The plugin name can be a quoted string literal or an unquoted name. `'`*`auth_string`*`'` is an optional quoted string literal to pass to the plugin. The plugin interprets the meaning of the string, so its format is plugin specific. Consult the documentation for a given plugin for information about the authentication string values it accepts, if any.

```
CREATE USER 'jeffrey'@'localhost' IDENTIFIED WITH mysql_native_password;
```

  The server assigns the given authentication plugin to the account but no password. Clients must provide no password when they connect. However, an account with no password is insecure. To ensure that an account uses a specific authentication plugin and has a password with the corresponding hash format, specify the plugin explicitly with `IDENTIFIED WITH`, then use `SET PASSWORD` to set the password:

```
CREATE USER 'jeffrey'@'localhost' IDENTIFIED WITH mysql_native_password;
SET old_passwords = 0;
SET PASSWORD FOR 'jeffrey'@'localhost' = PASSWORD('mypass');
```

  Changes to the account password using `SET PASSWORD` with `PASSWORD()` must be made with the `old_passwords` system variable set to the value required by the account's authentication plugin, so that `PASSWORD()` uses the appropriate password hashing method. Therefore, to use the `sha256_password` or `mysql_old_password` plugin instead, name that plugin in the `CREATE USER` statement and set `old_passwords` to 2 or 1, respectively, before using `SET PASSWORD`. (Use of `mysql_old_password` is not recommended. It is deprecated and support for it will be removed in a future MySQL release.)

- To specify a password for an account at account-creation time, use `IDENTIFIED BY` with the literal plaintext password value:

```
CREATE USER 'jeffrey'@'localhost' IDENTIFIED BY 'mypass';
```

  The server assigns an authentication plugin to the account implicitly, as described previously, and assigns the given password. Clients must provide the given password when they connect.

  If the implicitly assigned plugin is `mysql_native_password`, the `old_passwords` system variable must be set to 0. Otherwise, `CREATE USER` does not hash the password in the format required by the plugin and an error occurs:

```
mysql> SET old_passwords = 1;
mysql> CREATE USER 'jeffrey'@'localhost' IDENTIFIED BY 'mypass';
ERROR 1827 (HY000): The password hash doesn't have the expected
format. Check if the correct password algorithm is being used with
the PASSWORD() function.

mysql> SET old_passwords = 0;
mysql> CREATE USER 'jeffrey'@'localhost' IDENTIFIED BY 'mypass';
```

```
Query OK, 0 rows affected (0.00 sec)
```

- To avoid specifying the plaintext password if you know its hash value (the value that PASSWORD() would return for the password), specify the hash value preceded by the keyword PASSWORD:

  ```
  CREATE USER 'jeffrey'@'localhost'
  IDENTIFIED BY PASSWORD '*90E462C37378CED12064BB3388827D2BA3A9B689';
  ```

  The server assigns an authentication plugin to the account implicitly, as described previously, and assigns the given password. The password hash must be in the format required by the assigned plugin. Clients must provide the password when they connect.

- To enable the user to connect with no password, include no IDENTIFIED BY clause:

  ```
  CREATE USER 'jeffrey'@'localhost';
  ```

  The server assigns an authentication plugin to the account implicitly, as described previously, but no password. Clients must provide no password when they connect. However, an account with no password is insecure. To avoid this, use SET PASSWORD to set the account password.

As mentioned previously, implicit plugin assignment depends on the default authentication plugin. Permitted values of default_authentication_plugin are mysql_native_plugin and sha256_password, but not mysql_old_password. This means it is not possible to set the default plugin so as to be able to create an account that uses mysql_old_password with CREATE USER ... IDENTIFIED BY syntax. To create an account that uses mysql_old_password, use CREATE USER ... IDENTIFIED WITH to name the plugin explicitly, then set the password:

```
CREATE USER 'jeffrey'@'localhost' IDENTIFIED WITH mysql_old_password;
SET old_passwords = 1;
SET PASSWORD FOR 'jeffrey'@'localhost' = PASSWORD('mypass');
```

However, the preceding procedure is not recommended because mysql_old_password is deprecated.

For additional information about setting passwords and authentication plugins, see Section 6.3.5, “Assigning Account Passwords”, and Section 6.3.8, “Pluggable Authentication”.

⚠ **Important**

CREATE USER may be recorded in server logs or in a history file such as ~/.mysql_history, which means that cleartext passwords may be read by anyone having read access to that information. See Section 6.1.2, “Keeping Passwords Secure”.

⚠ **Important**

Some releases of MySQL introduce changes to the structure of the grant tables to add new privileges or features. To make sure that you can take advantage of any new capabilities, update your grant tables to have the current structure whenever you update to a new version of MySQL. See Section 4.4.7, “mysql_upgrade — Check and Upgrade MySQL Tables”.

13.7.1.3 DROP USER Syntax

```
DROP USER user [, user] ...
```

The DROP USER statement removes one or more MySQL accounts and their privileges. It removes privilege rows for the account from all grant tables. An error occurs for accounts that do not exist. To use this statement, you must have the global CREATE USER privilege or the DELETE privilege for the mysql database.

Each account name uses the format described in Section 6.2.3, “Specifying Account Names”. For example:

```
DROP USER 'jeffrey'@'localhost';
```

If you specify only the user name part of the account name, a host name part of '%' is used.

⚠ **Important**

DROP USER does not automatically close any open user sessions. Rather, in the event that a user with an open session is dropped, the statement does not take effect until that user's session is closed. Once the session is closed, the user is dropped, and that user's next attempt to log in will fail. *This is by design*.

DROP USER does not automatically drop or invalidate databases or objects within them that the old user created. This includes stored programs or views for which the DEFINER attribute names the dropped user. Attempts to access such objects may produce an error if they execute in definer security context. (For information about security context, see Section 18.6, “Access Control for Stored Programs and Views”.)

13.7.1.4 GRANT Syntax

```
GRANT
    priv_type [(column_list)]
      [, priv_type [(column_list)]] ...
    ON [object_type] priv_level
    TO user_specification [, user_specification] ...
    [REQUIRE {NONE | ssl_option [[AND] ssl_option] ...}]
    [WITH with_option ...]

GRANT PROXY ON user_specification
    TO user_specification [, user_specification] ...
    [WITH GRANT OPTION]

object_type:
    TABLE
  | FUNCTION
  | PROCEDURE

priv_level:
    *
  | *.*
  | db_name.*
  | db_name.tbl_name
  | tbl_name
  | db_name.routine_name

user_specification:
    user
    [
      | IDENTIFIED WITH auth_plugin [AS 'auth_string']
        IDENTIFIED BY [PASSWORD] 'password'
    ]

ssl_option:
    SSL
  | X509
  | CIPHER 'cipher'
  | ISSUER 'issuer'
  | SUBJECT 'subject'

with_option:
    GRANT OPTION
  | MAX_QUERIES_PER_HOUR count
  | MAX_UPDATES_PER_HOUR count
  | MAX_CONNECTIONS_PER_HOUR count
  | MAX_USER_CONNECTIONS count
```

The GRANT statement grants privileges to MySQL user accounts. GRANT also serves to specify other account characteristics such as use of secure connections and limits on access to server resources. To use GRANT, you must have the GRANT OPTION privilege, and you must have the privileges that you are granting.

Normally, a database administrator first uses CREATE USER to create an account, then GRANT to define its privileges and characteristics. For example:

```
CREATE USER 'jeffrey'@'localhost' IDENTIFIED BY 'mypass';
GRANT ALL ON db1.* TO 'jeffrey'@'localhost';
GRANT SELECT ON db2.invoice TO 'jeffrey'@'localhost';
GRANT USAGE ON *.* TO 'jeffrey'@'localhost' WITH MAX_QUERIES_PER_HOUR 90;
```

However, if an account named in a GRANT statement does not already exist, GRANT may create it under the conditions described later in the discussion of the NO_AUTO_CREATE_USER SQL mode.

The REVOKE statement is related to GRANT and enables administrators to remove account privileges. See Section 13.7.1.6, "REVOKE Syntax".

When successfully executed from the mysql program, GRANT responds with Query OK, 0 rows affected. To determine what privileges result from the operation, use SHOW GRANTS. See Section 13.7.5.20, "SHOW GRANTS Syntax".

There are several aspects to the GRANT statement, described under the following topics in this section:

- Privileges Supported by MySQL
- Global Privileges
- Database Privileges
- Table Privileges
- Column Privileges
- Stored Routine Privileges
- Proxy User Privileges
- Account Names and Passwords
- Other Account Characteristics
- MySQL and Standard SQL Versions of GRANT

⚠ **Important**

Some releases of MySQL introduce changes to the structure of the grant tables to add new privileges or features. To make sure that you can take advantage of any new capabilities, update your grant tables to have the current structure whenever you update to a new version of MySQL. See Section 4.4.7, “mysql_upgrade — Check and Upgrade MySQL Tables”.

Privileges Supported by MySQL

The following table summarizes the permissible *priv_type* privilege types that can be specified for the GRANT and REVOKE statements, and the levels at which each privilege can be granted. For additional information about these privileges, see Section 6.2.1, “Privileges Provided by MySQL”.

Table 13.1 Permissible Privileges for GRANT and REVOKE

Privilege	Meaning and Grantable Levels
ALL [PRIVILEGES]	Grant all privileges at specified access level except GRANT OPTION
ALTER	Enable use of ALTER TABLE. Levels: Global, database, table.
ALTER ROUTINE	Enable stored routines to be altered or dropped. Levels: Global, database, procedure.
CREATE	Enable database and table creation. Levels: Global, database, table.
CREATE ROUTINE	Enable stored routine creation. Levels: Global, database.
CREATE TABLESPACE	Enable tablespaces and log file groups to be created, altered, or dropped. Level: Global.
CREATE TEMPORARY TABLES	Enable use of CREATE TEMPORARY TABLE. Levels: Global, database.
CREATE USER	Enable use of CREATE USER, DROP USER, RENAME USER, and REVOKE ALL PRIVILEGES. Level: Global.
CREATE VIEW	Enable views to be created or altered. Levels: Global, database, table.
DELETE	Enable use of DELETE. Level: Global, database, table.
DROP	Enable databases, tables, and views to be dropped. Levels: Global, database, table.
EVENT	Enable use of events for the Event Scheduler. Levels: Global, database.
EXECUTE	Enable the user to execute stored routines. Levels: Global, database, table.
FILE	Enable the user to cause the server to read or write files. Level: Global.
GRANT OPTION	Enable privileges to be granted to or removed from other accounts. Levels: Global, database, table, procedure, proxy.
INDEX	Enable indexes to be created or dropped. Levels: Global, database, table.
INSERT	Enable use of INSERT. Levels: Global, database, table, column.
LOCK TABLES	Enable use of LOCK TABLES on tables for which you have the SELECT privilege. Levels: Global, database.
PROCESS	Enable the user to see all processes with SHOW PROCESSLIST. Level: Global.
PROXY	Enable user proxying. Level: From user to user.
REFERENCES	Not implemented
RELOAD	Enable use of FLUSH operations. Level: Global.
REPLICATION CLIENT	Enable the user to ask where master or slave servers are. Level: Global.
REPLICATION SLAVE	Enable replication slaves to read binary log events from the master. Level: Global.

Privilege	Meaning and Grantable Levels
SELECT	Enable use of SELECT. Levels: Global, database, table, column.
SHOW DATABASES	Enable SHOW DATABASES to show all databases. Level: Global.
SHOW VIEW	Enable use of SHOW CREATE VIEW. Levels: Global, database, table.
SHUTDOWN	Enable use of mysqladmin shutdown. Level: Global.
SUPER	Enable use of other administrative operations such as CHANGE MASTER TO, KILL, PURGE BINARY LOGS, SET GLOBAL, and mysqladmin debug command. Level: Global.
TRIGGER	Enable trigger operations. Levels: Global, database, table.
UPDATE	Enable use of UPDATE. Levels: Global, database, table, column.
USAGE	Synonym for “no privileges”

A trigger is associated with a table, so to create or drop a trigger, you must have the TRIGGER privilege for the table, not the trigger.

In GRANT statements, the ALL [PRIVILEGES] or PROXY privilege must be named by itself and cannot be specified along with other privileges. ALL [PRIVILEGES] stands for all privileges available for the level at which privileges are to be granted except for the GRANT OPTION and PROXY privileges.

USAGE can be specified to create a user that has no privileges, or to specify the REQUIRE or WITH clauses for an account without changing its existing privileges.

MySQL account information is stored in the tables of the mysql database. This database and the access control system are discussed extensively in Section 6.2, “The MySQL Access Privilege System”, which you should consult for additional details.

If the grant tables hold privilege rows that contain mixed-case database or table names and the lower_case_table_names system variable is set to a nonzero value, REVOKE cannot be used to revoke these privileges. It will be necessary to manipulate the grant tables directly. (GRANT will not create such rows when lower_case_table_names is set, but such rows might have been created prior to setting that variable.)

Privileges can be granted at several levels, depending on the syntax used for the ON clause. For REVOKE, the same ON syntax specifies which privileges to take away. The examples shown here include no IDENTIFIED BY '*password*' clause for brevity, but you should include one if the account does not already exist, to avoid creating an insecure account that has no password.

Global Privileges

Global privileges are administrative or apply to all databases on a given server. To assign global privileges, use ON *.* syntax:

```
GRANT ALL ON *.* TO 'someuser'@'somehost';
GRANT SELECT, INSERT ON *.* TO 'someuser'@'somehost';
```

The CREATE TABLESPACE, CREATE USER, FILE, PROCESS, RELOAD, REPLICATION CLIENT, REPLICATION SLAVE, SHOW DATABASES, SHUTDOWN, and SUPER privileges are administrative and can only be granted globally.

Other privileges can be granted globally or at more specific levels.

MySQL stores global privileges in the mysql.user table.

Database Privileges

Database privileges apply to all objects in a given database. To assign database-level privileges, use `ON` *`db_name`*`.*` syntax:

```
GRANT ALL ON mydb.* TO 'someuser'@'somehost';
GRANT SELECT, INSERT ON mydb.* TO 'someuser'@'somehost';
```

If you use `ON *` syntax (rather than `ON *.*`) and you have selected a default database, privileges are assigned at the database level for the default database. An error occurs if there is no default database.

The `CREATE`, `DROP`, `EVENT`, `GRANT OPTION`, and `LOCK TABLES` privileges can be specified at the database level. Table or routine privileges also can be specified at the database level, in which case they apply to all tables or routines in the database.

MySQL stores database privileges in the `mysql.db` table.

Table Privileges

Table privileges apply to all columns in a given table. To assign table-level privileges, use `ON` *`db_name.tbl_name`* syntax:

```
GRANT ALL ON mydb.mytbl TO 'someuser'@'somehost';
GRANT SELECT, INSERT ON mydb.mytbl TO 'someuser'@'somehost';
```

If you specify *`tbl_name`* rather than *`db_name.tbl_name`*, the statement applies to *`tbl_name`* in the default database. An error occurs if there is no default database.

The permissible *`priv_type`* values at the table level are `ALTER`, `CREATE VIEW`, `CREATE`, `DELETE`, `DROP`, `GRANT OPTION`, `INDEX`, `INSERT`, `SELECT`, `SHOW VIEW`, `TRIGGER`, and `UPDATE`.

MySQL stores table privileges in the `mysql.tables_priv` table.

Column Privileges

Column privileges apply to single columns in a given table. Each privilege to be granted at the column level must be followed by the column or columns, enclosed within parentheses.

```
GRANT SELECT (col1), INSERT (col1,col2) ON mydb.mytbl TO 'someuser'@'somehost';
```

The permissible *`priv_type`* values for a column (that is, when you use a *`column_list`* clause) are `INSERT`, `SELECT`, and `UPDATE`.

MySQL stores column privileges in the `mysql.columns_priv` table.

Stored Routine Privileges

The `ALTER ROUTINE`, `CREATE ROUTINE`, `EXECUTE`, and `GRANT OPTION` privileges apply to stored routines (procedures and functions). They can be granted at the global and database levels. Except for `CREATE ROUTINE`, these privileges can be granted at the routine level for individual routines.

```
GRANT CREATE ROUTINE ON mydb.* TO 'someuser'@'somehost';
GRANT EXECUTE ON PROCEDURE mydb.myproc TO 'someuser'@'somehost';
```

The permissible *`priv_type`* values at the routine level are `ALTER ROUTINE`, `EXECUTE`, and `GRANT OPTION`. `CREATE ROUTINE` is not a routine-level privilege because you must have this privilege to create a routine in the first place.

MySQL stores routine-level privileges in the `mysql.procs_priv` table.

Proxy User Privileges

The `PROXY` privilege enables one user to be a proxy for another. The proxy user impersonates or takes the identity of the proxied user.

```
GRANT PROXY ON 'localuser'@'localhost' TO 'externaluser'@'somehost';
```

When `PROXY` is granted, it must be the only privilege named in the `GRANT` statement, the `REQUIRE` clause cannot be given, and the only permitted `WITH` option is `WITH GRANT OPTION`.

Proxying requires that the proxy user authenticate through a plugin that returns the name of the proxied user to the server when the proxy user connects, and that the proxy user have the `PROXY` privilege for the proxied user. For details and examples, see Section 6.3.10, “Proxy Users”.

MySQL stores proxy privileges in the `mysql.proxies_priv` table.

For the global, database, table, and routine levels, `GRANT ALL` assigns only the privileges that exist at the level you are granting. For example, `GRANT ALL ON db_name.*` is a database-level statement, so it does not grant any global-only privileges such as `FILE`. Granting `ALL` does not assign the `PROXY` privilege.

The *`object_type`* clause, if present, should be specified as `TABLE`, `FUNCTION`, or `PROCEDURE` when the following object is a table, a stored function, or a stored procedure.

The privileges for a database, table, column, or routine are formed additively as the logical `OR` of the privileges at each of the privilege levels. For example, if a user has a global `SELECT` privilege, the privilege cannot be denied by an absence of the privilege at the database, table, or column level. Details of the privilege-checking procedure are presented in Section 6.2.5, “Access Control, Stage 2: Request Verification”.

MySQL enables you to grant privileges on databases or tables that do not exist. For tables, the privileges to be granted must include the `CREATE` privilege. *This behavior is by design*, and is intended to enable the database administrator to prepare user accounts and privileges for databases or tables that are to be created at a later time.

⚠ **Important**

MySQL does not automatically revoke any privileges when you drop a database or table. However, if you drop a routine, any routine-level privileges granted for that routine are revoked.

Account Names and Passwords

The *`user`* value indicates the MySQL account to which the `GRANT` statement applies. To accommodate granting rights to users from arbitrary hosts, MySQL supports specifying the *`user`* value in the form *`user_name@host_name`*. If a *`user_name`* or *`host_name`* value is legal as an unquoted identifier, you need not quote it. However, quotation marks are necessary to specify a *`user_name`* string containing special characters (such as “`-`”), or a *`host_name`* string containing special characters or wildcard characters (such as “`%`”); for example, `'test-user'@'%.com'`. Quote the user name and host name separately.

You can specify wildcards in the host name. For example, *`user_name`*`@'%.example.com'` applies to *`user_name`* for any host in the `example.com` domain, and *`user_name`*`@'192.168.1.%'` applies to *`user_name`* for any host in the `192.168.1` class C subnet.

The simple form *`user_name`* is a synonym for *`user_name`*`@'%'`.

MySQL does not support wildcards in user names. To refer to an anonymous user, specify an account with an empty user name with the `GRANT` statement:

```
GRANT ALL ON test.* TO ''@'localhost' ...
```

In this case, any user who connects from the local host with the correct password for the anonymous user will be permitted access, with the privileges associated with the anonymous-user account.

For additional information about user name and host name values in account names, see Section 6.2.3, "Specifying Account Names".

To specify quoted values, quote database, table, column, and routine names as identifiers. Quote user names and host names as identifiers or as strings. Quote passwords as strings. For string-quoting and identifier-quoting guidelines, see Section 9.1.1, "String Literals", and Section 9.2, "Schema Object Names".

The "`_`" and "`%`" wildcards are permitted when specifying database names in `GRANT` statements that grant privileges at the global or database levels. This means, for example, that if you want to use a "`_`" character as part of a database name, you should specify it as "`\_`" in the `GRANT` statement, to prevent the user from being able to access additional databases matching the wildcard pattern; for example, `` GRANT ... ON `foo\_bar`.* TO ... ``.

Warning

If you permit anonymous users to connect to the MySQL server, you should also grant privileges to all local users as *`user_name`*`@localhost`. Otherwise, the anonymous user account for `localhost` in the `mysql.user` table (created during MySQL installation) is used when named users try to log in to the MySQL server from the local machine. For details, see Section 6.2.4, "Access Control, Stage 1: Connection Verification".

To determine whether the preceding warning applies to you, execute the following query, which lists any anonymous users:

```
SELECT Host, User FROM mysql.user WHERE User='';
```

To avoid the problem just described, delete the local anonymous user account using this statement:

```
DROP USER ''@'localhost';
```

`GRANT` supports host names up to 60 characters long. Database, table, column, and routine names can be up to 64 characters. User names can be up to 16 characters.

Warning

The permissible length for user names cannot be changed by altering the `mysql.user` table. Attempting to do so results in unpredictable behavior which may even make it impossible for users to log in to the MySQL server. You should never alter any of the tables in the `mysql` database in any manner whatsoever except by means of the procedure described in Section 4.4.7, "`mysql_upgrade` — Check and Upgrade MySQL Tables".

To indicate how the user should authenticate when connecting to the server, the *`user_specification`* clause may include `IDENTIFIED WITH` to specify an authentication plugin or `IDENTIFIED BY` to specify a password. Syntax of the user specification is the same as for the CREATE USER statement. For details, see Section 13.7.1.2, "CREATE USER Syntax".

When IDENTIFIED BY is present and you have the global grant privilege (GRANT OPTION), the password becomes the new password for the account, even if the account exists and already has a password. Without IDENTIFIED BY, the account password remains unchanged.

If an account named in a GRANT statement does not exist, the action taken depends on the NO_AUTO_CREATE_USER SQL mode:

- If NO_AUTO_CREATE_USER is not enabled, GRANT creates the account. *This is very insecure* unless you specify a nonempty password using IDENTIFIED BY.
- If NO_AUTO_CREATE_USER is enabled, GRANT fails and does not create the account, unless you specify a nonempty password using IDENTIFIED BY or name an authentication plugin using IDENTIFIED WITH.

As of MySQL 5.7.2, if the account already exists, IDENTIFIED WITH is prohibited because it is intended only for use when creating new accounts.

> **Important**
>
> GRANT may be recorded in server logs or in a history file such as ~/.mysql_history, which means that cleartext passwords may be read by anyone having read access to that information. See Section 6.1.2, "Keeping Passwords Secure".

Other Account Characteristics

The WITH clause is used for several purposes:

- To enable a user to grant privileges to other users
- To specify resource limits for a user
- To specify whether and how a user must use secure connections to the server

The WITH GRANT OPTION clause gives the user the ability to give to other users any privileges the user has at the specified privilege level. You should be careful to whom you give the GRANT OPTION privilege because two users with different privileges may be able to combine privileges!

You cannot grant another user a privilege which you yourself do not have; the GRANT OPTION privilege enables you to assign only those privileges which you yourself possess.

Be aware that when you grant a user the GRANT OPTION privilege at a particular privilege level, any privileges the user possesses (or may be given in the future) at that level can also be granted by that user to other users. Suppose that you grant a user the INSERT privilege on a database. If you then grant the SELECT privilege on the database and specify WITH GRANT OPTION, that user can give to other users not only the SELECT privilege, but also INSERT. If you then grant the UPDATE privilege to the user on the database, the user can grant INSERT, SELECT, and UPDATE.

For a nonadministrative user, you should not grant the ALTER privilege globally or for the mysql database. If you do that, the user can try to subvert the privilege system by renaming tables!

For additional information about security risks associated with particular privileges, see Section 6.2.1, "Privileges Provided by MySQL".

Several `WITH` clause options specify limits on use of server resources by an account:

- The `MAX_QUERIES_PER_HOUR` *`count`*, `MAX_UPDATES_PER_HOUR` *`count`*, and `MAX_CONNECTIONS_PER_HOUR` *`count`* limits restrict the number of queries, updates, and connections to the server permitted to this account during any given one-hour period. (Queries for which results are served from the query cache do not count against the `MAX_QUERIES_PER_HOUR` limit.) If *`count`* is `0` (the default), this means that there is no limitation for the account.

- The `MAX_USER_CONNECTIONS` *`count`* limit restricts the maximum number of simultaneous connections to the server by the account. A nonzero *`count`* specifies the limit for the account explicitly. If *`count`* is `0` (the default), the server determines the number of simultaneous connections for the account from the global value of the `max_user_connections` system variable. If `max_user_connections` is also zero, there is no limit for the account.

To specify resource limits for an existing user without affecting existing privileges, use `GRANT USAGE` at the global level (`ON *.*`) and name the limits to be changed. For example:

```
GRANT USAGE ON *.* TO ...
  WITH MAX_QUERIES_PER_HOUR 500 MAX_UPDATES_PER_HOUR 100;
```

Limits not specified retain their current values.

For more information on restricting access to server resources, see Section 6.3.4, “Setting Account Resource Limits”.

MySQL can check X509 certificate attributes in addition to the usual authentication that is based on the user name and password. To specify SSL-related options for a MySQL account, use the `REQUIRE` clause of the `GRANT` statement. (For background information on the use of SSL with MySQL, see Section 6.3.11, “Using SSL for Secure Connections”.)

There are a number of different possibilities for limiting connection types for a given account:

- `REQUIRE NONE` indicates that the account has no SSL or X509 requirements. This is the default if no SSL-related `REQUIRE` options are specified. Unencrypted connections are permitted if the user name and password are valid. However, encrypted connections can also be used, at the client's option, if the client has the proper certificate and key files. That is, the client need not specify any SSL command options, in which case the connection will be unencrypted. To use an encrypted connection, the client must specify either the `--ssl-ca` option, or all three of the `--ssl-ca`, `--ssl-key`, and `--ssl-cert` options.

- The `REQUIRE SSL` option tells the server to permit only SSL-encrypted connections for the account.

  ```
  GRANT ALL PRIVILEGES ON test.* TO 'root'@'localhost'
    IDENTIFIED BY 'goodsecret' REQUIRE SSL;
  ```

  To connect, the client must specify the `--ssl-ca` option to authenticate the server certificate, and may additionally specify the `--ssl-key` and `--ssl-cert` options. If neither `--ssl-ca` option nor `--ssl-capath` option is specified, the client does not authenticate the server certificate.

- `REQUIRE X509` means that the client must have a valid certificate but that the exact certificate, issuer, and subject do not matter. The only requirement is that it should be possible to verify its signature with one of the CA certificates.

  ```
  GRANT ALL PRIVILEGES ON test.* TO 'root'@'localhost'
  ```

```
  IDENTIFIED BY 'goodsecret' REQUIRE X509;
```

To connect, the client must specify the `--ssl-ca`, `--ssl-key`, and `--ssl-cert` options. This is also true for `ISSUER` and `SUBJECT` because those `REQUIRE` options imply `X509`.

- `REQUIRE ISSUER 'issuer'` places the restriction on connection attempts that the client must present a valid X509 certificate issued by CA `'issuer'`. If the client presents a certificate that is valid but has a different issuer, the server rejects the connection. Use of X509 certificates always implies encryption, so the `SSL` option is unnecessary in this case.

  ```
  GRANT ALL PRIVILEGES ON test.* TO 'root'@'localhost'
    IDENTIFIED BY 'goodsecret'
    REQUIRE ISSUER '/C=FI/ST=Some-State/L=Helsinki/
      O=MySQL Finland AB/CN=Tonu Samuel/emailAddress=tonu@example.com';
  ```

  The `'issuer'` value should be entered as a single string.

  Note

  If MySQL is linked against a version of OpenSSL older than 0.9.6h, use `Email` rather than `emailAddress` in the `'issuer'` value.

- `REQUIRE SUBJECT 'subject'` places the restriction on connection attempts that the client must present a valid X509 certificate containing the subject *`subject`*. If the client presents a certificate that is valid but has a different subject, the server rejects the connection.

  ```
  GRANT ALL PRIVILEGES ON test.* TO 'root'@'localhost'
    IDENTIFIED BY 'goodsecret'
    REQUIRE SUBJECT '/C=EE/ST=Some-State/L=Tallinn/
      O=MySQL demo client certificate/
      CN=Tonu Samuel/emailAddress=tonu@example.com';
  ```

  The `'subject'` value should be entered as a single string. MySQL does a simple string comparison of this value to the value in the certificate, so lettercase and component ordering must be given exactly as present in the certificate.

  Note

  Regarding `emailAddress`, see the note in the description of `REQUIRE ISSUER`.

- `REQUIRE CIPHER 'cipher'` is needed to ensure that ciphers and key lengths of sufficient strength are used. SSL itself can be weak if old algorithms using short encryption keys are used. Using this option, you can ask that a specific cipher method is used for a connection.

  ```
  GRANT ALL PRIVILEGES ON test.* TO 'root'@'localhost'
    IDENTIFIED BY 'goodsecret'
    REQUIRE CIPHER 'EDH-RSA-DES-CBC3-SHA';
  ```

The `SUBJECT`, `ISSUER`, and `CIPHER` options can be combined in the `REQUIRE` clause like this:

```
GRANT ALL PRIVILEGES ON test.* TO 'root'@'localhost'
  IDENTIFIED BY 'goodsecret'
  REQUIRE SUBJECT '/C=EE/ST=Some-State/L=Tallinn/
    O=MySQL demo client certificate/
    CN=Tonu Samuel/emailAddress=tonu@example.com'
  AND ISSUER '/C=FI/ST=Some-State/L=Helsinki/
    O=MySQL Finland AB/CN=Tonu Samuel/emailAddress=tonu@example.com'
```

```
  AND CIPHER 'EDH-RSA-DES-CBC3-SHA';
```

The order of the options does not matter, but no option can be specified twice. The AND keyword is optional between REQUIRE options.

If you are using table, column, or routine privileges for even one user, the server examines table, column, and routine privileges for all users and this slows down MySQL a bit. Similarly, if you limit the number of queries, updates, or connections for any users, the server must monitor these values.

MySQL and Standard SQL Versions of GRANT

The biggest differences between the MySQL and standard SQL versions of GRANT are:

- MySQL associates privileges with the combination of a host name and user name and not with only a user name.

- Standard SQL does not have global or database-level privileges, nor does it support all the privilege types that MySQL supports.

- MySQL does not support the standard SQL UNDER privilege.

- Standard SQL privileges are structured in a hierarchical manner. If you remove a user, all privileges the user has been granted are revoked. This is also true in MySQL if you use DROP USER. See Section 13.7.1.3, “DROP USER Syntax”.

- In standard SQL, when you drop a table, all privileges for the table are revoked. In standard SQL, when you revoke a privilege, all privileges that were granted based on that privilege are also revoked. In MySQL, privileges can be dropped only with explicit DROP USER or REVOKE statements or by manipulating the MySQL grant tables directly.

- In MySQL, it is possible to have the INSERT privilege for only some of the columns in a table. In this case, you can still execute INSERT statements on the table, provided that you insert values only for those columns for which you have the INSERT privilege. The omitted columns are set to their implicit default values if strict SQL mode is not enabled. In strict mode, the statement is rejected if any of the omitted columns have no default value. (Standard SQL requires you to have the INSERT privilege on all columns.) Section 5.1.7, “Server SQL Modes”, discusses strict mode. Section 11.5, “Data Type Default Values”, discusses implicit default values.

13.7.1.5 RENAME USER Syntax

```
RENAME USER old_user TO new_user
    [, old_user TO new_user] ...
```

The RENAME USER statement renames existing MySQL accounts. An error occurs for old accounts that do not exist or new accounts that already exist. To use this statement, you must have the global CREATE USER privilege or the UPDATE privilege for the mysql database.

Each account name uses the format described in Section 6.2.3, “Specifying Account Names”. For example:

```
RENAME USER 'jeffrey'@'localhost' TO 'jeff'@'127.0.0.1';
```

If you specify only the user name part of the account name, a host name part of '%' is used.

RENAME USER causes the privileges held by the old user to be those held by the new user. However, RENAME USER does not automatically drop or invalidate databases or objects within them that the old user created. This includes stored programs or views for which the DEFINER attribute names the old user. Attempts to access such objects may produce an error if they execute in definer security context. (For information about security context, see Section 18.6, "Access Control for Stored Programs and Views".)

The privilege changes take effect as indicated in Section 6.2.6, "When Privilege Changes Take Effect".

13.7.1.6 REVOKE Syntax

```
REVOKE
    priv_type [(column_list)]
      [, priv_type [(column_list)]] ...
    ON [object_type] priv_level
    FROM user [, user] ...

REVOKE ALL PRIVILEGES, GRANT OPTION
    FROM user [, user] ...

REVOKE PROXY ON user
    FROM user [, user] ...
```

The REVOKE statement enables system administrators to revoke privileges from MySQL accounts. Each account name uses the format described in Section 6.2.3, "Specifying Account Names". For example:

```
REVOKE INSERT ON *.* FROM 'jeffrey'@'localhost';
```

If you specify only the user name part of the account name, a host name part of '%' is used.

For details on the levels at which privileges exist, the permissible *priv_type* and *priv_level* values, and the syntax for specifying users and passwords, see Section 13.7.1.4, "GRANT Syntax"

To use the first REVOKE syntax, you must have the GRANT OPTION privilege, and you must have the privileges that you are revoking.

To revoke all privileges, use the second syntax, which drops all global, database, table, column, and routine privileges for the named user or users:

```
REVOKE ALL PRIVILEGES, GRANT OPTION FROM user [, user] ...
```

To use this REVOKE syntax, you must have the global CREATE USER privilege or the UPDATE privilege for the mysql database.

REVOKE removes privileges, but does not drop mysql.user table entries. To remove a user account entirely, use DROP USER (see Section 13.7.1.3, "DROP USER Syntax") or DELETE.

If the grant tables hold privilege rows that contain mixed-case database or table names and the lower_case_table_names system variable is set to a nonzero value, REVOKE cannot be used to revoke these privileges. It will be necessary to manipulate the grant tables directly. (GRANT will not create such rows when lower_case_table_names is set, but such rows might have been created prior to setting the variable.)

When successfully executed from the mysql program, REVOKE responds with Query OK, 0 rows affected. To determine what privileges result from the operation, use SHOW GRANTS. See Section 13.7.5.20, "SHOW GRANTS Syntax".

13.7.1.7 SET PASSWORD Syntax

```
SET PASSWORD [FOR user] =
    {
        PASSWORD('cleartext password')
      | OLD_PASSWORD('cleartext password')
      | 'encrypted password'
    }
```

The SET PASSWORD statement assigns a password to a MySQL user account:

- With no FOR user clause, this statement sets the password for the current user:

  ```
  SET PASSWORD = PASSWORD('cleartext password');
  ```

  Any client who connects to the server using a nonanonymous account can change the password for that account. To see which account the server authenticated you for, invoke the CURRENT_USER() function:

  ```
  SELECT CURRENT_USER();
  ```

- With a FOR user clause, this statement sets the password for the named account, which must exist:

  ```
  SET PASSWORD FOR 'jeffrey'@'localhost' = PASSWORD('cleartext password');
  ```

  In this case, you must have the UPDATE privilege for the mysql database.

When the read_only system variable is enabled, SET PASSWORD requires the SUPER privilege, in addition to any other required privileges.

If a FOR user clause is given, the account name uses the format described in Section 6.2.3, “Specifying Account Names”. The user value should be given as 'user_name'@'host_name', where 'user_name' and 'host_name' are exactly as listed in the User and Host columns of the account's mysql.user table row. (If you specify only a user name, a host name of '%' is used.) For example, to set the password for an account with User and Host column values of 'bob' and '%.example.org', write the statement like this:

```
SET PASSWORD FOR 'bob'@'%.example.org' = PASSWORD('cleartext password');
```

The password can be specified in these ways:

- Using the PASSWORD() function

  The function argument is the cleartext (unencrypted) password. PASSWORD() hashes the password and returns the encrypted password string.

  The old_passwords system variable value determines the hashing method used by PASSWORD(). If SET PASSWORD rejects the password as not being in the correct format, it may be necessary to change old_passwords to change the hashing method. For example, if the account uses the mysql_native_password plugin, the old_passwords value must be 0:

  ```
  SET old_passwords = 0;
  SET PASSWORD FOR 'jeffrey'@'localhost' = PASSWORD('mypass');
  ```

  If the old_passwords value differs from that required by the authentication plugin, hashed password values returned by PASSWORD() are not acceptable for that plugin and attempts to set the password produce an error. For example:

```
mysql> SET old_passwords = 1;
mysql> SET PASSWORD FOR 'jeffrey'@'localhost' = PASSWORD('mypass');
ERROR 1372 (HY000): Password hash should be a 41-digit hexadecimal number
```

- Using the `OLD_PASSWORD()` function:

  The function argument is the cleartext (unencrypted) password. `OLD_PASSWORD()` hashes the password using pre-4.1 hashing and returns the encrypted password string. This hashing method is appropriate only for accounts that use the `mysql_old_password` authentication plugin.

- Using an already encrypted password string

  The password is specified as a string literal. It must represent the already encrypted password value, in the hash format required by the authentication method used for the account.

The following table shows the permitted values of `old_passwords`, the password hashing method for each value, and which authentication plugins use passwords hashed with each method.

Value	Password Hashing Method	Associated Authentication Plugin
0	MySQL 4.1 native hashing	`mysql_native_password`
1	Pre-4.1 (“old”) hashing	`mysql_old_password`
2	SHA-256 hashing	`sha256_password`

For more information about setting passwords, see Section 6.3.5, “Assigning Account Passwords”

Important

`SET PASSWORD` may be recorded in server logs or in a history file such as `~/.mysql_history`, which means that cleartext passwords may be read by anyone having read access to that information. See Section 6.1.2, “Keeping Passwords Secure”.

Caution

If you are connecting to a MySQL 4.1 or later server using a pre-4.1 client program, do not change your password without first reading Section 6.1.2.4, “Password Hashing in MySQL”. The default password hashing format changed in MySQL 4.1, and if you change your password, it might be stored using a hashing format that pre-4.1 clients cannot generate, thus preventing you from connecting to the server afterward.

If you are using MySQL Replication, be aware that, currently, a password used by a replication slave as part of a `CHANGE MASTER TO` statement is effectively limited to 32 characters in length; if the password is longer, any excess characters are truncated. This is not due to any limit imposed by the MySQL Server generally, but rather is an issue specific to MySQL Replication. (For more information, see Bug #43439.)

13.7.2 Table Maintenance Statements

13.7.2.1 `ANALYZE TABLE` Syntax

```
ANALYZE [NO_WRITE_TO_BINLOG | LOCAL] TABLE
    tbl_name [, tbl_name] ...
```

ANALYZE TABLE analyzes and stores the key distribution for a table. During the analysis, the table is locked with a read lock for InnoDB and MyISAM. This statement works with InnoDB, NDB, and MyISAM tables. For MyISAM tables, this statement is equivalent to using myisamchk --analyze.

For more information on how the analysis works within InnoDB, see Persistent Optimizer Statistics for InnoDB Tables and Section 14.2.6.7, “Limits on InnoDB Tables”. In particular, when you enable the innodb_stats_persistent option, you must run ANALYZE TABLE after loading substantial data into an InnoDB table, or creating a new index for one.

MySQL uses the stored key distribution to decide the order in which tables should be joined when you perform a join on something other than a constant. In addition, key distributions can be used when deciding which indexes to use for a specific table within a query.

This statement requires SELECT and INSERT privileges for the table.

ANALYZE TABLE is supported for partitioned tables, and you can use ALTER TABLE ... ANALYZE PARTITION to analyze one or more partitions; for more information, see Section 13.1.6, “ALTER TABLE Syntax”, and Section 17.3.4, “Maintenance of Partitions”.

In MySQL 5.7.1, gtid_next must be set to AUTOMATIC before issuing this statement. This restriction does not apply in MySQL 5.7.2 or later. (Bug #16062608, Bug #16715809, Bug #69045)

ANALYZE TABLE returns a result set with the following columns.

Column	Value
Table	The table name
Op	Always analyze
Msg_type	status, error, info, note, or warning
Msg_text	An informational message

You can check the stored key distribution with the SHOW INDEX statement. See Section 13.7.5.21, “SHOW INDEX Syntax”.

If the table has not changed since the last ANALYZE TABLE statement, the table is not analyzed again.

By default, the server writes ANALYZE TABLE statements to the binary log so that they replicate to replication slaves. To suppress logging, specify the optional NO_WRITE_TO_BINLOG keyword or its alias LOCAL.

13.7.2.2 CHECK TABLE Syntax

```
CHECK TABLE tbl_name [, tbl_name] ... [option] ...

option = {FOR UPGRADE | QUICK | FAST | MEDIUM | EXTENDED | CHANGED}
```

CHECK TABLE checks a table or tables for errors. CHECK TABLE works for InnoDB, MyISAM, ARCHIVE, and CSV tables. For MyISAM tables, the key statistics are updated as well.

To check a table, you must have some privilege for it.

CHECK TABLE can also check views for problems, such as tables that are referenced in the view definition that no longer exist.

CHECK TABLE is supported for partitioned tables, and you can use ALTER TABLE ... CHECK PARTITION to check one or more partitions; for more information, see Section 13.1.6, "ALTER TABLE Syntax", and Section 17.3.4, "Maintenance of Partitions".

In MySQL 5.7.1, gtid_next must be set to AUTOMATIC before issuing this statement. This restriction does not apply in MySQL 5.7.2 or later. (Bug #16062608, Bug #16715809, Bug #69045)

Output

CHECK TABLE returns a result set with the following columns.

Column	Value
Table	The table name
Op	Always check
Msg_type	status, error, info, note, or warning
Msg_text	An informational message

Note that the statement might produce many rows of information for each checked table. The last row has a Msg_type value of status and the Msg_text normally should be OK. If you don't get OK, or Table is already up to date you should normally run a repair of the table. See Section 7.6, "MyISAM Table Maintenance and Crash Recovery". Table is already up to date means that the storage engine for the table indicated that there was no need to check the table.

Checking Version Compatibility

The FOR UPGRADE option checks whether the named tables are compatible with the current version of MySQL. With FOR UPGRADE, the server checks each table to determine whether there have been any incompatible changes in any of the table's data types or indexes since the table was created. If not, the check succeeds. Otherwise, if there is a possible incompatibility, the server runs a full check on the table (which might take some time). If the full check succeeds, the server marks the table's .frm file with the current MySQL version number. Marking the .frm file ensures that further checks for the table with the same version of the server will be fast.

Incompatibilities might occur because the storage format for a data type has changed or because its sort order has changed. Our aim is to avoid these changes, but occasionally they are necessary to correct problems that would be worse than an incompatibility between releases.

Currently, FOR UPGRADE discovers these incompatibilities:

- The indexing order for end-space in TEXT columns for InnoDB and MyISAM tables changed between MySQL 4.1 and 5.0.

- The storage method of the new DECIMAL data type changed between MySQL 5.0.3 and 5.0.5.

- If your table was created by a different version of the MySQL server than the one you are currently running, FOR UPGRADE indicates that the table has an .frm file with an incompatible version. In this case, the result set returned by CHECK TABLE contains a line with a Msg_type value of error and a Msg_text value of Table upgrade required. Please do "REPAIR TABLE `*tbl_name*`" to fix it!

- Changes are sometimes made to character sets or collations that require table indexes to be rebuilt. For details about these changes and when FOR UPGRADE detects them, see Section 2.10.3, "Checking Whether Tables or Indexes Must Be Rebuilt".

- The YEAR(2) is deprecated as of MySQL 5.6.6. CHECK TABLE recommends REPAIR TABLE for tables containing this data type. REPAIR TABLE converts YEAR(2) to YEAR(4).
- As of MySQL 5.7.2, trigger creation time is maintained. If run against a table that has triggers, CHECK TABLE ... FOR UPGRADE displays this warning for each trigger created before MySQL 5.7.2:

  ```
  Trigger db_name.tbl_name.trigger_name does not have CREATED attribute.
  ```

  The warning is informational only. No change is made to the trigger.

Checking Data Consistency

The other check options that can be given are shown in the following table. These options are passed to the storage engine, which may use them or not.

Type	Meaning
QUICK	Do not scan the rows to check for incorrect links. Applies to InnoDB and MyISAM tables and views.
FAST	Check only tables that have not been closed properly. Applies only to MyISAM tables and views; ignored for InnoDB.
CHANGED	Check only tables that have been changed since the last check or that have not been closed properly. Applies only to MyISAM tables and views; ignored for InnoDB.
MEDIUM	Scan rows to verify that deleted links are valid. This also calculates a key checksum for the rows and verifies this with a calculated checksum for the keys. Applies only to MyISAM tables and views; ignored for InnoDB.
EXTENDED	Do a full key lookup for all keys for each row. This ensures that the table is 100% consistent, but takes a long time. Applies only to MyISAM tables and views; ignored for InnoDB.

If none of the options QUICK, MEDIUM, or EXTENDED are specified, the default check type for dynamic-format MyISAM tables is MEDIUM. This has the same result as running myisamchk --medium-check *tbl_name* on the table. The default check type also is MEDIUM for static-format MyISAM tables, unless CHANGED or FAST is specified. In that case, the default is QUICK. The row scan is skipped for CHANGED and FAST because the rows are very seldom corrupted.

You can combine check options, as in the following example that does a quick check on the table to determine whether it was closed properly:

```
CHECK TABLE test_table FAST QUICK;
```

Note

In some cases, CHECK TABLE changes the table. This happens if the table is marked as “corrupted” or “not closed properly” but CHECK TABLE does not find any problems in the table. In this case, CHECK TABLE marks the table as okay.

If a table is corrupted, it is most likely that the problem is in the indexes and not in the data part. All of the preceding check types check the indexes thoroughly and should thus find most errors.

If you just want to check a table that you assume is okay, you should use no check options or the QUICK option. The latter should be used when you are in a hurry and can take the very small risk that QUICK does not find an error in the data file. (In most cases, under normal usage, MySQL should find any error in the data file. If this happens, the table is marked as “corrupted” and cannot be used until it is repaired.)

FAST and CHANGED are mostly intended to be used from a script (for example, to be executed from cron) if you want to check tables from time to time. In most cases, FAST is to be preferred over CHANGED. (The only case when it is not preferred is when you suspect that you have found a bug in the MyISAM code.)

EXTENDED is to be used only after you have run a normal check but still get strange errors from a table when MySQL tries to update a row or find a row by key. This is very unlikely if a normal check has succeeded.

Use of CHECK TABLE ... EXTENDED might influence the execution plan generated by the query optimizer.

Some problems reported by CHECK TABLE cannot be corrected automatically:

- Found row where the auto_increment column has the value 0.

  This means that you have a row in the table where the AUTO_INCREMENT index column contains the value 0. (It is possible to create a row where the AUTO_INCREMENT column is 0 by explicitly setting the column to 0 with an UPDATE statement.)

  This is not an error in itself, but could cause trouble if you decide to dump the table and restore it or do an ALTER TABLE on the table. In this case, the AUTO_INCREMENT column changes value according to the rules of AUTO_INCREMENT columns, which could cause problems such as a duplicate-key error.

  To get rid of the warning, simply execute an UPDATE statement to set the column to some value other than 0.

InnoDB Tables

The following notes apply to InnoDB tables:

- If CHECK TABLE finds a problem for an InnoDB table, the server shuts down to prevent error propagation. Details of the error will be written to the error log.
- If CHECK TABLE encounters corruptions or errors in InnoDB tables or indexes, it reports an error. It does not shut down the server. Starting with MySQL 5.5, CHECK TABLE usually marks the index and sometimes marks the table as corrupted, preventing further use of the index or table.
- If CHECK TABLE finds the wrong number of entries in a secondary index, it will report an error but will not shut down the server or prevent access to the file.
- CHECK TABLE surveys the index page structure, then surveys each key entry. It does not validate the key pointer to a clustered record or follow the path for BLOB pointers.
- When an InnoDB table is stored in its own .ibd file in file-per-table mode, the first 3 pages of the .ibd contain header information rather than table or index data. The CHECK TABLE statement does not detect inconsistencies that only affect the header data. To verify the entire contents of an InnoDB .ibd file, use the innochecksum command.
- When running CHECK TABLE on large InnoDB tables, other threads may be blocked during CHECK TABLE execution. To avoid timeouts, the semaphore wait threshold (600 seconds) is extended by 2 hours (7200 seconds) for CHECK TABLE operations. If InnoDB detects semaphore waits of 240 seconds or more it starts printing InnoDB monitor output to the error log. If a lock request extends beyond the semaphore wait threshold, InnoDB will abort the process. To avoid the possibility of a semaphore wait timeout entirely, you can run CHECK TABLE QUICK instead of CHECK TABLE.

13.7.2.3 CHECKSUM TABLE Syntax

```
CHECKSUM TABLE tbl_name [, tbl_name] ... [ QUICK | EXTENDED ]
```

CHECKSUM TABLE reports a checksum for the contents of a table. You can use this statement to verify that the contents are the same before and after a backup, rollback, or other operation that is intended to put the data back to a known state. This statement requires the SELECT privilege for the table.

Performance Considerations

By default, the entire table is read row by row and the checksum is calculated. For large tables, this could take a long time, thus you would only perform this operation occasionally. This row-by-row calculation is what you get with the EXTENDED clause, with InnoDB and all other storage engines other than MyISAM, and with MyISAM tables not created with the CHECKSUM=1 clause.

For MyISAM tables created with the CHECKSUM=1 clause, CHECKSUM TABLE or CHECKSUM TABLE ... QUICK returns the “live” table checksum that can be returned very fast. If the table does not meet all these conditions, the QUICK method returns NULL. See Section 13.1.14, “CREATE TABLE Syntax” for the syntax of the CHECKSUM clause.

For a nonexistent table, CHECKSUM TABLE returns NULL and generates a warning.

The checksum value depends on the table row format. If the row format changes, the checksum also changes. For example, the storage format for VARCHAR changed between MySQL 4.1 and 5.0, so if a 4.1 table is upgraded to MySQL 5.0, the checksum value may change.

Important

If the checksums for two tables are different, then it is almost certain that the tables are different in some way. However, because the hashing function used by CHECKSUM TABLE is not guaranteed to be collision-free, there is a slight chance that two tables which are not identical can produce the same checksum.

13.7.2.4 OPTIMIZE TABLE Syntax

```
OPTIMIZE [NO_WRITE_TO_BINLOG | LOCAL] TABLE
    tbl_name [, tbl_name] ...
```

Reorganizes the physical storage of table data and associated index data, to reduce storage space and improve I/O efficiency when accessing the table. The exact changes made to each table depend on the storage engine used by that table.

Use OPTIMIZE TABLE in these cases, depending on the type of table:

- After doing substantial insert, update, or delete operations on an InnoDB table that has its own .ibd file because it was created with the innodb_file_per_table option enabled. The table and indexes are reorganized, and disk space can be reclaimed for use by the operating system.

- After doing substantial insert, update, or delete operations on columns that are part of a FULLTEXT index in an InnoDB table. Set the configuration option innodb_optimize_fulltext_only=1 first. To keep the index maintenance period to a reasonable time, set the innodb_ft_num_word_optimize option to specify how many words to update in the search index, and run a sequence of OPTIMIZE TABLE statements until the search index is fully updated.

- After deleting a large part of a MyISAM or ARCHIVE table, or making many changes to a MyISAM or ARCHIVE table with variable-length rows (tables that have VARCHAR, VARBINARY, BLOB, or TEXT columns). Deleted rows are maintained in a linked list and subsequent INSERT operations reuse old row positions. You can use OPTIMIZE TABLE to reclaim the unused space and to defragment the data file. After extensive changes to a table, this statement may also improve performance of statements that use the table, sometimes significantly.

This statement requires SELECT and INSERT privileges for the table.

OPTIMIZE TABLE is also supported for partitioned tables. For information about using this statement with partitioned tables and table partitions, see Section 17.3.4, "Maintenance of Partitions".

In MySQL 5.7.1, gtid_next must be set to AUTOMATIC before issuing this statement. This restriction does not apply in MySQL 5.7.2 or later. (Bug #16062608, Bug #16715809, Bug #69045)

OPTIMIZE TABLE works for InnoDB, MyISAM, and ARCHIVE tables.

By default, OPTIMIZE TABLE does *not* work for tables created using any other storage engine and returns a result indicating this lack of support. You can make OPTIMIZE TABLE work for other storage engines by starting mysqld with the --skip-new option. In this case, OPTIMIZE TABLE is just mapped to ALTER TABLE.

InnoDB Details

For InnoDB tables, OPTIMIZE TABLE is mapped to ALTER TABLE ... FORCE, which rebuilds the table to update index statistics and free unused space in the clustered index. This is displayed in the output of OPTIMIZE TABLE when you run it on an InnoDB table, as shown here:

```
mysql> OPTIMIZE TABLE foo;
+----------+----------+----------+-------------------------------------------------------------------+
| Table    | Op       | Msg_type | Msg_text                                                          |
+----------+----------+----------+-------------------------------------------------------------------+
| test.foo | optimize | note     | Table does not support optimize, doing recreate + analyze instead |
| test.foo | optimize | status   | OK                                                                |
+----------+----------+----------+-------------------------------------------------------------------+
```

Prior to Mysql 5.7.4, OPTIMIZE TABLE does not use online DDL (ALGORITHM=INPLACE). Consequently, concurrent DML (INSERT, UPDATE, DELETE) is not permitted on a table while OPTIMIZE TABLE is running, i.e. the table is locked. Also, secondary indexes are not created as efficiently because keys are inserted in the order they appeared in the primary key.

As of 5.7.4, OPTIMIZE TABLE uses online DDL (ALGORITHM=INPLACE) for both regular and partitioned InnoDB tables. The table rebuild, triggered by OPTIMIZE TABLE and performed under the cover by ALTER TABLE ... FORCE, is now performed using online DDL (ALGORITHM=INPLACE) and only locks the table for a brief interval, which reduces downtime for concurrent DML operations.

OPTIMIZE TABLE continues to use ALGORITHM=COPY under the following conditions:

- When the old_alter_table system variable is turned ON.

- When the mysqld --skip-new option is enabled.

OPTIMIZE TABLE using online DDL (ALGORITHM=INPLACE) is not supported for InnoDB tables that contain FULLTEXT indexes. ALGORITHM=COPY must be used instead.

InnoDB stores data using a page-allocation method and does not suffer from fragmentation in the same way that legacy storage engines (such as MyISAM) will. When considering whether or not to run optimize, consider the workload of transactions that your server will process:

- Some level of fragmentation is expected. InnoDB only fills pages 93% full, to leave room for updates without having to split pages.

- Delete operations might leave gaps that leave pages less filled than desired, which could make it worthwhile to optimize the table.

- Updates to rows usually rewrite the data within the same page, depending on the data type and row format, when sufficient space is available. See Section 14.2.7.5, “How Compression Works for InnoDB Tables” and Section 14.2.9.1, “Overview of InnoDB Row Storage”.

- High-concurrency workloads might leave gaps in indexes over time, as InnoDB retains multiple versions of the same data due through its MVCC mechanism. See Section 14.2.2.12, “InnoDB Multi-Versioning”.

MyISAM Details

For MyISAM tables, OPTIMIZE TABLE works as follows:

1. If the table has deleted or split rows, repair the table.

2. If the index pages are not sorted, sort them.

3. If the table's statistics are not up to date (and the repair could not be accomplished by sorting the index), update them.

Other Considerations

OPTIMIZE TABLE returns a result set with the following columns.

Column	Value
Table	The table name
Op	Always optimize
Msg_type	status, error, info, note, or warning
Msg_text	An informational message

For InnoDB tables prior to 5.7.4 and other table types, MySQL locks the table during the time OPTIMIZE TABLE is running. As of MySQL 5.7.4, OPTIMIZE TABLE is performed online for regular and partitioned InnoDB tables.

By default, the server writes OPTIMIZE TABLE statements to the binary log so that they replicate to replication slaves. To suppress logging, specify the optional NO_WRITE_TO_BINLOG keyword or its alias LOCAL.

OPTIMIZE TABLE does not sort R-tree indexes, such as spatial indexes on POINT columns. (Bug #23578)

OPTIMIZE TABLE table catches and throws any errors that occur while copying table statistics from the old file to the newly created file. For example. if the user ID of the owner of the .frm, .MYD, or .MYI file is different from the user ID of the mysqld process, OPTIMIZE TABLE generates a "cannot change ownership of the file" error unless mysqld is started by the root user.

13.7.2.5 REPAIR TABLE Syntax

```
REPAIR [NO_WRITE_TO_BINLOG | LOCAL] TABLE
    tbl_name [, tbl_name] ...
    [QUICK] [EXTENDED] [USE_FRM]
```

REPAIR TABLE repairs a possibly corrupted table, for certain storage engines only. By default, it has the same effect as myisamchk --recover *tbl_name*.

Note

REPAIR TABLE only applies to MyISAM, ARCHIVE, and CSV tables. See Section 14.3, “The MyISAM Storage Engine”, and Section 14.6, “The ARCHIVE Storage Engine”, and Section 14.5, “The CSV Storage Engine”

This statement requires SELECT and INSERT privileges for the table.

REPAIR TABLE is supported for partitioned tables. However, the USE_FRM option cannot be used with this statement on a partitioned table.

In MySQL 5.7.1, gtid_next must be set to AUTOMATIC before issuing this statement. This restriction does not apply in MySQL 5.7.2 or later. (Bug #16062608, Bug #16715809, Bug #69045)

You can use ALTER TABLE ... REPAIR PARTITION to repair one or more partitions; for more information, see Section 13.1.6, “ALTER TABLE Syntax”, and Section 17.3.4, “Maintenance of Partitions”.

Although normally you should never have to run REPAIR TABLE, if disaster strikes, this statement is very likely to get back all your data from a MyISAM table. If your tables become corrupted often, try to find the reason for it, to eliminate the need to use REPAIR TABLE. See Section C.5.4.2, “What to Do If MySQL Keeps Crashing”, and Section 14.3.4, “MyISAM Table Problems”.

Caution

Make a backup of a table before performing a table repair operation; under some circumstances the operation might cause data loss. Possible causes include but are not limited to file system errors. See Chapter 7, *Backup and Recovery*.

Warning

If the server crashes during a REPAIR TABLE operation, it is essential after restarting it that you immediately execute another REPAIR TABLE statement for the table before performing any other operations on it. In the worst case, you might have a new clean index file without information about the data file, and then the next operation you perform could overwrite the data file. This is an unlikely but possible scenario that underscores the value of making a backup first.

REPAIR TABLE returns a result set with the following columns.

Column	Value
Table	The table name
Op	Always repair
Msg_type	status, error, info, note, or warning
Msg_text	An informational message

The REPAIR TABLE statement might produce many rows of information for each repaired table. The last row has a Msg_type value of status and Msg_test normally should be OK. If you do not get OK for a MyISAM table, you should try repairing it with myisamchk --safe-recover. (REPAIR TABLE does not implement all the options of myisamchk.) With myisamchk --safe-recover, you can also use options that REPAIR TABLE does not support, such as --max-record-length.

If you use the QUICK option, REPAIR TABLE tries to repair only the index file, and not the data file. This type of repair is like that done by myisamchk --recover --quick.

If you use the EXTENDED option, MySQL creates the index row by row instead of creating one index at a time with sorting. This type of repair is like that done by myisamchk --safe-recover.

The USE_FRM option is available for use if the .MYI index file is missing or if its header is corrupted. This option tells MySQL not to trust the information in the .MYI file header and to re-create it using information from the .frm file. This kind of repair cannot be done with myisamchk.

Note

Use the USE_FRM option *only* if you cannot use regular REPAIR modes! Telling the server to ignore the .MYI file makes important table metadata stored in the .MYI unavailable to the repair process, which can have deleterious consequences:

- The current AUTO_INCREMENT value is lost.
- The link to deleted records in the table is lost, which means that free space for deleted records will remain unoccupied thereafter.
- The .MYI header indicates whether the table is compressed. If the server ignores this information, it cannot tell that a table is compressed and repair can cause change or loss of table contents. This means that USE_FRM should not be used with compressed tables. That should not be necessary, anyway: Compressed tables are read only, so they should not become corrupt.

Caution

If you use USE_FRM for a table that was created by a different version of the MySQL server than the one you are currently running, REPAIR TABLE will not attempt to repair the table. In this case, the result set returned by REPAIR TABLE contains a line with a Msg_type value of error and a Msg_text value of Failed repairing incompatible .FRM file.

If USE_FRM is *not* used, REPAIR TABLE checks the table to see whether an upgrade is required. If so, it performs the upgrade, following the same rules as CHECK TABLE ... FOR UPGRADE. See Section 13.7.2.2, “CHECK TABLE Syntax”, for more information. REPAIR TABLE without USE_FRM upgrades the .frm file to the current version.

By default, the server writes REPAIR TABLE statements to the binary log so that they replicate to replication slaves. To suppress logging, specify the optional NO_WRITE_TO_BINLOG keyword or its alias LOCAL.

Important

In the event that a table on the master becomes corrupted and you run REPAIR TABLE on it, any resulting changes to the original table are *not* propagated to slaves.

You may be able to increase REPAIR TABLE performance by setting certain system variables. See Section 8.6.3, “Speed of REPAIR TABLE Statements”.

REPAIR TABLE table catches and throws any errors that occur while copying table statistics from the old corrupted file to the newly created file. For example. if the user ID of the owner of the .frm, .MYD, or .MYI file is different from the user ID of the mysqld process, REPAIR TABLE generates a "cannot change ownership of the file" error unless mysqld is started by the root user.

13.7.3 Plugin and User-Defined Function Statements

13.7.3.1 CREATE FUNCTION Syntax for User-Defined Functions

```
CREATE [AGGREGATE] FUNCTION function_name RETURNS {STRING|INTEGER|REAL|DECIMAL}
    SONAME shared_library_name
```

A user-defined function (UDF) is a way to extend MySQL with a new function that works like a native (built-in) MySQL function such as `ABS()` or `CONCAT()`.

`function_name` is the name that should be used in SQL statements to invoke the function. The `RETURNS` clause indicates the type of the function's return value. `DECIMAL` is a legal value after `RETURNS`, but currently `DECIMAL` functions return string values and should be written like `STRING` functions.

`shared_library_name` is the basename of the shared object file that contains the code that implements the function. The file must be located in the plugin directory. This directory is given by the value of the `plugin_dir` system variable. For more information, see Section 22.3.2.5, “Compiling and Installing User-Defined Functions”.

To create a function, you must have the `INSERT` privilege for the `mysql` database. This is necessary because `CREATE FUNCTION` adds a row to the `mysql.func` system table that records the function's name, type, and shared library name. If you do not have this table, you should run the `mysql_upgrade` command to create it. See Section 4.4.7, “`mysql_upgrade` — Check and Upgrade MySQL Tables”.

An active function is one that has been loaded with `CREATE FUNCTION` and not removed with `DROP FUNCTION`. All active functions are reloaded each time the server starts, unless you start `mysqld` with the `--skip-grant-tables` option. In this case, UDF initialization is skipped and UDFs are unavailable.

For instructions on writing user-defined functions, see Section 22.3.2, “Adding a New User-Defined Function”. For the UDF mechanism to work, functions must be written in C or C++ (or another language that can use C calling conventions), your operating system must support dynamic loading and you must have compiled `mysqld` dynamically (not statically).

An `AGGREGATE` function works exactly like a native MySQL aggregate (summary) function such as `SUM` or `COUNT()`. For `AGGREGATE` to work, your `mysql.func` table must contain a `type` column. If your `mysql.func` table does not have this column, you should run the `mysql_upgrade` program to create it (see Section 4.4.7, “`mysql_upgrade` — Check and Upgrade MySQL Tables”).

Note

To upgrade the shared library associated with a UDF, issue a `DROP FUNCTION` statement, upgrade the shared library, and then issue a `CREATE FUNCTION` statement. If you upgrade the shared library first and then use `DROP FUNCTION`, the server may crash.

13.7.3.2 `DROP FUNCTION` Syntax

```
DROP FUNCTION function_name
```

This statement drops the user-defined function (UDF) named *`function_name`*.

To drop a function, you must have the `DELETE` privilege for the `mysql` database. This is because `DROP FUNCTION` removes a row from the `mysql.func` system table that records the function's name, type, and shared library name.

Note

To upgrade the shared library associated with a UDF, issue a `DROP FUNCTION` statement, upgrade the shared library, and then issue a `CREATE FUNCTION` statement. If you upgrade the shared library first and then use `DROP FUNCTION`, the server may crash.

DROP FUNCTION is also used to drop stored functions (see Section 13.1.21, “DROP PROCEDURE and DROP FUNCTION Syntax”).

13.7.3.3 INSTALL PLUGIN Syntax

```
INSTALL PLUGIN plugin_name SONAME 'shared_library_name'
```

This statement installs a server plugin. It requires the INSERT privilege for the mysql.plugin table.

plugin_name is the name of the plugin as defined in the plugin descriptor structure contained in the library file (see Section 22.2.4.2, “Plugin Data Structures”). Plugin names are not case sensitive. For maximal compatibility, plugin names should be limited to ASCII letters, digits, and underscore because they are used in C source files, shell command lines, M4 and Bourne shell scripts, and SQL environments.

shared_library_name is the name of the shared library that contains the plugin code. The name includes the file name extension (for example, libmyplugin.so, libmyplugin.dll, or libmyplugin.dylib).

The shared library must be located in the plugin directory (the directory named by the plugin_dir system variable). The library must be in the plugin directory itself, not in a subdirectory. By default, plugin_dir is the plugin directory under the directory named by the pkglibdir configuration variable, but it can be changed by setting the value of plugin_dir at server startup. For example, set its value in a my.cnf file:

```
[mysqld]
plugin_dir=/path/to/plugin/directory
```

If the value of plugin_dir is a relative path name, it is taken to be relative to the MySQL base directory (the value of the basedir system variable).

INSTALL PLUGIN loads and initializes the plugin code to make the plugin available for use. A plugin is initialized by executing its initialization function, which handles any setup that the plugin must perform before it can be used. When the server shuts down, it executes the deinitialization function for each plugin that is loaded so that the plugin has a change to perform any final cleanup.

INSTALL PLUGIN also registers the plugin by adding a line that indicates the plugin name and library file name to the mysql.plugin table. At server startup, the server loads and initializes any plugin that is listed in the mysql.plugin table. This means that a plugin is installed with INSTALL PLUGIN only once, not every time the server starts. Plugin loading at startup does not occur if the server is started with the --skip-grant-tables option.

A plugin library can contain multiple plugins. For each of them to be installed, use a separate INSTALL PLUGIN statement. Each statement names a different plugin, but all of them specify the same library name.

INSTALL PLUGIN causes the server to read option (my.cnf) files just as during server startup. This enables the plugin to pick up any relevant options from those files. It is possible to add plugin options to an option file even before loading a plugin (if the loose prefix is used). It is also possible to uninstall a plugin, edit my.cnf, and install the plugin again. Restarting the plugin this way enables it to the new option values without a server restart.

For options that control individual plugin loading at server startup, see Section 5.1.8.1, “Installing and Uninstalling Plugins”. If you need to load plugins for a single server startup when the --skip-grant-tables option is given (which tells the server not to read system tables), use the --plugin-load option. See Section 5.1.3, “Server Command Options”.

To remove a plugin, use the UNINSTALL PLUGIN statement.

For additional information about plugin loading, see Section 5.1.8.1, “Installing and Uninstalling Plugins”.

To see what plugins are installed, use the SHOW PLUGINS statement or query the INFORMATION_SCHEMA.PLUGINS table.

If you recompile a plugin library and need to reinstall it, you can use either of the following methods:

- Use UNINSTALL PLUGIN to uninstall all plugins in the library, install the new plugin library file in the plugin directory, and then use INSTALL PLUGIN to install all plugins in the library. This procedure has the advantage that it can be used without stopping the server. However, if the plugin library contains many plugins, you must issue many INSTALL PLUGIN and UNINSTALL PLUGIN statements.
- Stop the server, install the new plugin library file in the plugin directory, and restart the server.

13.7.3.4 UNINSTALL PLUGIN Syntax

```
UNINSTALL PLUGIN plugin_name
```

This statement removes an installed server plugin. It requires the DELETE privilege for the mysql.plugin table.

plugin_name must be the name of some plugin that is listed in the mysql.plugin table. The server executes the plugin's deinitialization function and removes the row for the plugin from the mysql.plugin table, so that subsequent server restarts will not load and initialize the plugin. UNINSTALL PLUGIN does not remove the plugin's shared library file.

You cannot uninstall a plugin if any table that uses it is open.

Plugin removal has implications for the use of associated tables. For example, if a full-text parser plugin is associated with a FULLTEXT index on the table, uninstalling the plugin makes the table unusable. Any attempt to access the table results in an error. The table cannot even be opened, so you cannot drop an index for which the plugin is used. This means that uninstalling a plugin is something to do with care unless you do not care about the table contents. If you are uninstalling a plugin with no intention of reinstalling it later and you care about the table contents, you should dump the table with mysqldump and remove the WITH PARSER clause from the dumped CREATE TABLE statement so that you can reload the table later. If you do not care about the table, DROP TABLE can be used even if any plugins associated with the table are missing.

For additional information about plugin loading, see Section 5.1.8.1, “Installing and Uninstalling Plugins”.

13.7.4 SET Syntax

```
SET variable_assignment [, variable_assignment] ...

variable_assignment:
      user_var_name = expr
    | [GLOBAL | SESSION] system_var_name = expr
    | [@@global. | @@session. | @@]system_var_name = expr
```

The SET statement assigns values to different types of variables that affect the operation of the server or your client.

This section describes use of SET for assigning values to variables. The SET statement can be used to assign values to these types of variables:

- System variables. See Section 5.1.4, “Server System Variables”. System variables also can be set at server startup, as described in Section 5.1.5, “Using System Variables”.

User-defined variables. See Section 9.4, “User-Defined Variables”.

- Stored procedure and function parameters, and stored program local variables. See Section 13.6.4, “Variables in Stored Programs”.

Some variants of SET syntax are used in other contexts:

- SET CHARACTER SET and SET NAMES assign values to character set and collation variables associated with the connection to the server. SET ONE_SHOT is used for replication. These variants are described later in this section.
- SET PASSWORD assigns account passwords. See Section 13.7.1.7, “SET PASSWORD Syntax”.
- SET TRANSACTION ISOLATION LEVEL sets the isolation level for transaction processing. See Section 13.3.6, “SET TRANSACTION Syntax”.

The following discussion shows the different SET syntaxes that you can use to set variables. The examples use the = assignment operator, but you can also use the := assignment operator for this purpose.

A user variable is written as *@var_name* and can be set as follows:

```
SET @var_name = expr;
```

Many system variables are dynamic and can be changed while the server runs by using the SET statement. For a list, see Section 5.1.5.2, “Dynamic System Variables”. To change a system variable with SET, refer to it as *var_name*, optionally preceded by a modifier:

- To indicate explicitly that a variable is a global variable, precede its name by GLOBAL or @@global.. The SUPER privilege is required to set global variables.
- To indicate explicitly that a variable is a session variable, precede its name by SESSION, @@session., or @@. Setting a session variable requires no special privilege, but a client can change only its own session variables, not those of any other client.
- LOCAL and @@local. are synonyms for SESSION and @@session..
- If no modifier is present, SET changes the session variable.

A SET statement can contain multiple variable assignments, separated by commas. For example, the statement can assign values to a user-defined variable and a system variable. If you set several system variables, the most recent GLOBAL or SESSION modifier in the statement is used for following variables that have no modifier specified.

Examples:

```
SET sort_buffer_size=10000;
SET @@local.sort_buffer_size=10000;
SET GLOBAL sort_buffer_size=1000000, SESSION sort_buffer_size=1000000;
SET @@sort_buffer_size=1000000;
SET @@global.sort_buffer_size=1000000, @@local.sort_buffer_size=1000000;
```

The *@@var_name* syntax for system variables is supported for compatibility with some other database systems.

If you change a session system variable, the value remains in effect until your session ends or until you change the variable to a different value. The change is not visible to other clients.

If you change a global system variable, the value is remembered and used for new connections until the server restarts. (To make a global system variable setting permanent, you should set it in an option file.) The change is visible to any client that accesses that global variable. However, the change affects the corresponding session variable only for clients that connect after the change. The global variable change does not affect the session variable for any client that is currently connected (not even that of the client that issues the `SET GLOBAL` statement).

To prevent incorrect usage, MySQL produces an error if you use `SET GLOBAL` with a variable that can only be used with `SET SESSION` or if you do not specify `GLOBAL` (or `@@global.`) when setting a global variable.

To set a `SESSION` variable to the `GLOBAL` value or a `GLOBAL` value to the compiled-in MySQL default value, use the `DEFAULT` keyword. For example, the following two statements are identical in setting the session value of `max_join_size` to the global value:

```
SET max_join_size=DEFAULT;
SET @@session.max_join_size=@@global.max_join_size;
```

Not all system variables can be set to `DEFAULT`. In such cases, use of `DEFAULT` results in an error.

It is not permitted to assign the value `DEFAULT` to user-defined variables, stored procedure or function parameters, or stored program local variables. This results in a syntax error for user-defined variables, parameters, and local variables.

You can refer to the values of specific global or session system variables in expressions by using one of the `@@`-modifiers. For example, you can retrieve values in a `SELECT` statement like this:

```
SELECT @@global.sql_mode, @@session.sql_mode, @@sql_mode;
```

When you refer to a system variable in an expression as `@@var_name` (that is, when you do not specify `@@global.` or `@@session.`), MySQL returns the session value if it exists and the global value otherwise. (This differs from `SET @@var_name = value`, which always refers to the session value.)

Note

Some variables displayed by `SHOW VARIABLES` may not be available using `SELECT @@var_name` syntax; an `Unknown system variable` occurs. As a workaround in such cases, you can use `SHOW VARIABLES LIKE 'var_name'`.

Suffixes for specifying a value multiplier can be used when setting a variable at server startup, but not to set the value with `SET` at runtime. On the other hand, with `SET` you can assign a variable's value using an expression, which is not true when you set a variable at server startup. For example, the first of the following lines is legal at server startup, but the second is not:

```
shell> mysql --max_allowed_packet=16M
shell> mysql --max_allowed_packet=16*1024*1024
```

Conversely, the second of the following lines is legal at runtime, but the first is not:

```
mysql> SET GLOBAL max_allowed_packet=16M;
mysql> SET GLOBAL max_allowed_packet=16*1024*1024;
```

To display system variables names and values, use the `SHOW VARIABLES` statement. (See Section 13.7.5.38, “SHOW VARIABLES Syntax”.)

The following list describes SET options that have nonstandard syntax (that is, options that are not set with *name = value* syntax).

- CHARACTER SET {*charset_name* | DEFAULT}

  This maps all strings from and to the client with the given mapping. You can add new mappings by editing sql/convert.cc in the MySQL source distribution. SET CHARACTER SET sets three session system variables: character_set_client and character_set_results are set to the given character set, and character_set_connection to the value of character_set_database. See Section 10.1.4, “Connection Character Sets and Collations”.

  The default mapping can be restored by using the value DEFAULT. The default depends on the server configuration.

  ucs2, utf16, and utf32 cannot be used as a client character set, which means that they do not work for SET CHARACTER SET.

- NAMES {'*charset_name*' [COLLATE '*collation_name*'] | DEFAULT}

  SET NAMES sets the three session system variables character_set_client, character_set_connection, and character_set_results to the given character set. Setting character_set_connection to charset_name also sets collation_connection to the default collation for charset_name. The optional COLLATE clause may be used to specify a collation explicitly. See Section 10.1.4, “Connection Character Sets and Collations”.

  The default mapping can be restored by using a value of DEFAULT. The default depends on the server configuration.

  ucs2, utf16, and utf32 cannot be used as a client character set, which means that they do not work for SET NAMES.

13.7.5 SHOW Syntax

SHOW has many forms that provide information about databases, tables, columns, or status information about the server. This section describes those following:

```
SHOW {BINARY | MASTER} LOGS
SHOW BINLOG EVENTS [IN 'log_name'] [FROM pos] [LIMIT [offset,] row_count]
SHOW CHARACTER SET [like_or_where]
SHOW COLLATION [like_or_where]
SHOW [FULL] COLUMNS FROM tbl_name [FROM db_name] [like_or_where]
SHOW CREATE DATABASE db_name
SHOW CREATE EVENT event_name
SHOW CREATE FUNCTION func_name
SHOW CREATE PROCEDURE proc_name
SHOW CREATE TABLE tbl_name
SHOW CREATE TRIGGER trigger_name
SHOW CREATE VIEW view_name
SHOW DATABASES [like_or_where]
SHOW ENGINE engine_name {STATUS | MUTEX}
SHOW [STORAGE] ENGINES
SHOW ERRORS [LIMIT [offset,] row_count]
SHOW EVENTS
SHOW FUNCTION CODE func_name
SHOW FUNCTION STATUS [like_or_where]
SHOW GRANTS FOR user
SHOW INDEX FROM tbl_name [FROM db_name]
SHOW MASTER STATUS
SHOW OPEN TABLES [FROM db_name] [like_or_where]
SHOW PLUGINS
SHOW PROCEDURE CODE proc_name
SHOW PROCEDURE STATUS [like_or_where]
SHOW PRIVILEGES
SHOW [FULL] PROCESSLIST
SHOW PROFILE [types] [FOR QUERY n] [OFFSET n] [LIMIT n]
SHOW PROFILES
SHOW SLAVE HOSTS
SHOW SLAVE STATUS [NONBLOCKING]
SHOW [GLOBAL | SESSION] STATUS [like_or_where]
SHOW TABLE STATUS [FROM db_name] [like_or_where]
SHOW [FULL] TABLES [FROM db_name] [like_or_where]
SHOW TRIGGERS [FROM db_name] [like_or_where]
SHOW [GLOBAL | SESSION] VARIABLES [like_or_where]
SHOW WARNINGS [LIMIT [offset,] row_count]

like_or_where:
    LIKE 'pattern'
  | WHERE expr
```

If the syntax for a given SHOW statement includes a LIKE 'pattern' part, 'pattern' is a string that can contain the SQL "%" and "_" wildcard characters. The pattern is useful for restricting statement output to matching values.

Several SHOW statements also accept a WHERE clause that provides more flexibility in specifying which rows to display. See Section 19.31, "Extensions to SHOW Statements".

Many MySQL APIs (such as PHP) enable you to treat the result returned from a SHOW statement as you would a result set from a SELECT; see Chapter 21, *Connectors and APIs*, or your API documentation for more information. In addition, you can work in SQL with results from queries on tables in the INFORMATION_SCHEMA database, which you cannot easily do with results from SHOW statements. See Chapter 19, *INFORMATION_SCHEMA Tables*.

13.7.5.1 SHOW BINARY LOGS Syntax

```
SHOW BINARY LOGS
SHOW MASTER LOGS
```

Lists the binary log files on the server. This statement is used as part of the procedure described in Section 13.4.1.1, "PURGE BINARY LOGS Syntax", that shows how to determine which logs can be purged.

```
mysql> SHOW BINARY LOGS;
+---------------+-----------+
| Log_name      | File_size |
+---------------+-----------+
| binlog.000015 |    724935 |
| binlog.000016 |    733481 |
+---------------+-----------+
```

SHOW MASTER LOGS is equivalent to SHOW BINARY LOGS.

A user with the SUPER or REPLICATION CLIENT privilege may execute this statement.

13.7.5.2 SHOW BINLOG EVENTS Syntax

```
SHOW BINLOG EVENTS
   [IN 'log_name'] [FROM pos] [LIMIT [offset,] row_count]
```

Shows the events in the binary log. If you do not specify 'log_name', the first binary log is displayed.

The LIMIT clause has the same syntax as for the SELECT statement. See Section 13.2.9, "SELECT Syntax".

Note

Issuing a SHOW BINLOG EVENTS with no LIMIT clause could start a very time- and resource-consuming process because the server returns to the client the complete contents of the binary log (which includes all statements executed by the server that modify data). As an alternative to SHOW BINLOG EVENTS, use the mysqlbinlog utility to save the binary log to a text file for later examination and analysis. See Section 4.6.7, "mysqlbinlog — Utility for Processing Binary Log Files".

Note

Some events relating to the setting of user and system variables are not included in the output from SHOW BINLOG EVENTS. To get complete coverage of events within a binary log, use mysqlbinlog.

Note

SHOW BINLOG EVENTS does *not* work with relay log files. You can use SHOW RELAYLOG EVENTS for this purpose.

13.7.5.3 SHOW CHARACTER SET Syntax

```
SHOW CHARACTER SET
    [LIKE 'pattern' | WHERE expr]
```

The SHOW CHARACTER SET statement shows all available character sets. The LIKE clause, if present, indicates which character set names to match. The WHERE clause can be given to select rows using more general conditions, as discussed in Section 19.31, "Extensions to SHOW Statements". For example:

```
mysql> SHOW CHARACTER SET LIKE 'latin%';
+---------+-----------------------------+-------------------+--------+
| Charset | Description                 | Default collation | Maxlen |
+---------+-----------------------------+-------------------+--------+
| latin1  | cp1252 West European        | latin1_swedish_ci |      1 |
| latin2  | ISO 8859-2 Central European | latin2_general_ci |      1 |
| latin5  | ISO 8859-9 Turkish          | latin5_turkish_ci |      1 |
| latin7  | ISO 8859-13 Baltic          | latin7_general_ci |      1 |
+---------+-----------------------------+-------------------+--------+
```

The Maxlen column shows the maximum number of bytes required to store one character.

The filename character set is for internal use only; consequently, SHOW CHARACTER SET does not display it.

13.7.5.4 SHOW COLLATION Syntax

```
SHOW COLLATION
    [LIKE 'pattern' | WHERE expr]
```

This statement lists collations supported by the server. By default, the output from SHOW COLLATION includes all available collations. The LIKE clause, if present, indicates which collation names to match. The WHERE clause can be given to select rows using more general conditions, as discussed in Section 19.31, "Extensions to SHOW Statements". For example:

```
mysql> SHOW COLLATION LIKE 'latin1%';
+-------------------+---------+----+---------+----------+---------+
| Collation         | Charset | Id | Default | Compiled | Sortlen |
+-------------------+---------+----+---------+----------+---------+
| latin1_german1_ci | latin1  |  5 |         |          |       0 |
| latin1_swedish_ci | latin1  |  8 | Yes     | Yes      |       0 |
| latin1_danish_ci  | latin1  | 15 |         |          |       0 |
| latin1_german2_ci | latin1  | 31 |         | Yes      |       2 |
| latin1_bin        | latin1  | 47 |         | Yes      |       0 |
| latin1_general_ci | latin1  | 48 |         |          |       0 |
| latin1_general_cs | latin1  | 49 |         |          |       0 |
| latin1_spanish_ci | latin1  | 94 |         |          |       0 |
+-------------------+---------+----+---------+----------+---------+
```

The Collation and Charset columns indicate the names of the collation and the character set with which it is associated. Id is the collation ID. Default indicates whether the collation is the default for its character set. Compiled indicates whether the character set is compiled into the server. Sortlen is related to the amount of memory required to sort strings expressed in the character set.

To see the default collation for each character set, use the following statement. Default is a reserved word, so to use it as an identifier, it must be quoted as such:

```
mysql> SHOW COLLATION WHERE `Default` = 'Yes';
+---------------------+----------+----+---------+----------+---------+
| Collation           | Charset  | Id | Default | Compiled | Sortlen |
+---------------------+----------+----+---------+----------+---------+
| big5_chinese_ci     | big5     |  1 | Yes     | Yes      |       1 |
| dec8_swedish_ci     | dec8     |  3 | Yes     | Yes      |       1 |
| cp850_general_ci    | cp850    |  4 | Yes     | Yes      |       1 |
| hp8_english_ci      | hp8      |  6 | Yes     | Yes      |       1 |
| koi8r_general_ci    | koi8r    |  7 | Yes     | Yes      |       1 |
| latin1_swedish_ci   | latin1   |  8 | Yes     | Yes      |       1 |
...
```

13.7.5.5 SHOW COLUMNS Syntax

```
SHOW [FULL] COLUMNS {FROM | IN} tbl_name [{FROM | IN} db_name]
    [LIKE 'pattern' | WHERE expr]
```

SHOW COLUMNS displays information about the columns in a given table. It also works for views. The LIKE clause, if present, indicates which column names to match. The WHERE clause can be given to select rows using more general conditions, as discussed in Section 19.31, "Extensions to SHOW Statements".

SHOW COLUMNS displays information only for those columns for which you have some privilege.

```
mysql> SHOW COLUMNS FROM City;
+------------+----------+------+-----+---------+----------------+
| Field      | Type     | Null | Key | Default | Extra          |
+------------+----------+------+-----+---------+----------------+
| Id         | int(11)  | NO   | PRI | NULL    | auto_increment |
| Name       | char(35) | NO   |     |         |                |
| Country    | char(3)  | NO   | UNI |         |                |
| District   | char(20) | YES  | MUL |         |                |
| Population | int(11)  | NO   |     | 0       |                |
+------------+----------+------+-----+---------+----------------+
5 rows in set (0.00 sec)
```

If the data types differ from what you expect them to be based on a CREATE TABLE statement, note that MySQL sometimes changes data types when you create or alter a table. The conditions under which this occurs are described in Section 13.1.14.3, "Silent Column Specification Changes".

The `FULL` keyword causes the output to include the column collation and comments, as well as the privileges you have for each column.

You can use *`db_name.tbl_name`* as an alternative to the *`tbl_name`* `FROM` *`db_name`* syntax. In other words, these two statements are equivalent:

```
mysql> SHOW COLUMNS FROM mytable FROM mydb;
mysql> SHOW COLUMNS FROM mydb.mytable;
```

`SHOW COLUMNS` displays the following values for each table column:

`Field` indicates the column name.

`Type` indicates the column data type.

`Collation` indicates the collation for nonbinary string columns, or `NULL` for other columns. This value is displayed only if you use the `FULL` keyword.

The `Null` field contains `YES` if `NULL` values can be stored in the column, `NO` if not.

The `Key` field indicates whether the column is indexed:

- If `Key` is empty, the column either is not indexed or is indexed only as a secondary column in a multiple-column, nonunique index.

- If `Key` is `PRI`, the column is a `PRIMARY KEY` or is one of the columns in a multiple-column `PRIMARY KEY`.

- If `Key` is `UNI`, the column is the first column of a `UNIQUE` index. (A `UNIQUE` index permits multiple `NULL` values, but you can tell whether the column permits `NULL` by checking the `Null` field.)

- If `Key` is `MUL`, the column is the first column of a nonunique index in which multiple occurrences of a given value are permitted within the column.

If more than one of the `Key` values applies to a given column of a table, `Key` displays the one with the highest priority, in the order `PRI`, `UNI`, `MUL`.

A `UNIQUE` index may be displayed as `PRI` if it cannot contain `NULL` values and there is no `PRIMARY KEY` in the table. A `UNIQUE` index may display as `MUL` if several columns form a composite `UNIQUE` index; although the combination of the columns is unique, each column can still hold multiple occurrences of a given value.

The `Default` field indicates the default value that is assigned to the column. This is `NULL` if the column has an explicit default of `NULL`, or if the column definition has no `DEFAULT` clause.

The `Extra` field contains any additional information that is available about a given column. The value is nonempty in these cases: `auto_increment` for columns that have the `AUTO_INCREMENT` attribute; `on update CURRENT_TIMESTAMP` for `TIMESTAMP` or `DATETIME` columns that have the `ON UPDATE CURRENT_TIMESTAMP` attribute.

`Privileges` indicates the privileges you have for the column. This value is displayed only if you use the `FULL` keyword.

`Comment` indicates any comment the column has. This value is displayed only if you use the `FULL` keyword.

`SHOW FIELDS` is a synonym for `SHOW COLUMNS`. You can also list a table's columns with the `mysqlshow` *`db_name tbl_name`* command.

The DESCRIBE statement provides information similar to SHOW COLUMNS. See Section 13.8.1, “DESCRIBE Syntax”.

The SHOW CREATE TABLE, SHOW TABLE STATUS, and SHOW INDEX statements also provide information about tables. See Section 13.7.5, “SHOW Syntax”.

13.7.5.6 SHOW CREATE DATABASE Syntax

```
SHOW CREATE {DATABASE | SCHEMA} [IF NOT EXISTS] db_name
```

Shows the CREATE DATABASE statement that creates the named database. If the SHOW statement includes an IF NOT EXISTS clause, the output too includes such a clause. SHOW CREATE SCHEMA is a synonym for SHOW CREATE DATABASE.

```
mysql> SHOW CREATE DATABASE test\G
*************************** 1. row ***************************
       Database: test
Create Database: CREATE DATABASE `test`
                 /*!40100 DEFAULT CHARACTER SET latin1 */

mysql> SHOW CREATE SCHEMA test\G
*************************** 1. row ***************************
       Database: test
Create Database: CREATE DATABASE `test`
                 /*!40100 DEFAULT CHARACTER SET latin1 */
```

SHOW CREATE DATABASE quotes table and column names according to the value of the sql_quote_show_create option. See Section 5.1.4, “Server System Variables”.

13.7.5.7 SHOW CREATE EVENT Syntax

```
SHOW CREATE EVENT event_name
```

This statement displays the CREATE EVENT statement needed to re-create a given event. It requires the EVENT privilege for the database from which the event is to be shown. For example (using the same event e_daily defined and then altered in Section 13.7.5.17, “SHOW EVENTS Syntax”):

```
mysql> SHOW CREATE EVENT test.e_daily\G
*************************** 1. row ***************************
               Event: e_daily
            sql_mode:
           time_zone: SYSTEM
        Create Event: CREATE EVENT `e_daily`
                        ON SCHEDULE EVERY 1 DAY
                        STARTS CURRENT_TIMESTAMP + INTERVAL 6 HOUR
                        ON COMPLETION NOT PRESERVE
                        ENABLE
                        COMMENT 'Saves total number of sessions then
                                clears the table each day'
                        DO BEGIN
                          INSERT INTO site_activity.totals (time, total)
                            SELECT CURRENT_TIMESTAMP, COUNT(*)
                            FROM site_activity.sessions;
                          DELETE FROM site_activity.sessions;
                        END
character_set_client: latin1
collation_connection: latin1_swedish_ci
  Database Collation: latin1_swedish_ci
```

character_set_client is the session value of the character_set_client system variable when the event was created. collation_connection is the session value of the collation_connection system variable when the event was created. Database Collation is the collation of the database with which the event is associated.

Note that the output reflects the current status of the event (ENABLE) rather than the status with which it was created.

13.7.5.8 SHOW CREATE FUNCTION Syntax

```
SHOW CREATE FUNCTION func_name
```

This statement is similar to SHOW CREATE PROCEDURE but for stored functions. See Section 13.7.5.9, “SHOW CREATE PROCEDURE Syntax”.

13.7.5.9 SHOW CREATE PROCEDURE Syntax

```
SHOW CREATE PROCEDURE proc_name
```

This statement is a MySQL extension. It returns the exact string that can be used to re-create the named stored procedure. A similar statement, SHOW CREATE FUNCTION, displays information about stored functions (see Section 13.7.5.8, “SHOW CREATE FUNCTION Syntax”).

To use either statement, you must be the owner of the routine or have SELECT access to the mysql.proc table. If you do not have privileges for the routine itself, the value displayed for the Create Procedure or Create Function field will be NULL.

```
mysql> SHOW CREATE PROCEDURE test.simpleproc\G
*************************** 1. row ***************************
           Procedure: simpleproc
            sql_mode:
    Create Procedure: CREATE PROCEDURE `simpleproc`(OUT param1 INT)
                      BEGIN
                      SELECT COUNT(*) INTO param1 FROM t;
                      END
character_set_client: latin1
collation_connection: latin1_swedish_ci
  Database Collation: latin1_swedish_ci

mysql> SHOW CREATE FUNCTION test.hello\G
*************************** 1. row ***************************
            Function: hello
            sql_mode:
     Create Function: CREATE FUNCTION `hello`(s CHAR(20))
                      RETURNS CHAR(50)
                      RETURN CONCAT('Hello, ',s,'!')
character_set_client: latin1
collation_connection: latin1_swedish_ci
  Database Collation: latin1_swedish_ci
```

character_set_client is the session value of the character_set_client system variable when the routine was created. collation_connection is the session value of the collation_connection system variable when the routine was created. Database Collation is the collation of the database with which the routine is associated.

13.7.5.10 SHOW CREATE TABLE Syntax

```
SHOW CREATE TABLE tbl_name
```

Shows the CREATE TABLE statement that creates the named table. To use this statement, you must have some privilege for the table. This statement also works with views.

```
mysql> SHOW CREATE TABLE t\G
*************************** 1. row ***************************
       Table: t
Create Table: CREATE TABLE t (
  id INT(11) default NULL auto_increment,
  s char(60) default NULL,
  PRIMARY KEY (id)
) ENGINE=MyISAM
```

SHOW CREATE TABLE quotes table and column names according to the value of the sql_quote_show_create option. See Section 5.1.4, “Server System Variables”.

13.7.5.11 SHOW CREATE TRIGGER Syntax

```
SHOW CREATE TRIGGER trigger_name
```

This statement shows the CREATE TRIGGER statement that creates the named trigger.

```
mysql> SHOW CREATE TRIGGER ins_sum\G
*************************** 1. row ***************************
               Trigger: ins_sum
              sql_mode: STRICT_TRANS_TABLES,NO_ENGINE_SUBSTITUTION
SQL Original Statement: CREATE DEFINER=`me`@`localhost` TRIGGER ins_sum
                        BEFORE INSERT ON account
                        FOR EACH ROW SET @sum = @sum + NEW.amount
  character_set_client: utf8
  collation_connection: utf8_general_ci
    Database Collation: latin1_swedish_ci
               Created: 2013-07-09 10:39:34.96
```

SHOW CREATE TRIGGER output has the following columns:

- Trigger: The trigger name.
- sql_mode: The SQL mode in effect when the trigger executes.
- SQL Original Statement: The CREATE TRIGGER statement that defines the trigger.
- character_set_client: The session value of the character_set_client system variable when the trigger was created.
- collation_connection: The session value of the collation_connection system variable when the trigger was created.
- Database Collation: The collation of the database with which the trigger is associated.
- Created: The date and time when the trigger was created. This is a TIMESTAMP(2) value (with a fractional part in hundredths of seconds) for triggers created in MySQL 5.7.2 or later, NULL for triggers created prior to 5.7.2. This column was added in MySQL 5.7.2.

You can also obtain information about trigger objects from INFORMATION_SCHEMA, which contains a TRIGGERS table. See Section 19.27, “The INFORMATION_SCHEMA TRIGGERS Table”.

13.7.5.12 SHOW CREATE VIEW Syntax

```
SHOW CREATE VIEW view_name
```

This statement shows the CREATE VIEW statement that creates the named view.

```
mysql> SHOW CREATE VIEW v\G
*************************** 1. row ***************************
                View: v
         Create View: CREATE ALGORITHM=UNDEFINED
                      DEFINER=`bob`@`localhost`
                      SQL SECURITY DEFINER VIEW
                      `v` AS select 1 AS `a`,2 AS `b`
character_set_client: latin1
collation_connection: latin1_swedish_ci
```

character_set_client is the session value of the character_set_client system variable when the view was created. collation_connection is the session value of the collation_connection system variable when the view was created.

Use of SHOW CREATE VIEW requires the SHOW VIEW privilege and the SELECT privilege for the view in question.

You can also obtain information about view objects from INFORMATION_SCHEMA, which contains a VIEWS table. See Section 19.29, "The INFORMATION_SCHEMA VIEWS Table".

MySQL lets you use different sql_mode settings to tell the server the type of SQL syntax to support. For example, you might use the ANSI SQL mode to ensure MySQL correctly interprets the standard SQL concatenation operator, the double bar (||), in your queries. If you then create a view that concatenates items, you might worry that changing the sql_mode setting to a value different from ANSI could cause the view to become invalid. But this is not the case. No matter how you write out a view definition, MySQL always stores it the same way, in a canonical form. Here is an example that shows how the server changes a double bar concatenation operator to a CONCAT() function:

```
mysql> SET sql_mode = 'ANSI';
Query OK, 0 rows affected (0.00 sec)

mysql> CREATE VIEW test.v AS SELECT 'a' || 'b' as col1;
Query OK, 0 rows affected (0.01 sec)

mysql> SHOW CREATE VIEW test.v\G
*************************** 1. row ***************************
                View: v
         Create View: CREATE VIEW "v" AS select concat('a','b') AS "col1"
...
1 row in set (0.00 sec)
```

The advantage of storing a view definition in canonical form is that changes made later to the value of sql_mode will not affect the results from the view. However an additional consequence is that comments prior to SELECT are stripped from the definition by the server.

13.7.5.13 SHOW DATABASES Syntax

```
SHOW {DATABASES | SCHEMAS}
    [LIKE 'pattern' | WHERE expr]
```

SHOW DATABASES lists the databases on the MySQL server host. SHOW SCHEMAS is a synonym for SHOW DATABASES. The LIKE clause, if present, indicates which database names to match. The WHERE clause can be given to select rows using more general conditions, as discussed in Section 19.31, "Extensions to SHOW Statements".

You see only those databases for which you have some kind of privilege, unless you have the global SHOW DATABASES privilege. You can also get this list using the mysqlshow command.

If the server was started with the --skip-show-database option, you cannot use this statement at all unless you have the SHOW DATABASES privilege.

MySQL implements databases as directories in the data directory, so this statement simply lists directories in that location. However, the output may include names of directories that do not correspond to actual databases.

13.7.5.14 SHOW ENGINE Syntax

```
SHOW ENGINE engine_name {STATUS | MUTEX}
```

SHOW ENGINE displays operational information about a storage engine. It requires the PROCESS privilege. The statement has these variants:

```
SHOW ENGINE INNODB STATUS
SHOW ENGINE INNODB MUTEX
SHOW ENGINE PERFORMANCE_SCHEMA STATUS
```

SHOW ENGINE INNODB STATUS displays extensive information from the standard InnoDB Monitor about the state of the InnoDB storage engine. For information about the standard monitor and other InnoDB Monitors that provide information about InnoDB processing, see Section 14.2.12.4, “InnoDB Monitors”.

SHOW ENGINE INNODB MUTEX displays InnoDB mutex and rw-lock statistics. Statement output has the following columns:

Note

SHOW ENGINE INNODB MUTEX output is removed in MySQL 5.7.2. Comparable information can be generated by creating views on Performance Schema tables.

- Type

  Always InnoDB.

- Name

  The source file where the mutex is implemented, and the line number in the file where the mutex is created. The line number is specific to your version of MySQL.

- Status

  The mutex status. This field displays several values if WITH_DEBUG was defined at MySQL compilation time. If WITH_DEBUG was not defined, the statement displays only the os_waits value. In the latter case (without WITH_DEBUG), the information on which the output is based is insufficient to distinguish regular mutexes and mutexes that protect rw-locks (which permit multiple readers or a single writer). Consequently, the output may appear to contain multiple rows for the same mutex.

  - count indicates how many times the mutex was requested.
  - spin_waits indicates how many times the spinlock had to run.
  - spin_rounds indicates the number of spinlock rounds. (spin_rounds divided by spin_waits provides the average round count.)

- os_waits indicates the number of operating system waits. This occurs when the spinlock did not work (the mutex was not locked during the spinlock and it was necessary to yield to the operating system and wait).
- os_yields indicates the number of times a thread trying to lock a mutex gave up its timeslice and yielded to the operating system (on the presumption that permitting other threads to run will free the mutex so that it can be locked).
- os_wait_times indicates the amount of time (in ms) spent in operating system waits. In MySQL 5.7 timing is disabled and this value is always 0.

SHOW ENGINE INNODB MUTEX skips the mutexes and rw-locks of buffer pool blocks, as the amount of output can be overwhelming on systems with a large buffer pool. (There is one mutex and one rw-lock in each 16K buffer pool block, and there are 65,536 blocks per gigabyte.) SHOW ENGINE INNODB MUTEX also does not list any mutexes or rw-locks that have never been waited on (os_waits=0). Thus, SHOW ENGINE INNODB MUTEX only displays information about mutexes and rw-locks outside of the buffer pool that have caused at least one OS-level wait.

SHOW ENGINE INNODB MUTEX information can be used to diagnose system problems. For example, large values of spin_waits and spin_rounds may indicate scalability problems.

Use SHOW ENGINE PERFORMANCE_SCHEMA STATUS to inspect the internal operation of the Performance Schema code:

```
mysql> SHOW ENGINE PERFORMANCE_SCHEMA STATUS\G
...
*************************** 3. row ***************************
  Type: performance_schema
  Name: events_waits_history.size
Status: 76
*************************** 4. row ***************************
  Type: performance_schema
  Name: events_waits_history.count
Status: 10000
*************************** 5. row ***************************
  Type: performance_schema
  Name: events_waits_history.memory
Status: 760000
...
*************************** 57. row ***************************
  Type: performance_schema
  Name: performance_schema.memory
Status: 26459600
...
```

This statement is intended to help the DBA understand the effects that different Performance Schema options have on memory requirements.

Name values consist of two parts, which name an internal buffer and a buffer attribute, respectively. Interpret buffer names as follows:

- An internal buffer that is not exposed as a table is named within parentheses. Examples: (pfs_cond_class).size, (pfs_mutex_class).memory.
- An internal buffer that is exposed as a table in the performance_schema database is named after the table, without parentheses. Examples: events_waits_history.size, mutex_instances.count.
- A value that applies to the Performance Schema as a whole begins with performance_schema. Example: performance_schema.memory.

Buffer attributes have these meanings:

- size is the size of the internal record used by the implementation, such as the size of a row in a table. size values cannot be changed.
- count is the number of internal records, such as the number of rows in a table. count values can be changed using Performance Schema configuration options.
- For a table, *tbl_name*.memory is the product of size and count. For the Performance Schema as a whole, performance_schema.memory is the sum of all the memory used (the sum of all other memory values).

Some size and count attributes were named row_size and row_count before MySQL 5.7.1.

In some cases, there is a direct relationship between a Performance Schema configuration parameter and a SHOW ENGINE value. For example, events_waits_history_long.count corresponds to performance_schema_events_waits_history_long_size. In other cases, the relationship is more complex. For example, events_waits_history.count corresponds to performance_schema_events_waits_history_size (the number of rows per thread) multiplied by performance_schema_max_thread_instances (the number of threads).

13.7.5.15 SHOW ENGINES Syntax

```
SHOW [STORAGE] ENGINES
```

SHOW ENGINES displays status information about the server's storage engines. This is particularly useful for checking whether a storage engine is supported, or to see what the default engine is. This information can also be obtained from the INFORMATION_SCHEMA ENGINES table. See Section 19.6, “The INFORMATION_SCHEMA ENGINES Table”.

```
mysql> SHOW ENGINES\G
*************************** 1. row ***************************
      Engine: MEMORY
     Support: YES
     Comment: Hash based, stored in memory, useful for temporary tables
Transactions: NO
          XA: NO
  Savepoints: NO
*************************** 2. row ***************************
      Engine: MyISAM
     Support: YES
     Comment: MyISAM storage engine
Transactions: NO
          XA: NO
  Savepoints: NO
*************************** 3. row ***************************
      Engine: InnoDB
     Support: DEFAULT
     Comment: Supports transactions, row-level locking, and foreign keys
Transactions: YES
          XA: YES
  Savepoints: YES
*************************** 4. row ***************************
      Engine: EXAMPLE
     Support: YES
     Comment: Example storage engine
Transactions: NO
          XA: NO
  Savepoints: NO
*************************** 5. row ***************************
```

```
      Engine: ARCHIVE
     Support: YES
     Comment: Archive storage engine
Transactions: NO
          XA: NO
  Savepoints: NO
*************************** 6. row ***************************
      Engine: CSV
     Support: YES
     Comment: CSV storage engine
Transactions: NO
          XA: NO
  Savepoints: NO
*************************** 7. row ***************************
      Engine: BLACKHOLE
     Support: YES
     Comment: /dev/null storage engine (anything you write »
              to it disappears)
Transactions: NO
          XA: NO
  Savepoints: NO
*************************** 8. row ***************************
      Engine: FEDERATED
     Support: YES
     Comment: Federated MySQL storage engine
Transactions: NO
          XA: NO
  Savepoints: NO
*************************** 9. row ***************************
      Engine: MRG_MYISAM
     Support: YES
     Comment: Collection of identical MyISAM tables
Transactions: NO
          XA: NO
  Savepoints: NO
```

The output from SHOW ENGINES may vary according to the MySQL version used and other factors. The values shown in the Support column indicate the server's level of support for the storage engine, as shown in the following table.

Value	Meaning
YES	The engine is supported and is active
DEFAULT	Like YES, plus this is the default engine
NO	The engine is not supported
DISABLED	The engine is supported but has been disabled

A value of NO means that the server was compiled without support for the engine, so it cannot be enabled at runtime.

A value of DISABLED occurs either because the server was started with an option that disables the engine, or because not all options required to enable it were given. In the latter case, the error log file should contain a reason indicating why the option is disabled. See Section 5.2.2, “The Error Log”.

You might also see DISABLED for a storage engine if the server was compiled to support it, but was started with a --skip-*engine_name* option.

All MySQL servers support MyISAM tables, because MyISAM is the default storage engine. It is not possible to disable MyISAM.

The Transactions, XA, and Savepoints columns indicate whether the storage engine supports transactions, XA transactions, and savepoints, respectively.

13.7.5.16 SHOW ERRORS Syntax

```
SHOW ERRORS [LIMIT [offset,] row_count]
SHOW COUNT(*) ERRORS
```

SHOW ERRORS is a diagnostic statement that is similar to SHOW WARNINGS, except that it displays information only for errors, rather than for errors, warnings, and notes.

The LIMIT clause has the same syntax as for the SELECT statement. See Section 13.2.9, "SELECT Syntax".

The SHOW COUNT(*) ERRORS statement displays the number of errors. You can also retrieve this number from the error_count variable:

```
SHOW COUNT(*) ERRORS;
SELECT @@error_count;
```

SHOW ERRORS and error_count apply only to errors, not warnings or notes. In other respects, they are similar to SHOW WARNINGS and warning_count. In particular, SHOW ERRORS cannot display information for more than max_error_count messages, and error_count can exceed the value of max_error_count if the number of errors exceeds max_error_count.

For more information, see Section 13.7.5.39, "SHOW WARNINGS Syntax".

13.7.5.17 SHOW EVENTS Syntax

```
SHOW EVENTS [{FROM | IN} schema_name]
    [LIKE 'pattern' | WHERE expr]
```

This statement displays information about Event Manager events. It requires the EVENT privilege for the database from which the events are to be shown.

In its simplest form, SHOW EVENTS lists all of the events in the current schema:

```
mysql> SELECT CURRENT_USER(), SCHEMA();
+----------------+----------+
| CURRENT_USER() | SCHEMA() |
+----------------+----------+
| jon@ghidora    | myschema |
+----------------+----------+
1 row in set (0.00 sec)

mysql> SHOW EVENTS\G
*************************** 1. row ***************************
                  Db: myschema
                Name: e_daily
             Definer: jon@ghidora
           Time zone: SYSTEM
                Type: RECURRING
          Execute at: NULL
      Interval value: 10
      Interval field: SECOND
              Starts: 2006-02-09 10:41:23
                Ends: NULL
              Status: ENABLED
          Originator: 0
character_set_client: latin1
collation_connection: latin1_swedish_ci
```

```
  Database Collation: latin1_swedish_ci
```

To see events for a specific schema, use the FROM clause. For example, to see events for the test schema, use the following statement:

```
SHOW EVENTS FROM test;
```

The LIKE clause, if present, indicates which event names to match. The WHERE clause can be given to select rows using more general conditions, as discussed in Section 19.31, “Extensions to SHOW Statements”.

SHOW EVENTS output has the following columns:

- Db: The schema (database) on which the event is defined.
- Name: The name of the event.
- Time zone: The event time zone, which is the time zone used for scheduling the event and that is in effect within the event as it executes. The default value is SYSTEM.
- Definer: The account of the user who created the event, in '*user_name*'@'*host_name*' format.
- Type: The event repetition type, either ONE TIME (transient) or RECURRING (repeating).
- Execute At: The date and time when a transient event is set to execute. Shown as a DATETIME value.

  For a recurring event, the value of this column is always NULL.
- Interval Value: For a recurring event, the number of intervals to wait between event executions.

  For a transient event, the value of this column is always NULL.
- Interval Field: The time units used for the interval which a recurring event waits before repeating.

  For a transient event, the value of this column is always NULL.
- Starts: The start date and time for a recurring event. This is displayed as a DATETIME value, and is NULL if no start date and time are defined for the event.

  For a transient event, this column is always NULL.
- Ends: The end date and time for a recurring event. This is displayed as a DATETIME value, and defaults to NULL if no end date and time is defined for the event.

  For a transient event, this column is always NULL.
- Status: The event status. One of ENABLED, DISABLED, or SLAVESIDE_DISABLED.

  SLAVESIDE_DISABLED indicates that the creation of the event occurred on another MySQL server acting as a replication master and replicated to the current MySQL server which is acting as a slave, but the event is not presently being executed on the slave.
- Originator: The server ID of the MySQL server on which the event was created. Defaults to 0.
- character_set_client is the session value of the character_set_client system variable when the routine was created. collation_connection is the session value of the collation_connection system variable when the routine was created. Database Collation is the collation of the database with which the routine is associated.

For more information about SLAVE_DISABLED and the Originator column, see Section 16.4.1.11, “Replication of Invoked Features”.

The event action statement is not shown in the output of SHOW EVENTS. Use SHOW CREATE EVENT or the INFORMATION_SCHEMA.EVENTS table.

Times displayed by SHOW EVENTS are given in the event time zone, as discussed in Section 18.4.4, “Event Metadata”.

The columns in the output of SHOW EVENTS are similar to, but not identical to the columns in the INFORMATION_SCHEMA.EVENTS table. See Section 19.7, “The INFORMATION_SCHEMA EVENTS Table”.

13.7.5.18 SHOW FUNCTION CODE Syntax

```
SHOW FUNCTION CODE func_name
```

This statement is similar to SHOW PROCEDURE CODE but for stored functions. See Section 13.7.5.26, “SHOW PROCEDURE CODE Syntax”.

13.7.5.19 SHOW FUNCTION STATUS Syntax

```
SHOW FUNCTION STATUS
    [LIKE 'pattern' | WHERE expr]
```

This statement is similar to SHOW PROCEDURE STATUS but for stored functions. See Section 13.7.5.27, “SHOW PROCEDURE STATUS Syntax”.

13.7.5.20 SHOW GRANTS Syntax

```
SHOW GRANTS [FOR user]
```

This statement lists the GRANT statement or statements that must be issued to duplicate the privileges that are granted to a MySQL user account. The account is named using the same format as for the GRANT statement; for example, 'jeffrey'@'localhost'. If you specify only the user name part of the account name, a host name part of '%' is used. For additional information about specifying account names, see Section 13.7.1.4, “GRANT Syntax”.

```
mysql> SHOW GRANTS FOR 'root'@'localhost';
+---------------------------------------------------------------------+
| Grants for root@localhost                                           |
+---------------------------------------------------------------------+
| GRANT ALL PRIVILEGES ON *.* TO 'root'@'localhost' WITH GRANT OPTION |
+---------------------------------------------------------------------+
```

To list the privileges granted to the account that you are using to connect to the server, you can use any of the following statements:

```
SHOW GRANTS;
SHOW GRANTS FOR CURRENT_USER;
SHOW GRANTS FOR CURRENT_USER();
```

If SHOW GRANTS FOR CURRENT_USER (or any of the equivalent syntaxes) is used in DEFINER context, such as within a stored procedure that is defined with SQL SECURITY DEFINER), the grants displayed are those of the definer and not the invoker.

SHOW GRANTS displays only the privileges granted explicitly to the named account. Other privileges might be available to the account, but they are not displayed. For example, if an anonymous account exists, the named account might be able to use its privileges, but SHOW GRANTS will not display them.

SHOW GRANTS requires the SELECT privilege for the mysql database, except to see the privileges for the current user.

13.7.5.21 SHOW INDEX Syntax

```
SHOW {INDEX | INDEXES | KEYS}
    {FROM | IN} tbl_name
    [{FROM | IN} db_name]
    [WHERE expr]
```

SHOW INDEX returns table index information. The format resembles that of the SQLStatistics call in ODBC. This statement requires some privilege for any column in the table.

SHOW INDEX returns the following fields:

- Table

  The name of the table.

- Non_unique

  0 if the index cannot contain duplicates, 1 if it can.

- Key_name

  The name of the index. If the index is the primary key, the name is always PRIMARY.

- Seq_in_index

  The column sequence number in the index, starting with 1.

- Column_name

  The column name.

- Collation

  How the column is sorted in the index. In MySQL, this can have values “A” (Ascending) or NULL (Not sorted).

- Cardinality

  An estimate of the number of unique values in the index. This is updated by running ANALYZE TABLE or myisamchk -a. Cardinality is counted based on statistics stored as integers, so the value is not necessarily exact even for small tables. The higher the cardinality, the greater the chance that MySQL uses the index when doing joins.

- Sub_part

  The number of indexed characters if the column is only partly indexed, NULL if the entire column is indexed.

- Packed

  Indicates how the key is packed. NULL if it is not.

- Null

  Contains YES if the column may contain NULL values and '' if not.

- Index_type

  The index method used (BTREE, FULLTEXT, HASH, RTREE).

- Comment

  Information about the index not described in its own column, such as disabled if the index is disabled.

- Index_comment

  Any comment provided for the index with a COMMENT attribute when the index was created.

You can use *db_name*.*tbl_name* as an alternative to the *tbl_name* FROM *db_name* syntax. These two statements are equivalent:

```
SHOW INDEX FROM mytable FROM mydb;
SHOW INDEX FROM mydb.mytable;
```

The WHERE clause can be given to select rows using more general conditions, as discussed in Section 19.31, “Extensions to SHOW Statements”.

You can also list a table's indexes with the mysqlshow -k *db_name* *tbl_name* command.

13.7.5.22 SHOW MASTER STATUS Syntax

```
SHOW MASTER STATUS
```

This statement provides status information about the binary log files of the master. It requires either the SUPER or REPLICATION CLIENT privilege.

Example:

```
mysql> SHOW MASTER STATUS\G
*************************** 1. row ***************************
             File: master-bin.000002
         Position: 1307
     Binlog_Do_DB: test
 Binlog_Ignore_DB: manual, mysql
Executed_Gtid_Set: 3E11FA47-71CA-11E1-9E33-C80AA9429562:1-5
1 row in set (0.00 sec)
```

When global transaction IDs are in use, Executed_Gtid_Set shows the set of GTIDs for transactions that have been executed on the master. This is the same as the master's value for the global gtid_executed system variable, as well as the slave's value for Executed_Gtid_Set in the output of SHOW SLAVE STATUS.

13.7.5.23 SHOW OPEN TABLES Syntax

```
SHOW OPEN TABLES [{FROM | IN} db_name]
    [LIKE 'pattern' | WHERE expr]
```

SHOW OPEN TABLES lists the non-TEMPORARY tables that are currently open in the table cache. See Section 8.4.3.1, “How MySQL Opens and Closes Tables”. The FROM clause, if present, restricts the tables shown to those present in the *db_name* database. The LIKE clause, if present, indicates which table names to match. The WHERE clause can be given to select rows using more general conditions, as discussed in Section 19.31, “Extensions to SHOW Statements”.

SHOW OPEN TABLES output has the following columns:

- Database

  The database containing the table.

- Table

  The table name.

- In_use

  The number of table locks or lock requests there are for the table. For example, if one client acquires a lock for a table using LOCK TABLE t1 WRITE, In_use will be 1. If another client issues LOCK TABLE t1 WRITE while the table remains locked, the client will block waiting for the lock, but the lock request causes In_use to be 2. If the count is zero, the table is open but not currently being used. In_use is also increased by the HANDLER ... OPEN statement and decreased by HANDLER ... CLOSE.

- Name_locked

  Whether the table name is locked. Name locking is used for operations such as dropping or renaming tables.

If you have no privileges for a table, it does not show up in the output from SHOW OPEN TABLES.

13.7.5.24 SHOW PLUGINS Syntax

```
SHOW PLUGINS
```

SHOW PLUGINS displays information about server plugins. Plugin information is also available in the INFORMATION_SCHEMA.PLUGINS table. See Section 19.15, “The INFORMATION_SCHEMA PLUGINS Table”.

Example of SHOW PLUGINS output:

```
mysql> SHOW PLUGINS\G
*************************** 1. row ***************************
   Name: binlog
 Status: ACTIVE
   Type: STORAGE ENGINE
Library: NULL
License: GPL
*************************** 2. row ***************************
   Name: CSV
 Status: ACTIVE
   Type: STORAGE ENGINE
Library: NULL
License: GPL
*************************** 3. row ***************************
   Name: MEMORY
 Status: ACTIVE
   Type: STORAGE ENGINE
Library: NULL
License: GPL
*************************** 4. row ***************************
```

```
   Name: MyISAM
 Status: ACTIVE
   Type: STORAGE ENGINE
Library: NULL
License: GPL
...
```

SHOW PLUGINS output has the following columns:

- Name: The name used to refer to the plugin in statements such as INSTALL PLUGIN and UNINSTALL PLUGIN.
- Status: The plugin status, one of ACTIVE, INACTIVE, DISABLED, or DELETED.
- Type: The type of plugin, such as STORAGE ENGINE, INFORMATION_SCHEMA, or AUTHENTICATION.
- Library: The name of the plugin shared object file. This is the name used to refer to the plugin file in statements such as INSTALL PLUGIN and UNINSTALL PLUGIN. This file is located in the directory named by the plugin_dir system variable. If the library name is NULL, the plugin is compiled in and cannot be uninstalled with UNINSTALL PLUGIN.
- License: How the plugin is licensed; for example, GPL.

For plugins installed with INSTALL PLUGIN, the Name and Library values are also registered in the mysql.plugin table.

For information about plugin data structures that form the basis of the information displayed by SHOW PLUGINS, see Section 22.2, “The MySQL Plugin API”.

13.7.5.25 SHOW PRIVILEGES Syntax

```
SHOW PRIVILEGES
```

SHOW PRIVILEGES shows the list of system privileges that the MySQL server supports. The exact list of privileges depends on the version of your server.

```
mysql> SHOW PRIVILEGES\G
*************************** 1. row ***************************
Privilege: Alter
  Context: Tables
  Comment: To alter the table
*************************** 2. row ***************************
Privilege: Alter routine
  Context: Functions,Procedures
  Comment: To alter or drop stored functions/procedures
*************************** 3. row ***************************
Privilege: Create
  Context: Databases,Tables,Indexes
  Comment: To create new databases and tables
*************************** 4. row ***************************
Privilege: Create routine
  Context: Databases
  Comment: To use CREATE FUNCTION/PROCEDURE
*************************** 5. row ***************************
Privilege: Create temporary tables
  Context: Databases
  Comment: To use CREATE TEMPORARY TABLE
...
```

Privileges belonging to a specific user are displayed by the SHOW GRANTS statement. See Section 13.7.5.20, “SHOW GRANTS Syntax”, for more information.

13.7.5.26 SHOW PROCEDURE CODE Syntax

```
SHOW PROCEDURE CODE proc_name
```

This statement is a MySQL extension that is available only for servers that have been built with debugging support. It displays a representation of the internal implementation of the named stored procedure. A similar statement, SHOW FUNCTION CODE, displays information about stored functions (see Section 13.7.5.18, “SHOW FUNCTION CODE Syntax”).

To use either statement, you must be the owner of the routine or have SELECT access to the mysql.proc table.

If the named routine is available, each statement produces a result set. Each row in the result set corresponds to one “instruction” in the routine. The first column is Pos, which is an ordinal number beginning with 0. The second column is Instruction, which contains an SQL statement (usually changed from the original source), or a directive which has meaning only to the stored-routine handler.

```
mysql> DELIMITER //
mysql> CREATE PROCEDURE p1 ()
    -> BEGIN
    ->   DECLARE fanta INT DEFAULT 55;
    ->   DROP TABLE t2;
    ->   LOOP
    ->     INSERT INTO t3 VALUES (fanta);
    ->     END LOOP;
    ->   END//
Query OK, 0 rows affected (0.00 sec)

mysql> SHOW PROCEDURE CODE p1//
+-----+----------------------------------------+
| Pos | Instruction                            |
+-----+----------------------------------------+
|   0 | set fanta@0 55                         |
|   1 | stmt 9 "DROP TABLE t2"                 |
|   2 | stmt 5 "INSERT INTO t3 VALUES (fanta)" |
|   3 | jump 2                                 |
+-----+----------------------------------------+
4 rows in set (0.00 sec)
```

In this example, the nonexecutable BEGIN and END statements have disappeared, and for the DECLARE *variable_name* statement, only the executable part appears (the part where the default is assigned). For each statement that is taken from source, there is a code word stmt followed by a type (9 means DROP, 5 means INSERT, and so on). The final row contains an instruction jump 2, meaning GOTO instruction #2.

13.7.5.27 SHOW PROCEDURE STATUS Syntax

```
SHOW PROCEDURE STATUS
    [LIKE 'pattern' | WHERE expr]
```

This statement is a MySQL extension. It returns characteristics of a stored procedure, such as the database, name, type, creator, creation and modification dates, and character set information. A similar statement, SHOW FUNCTION STATUS, displays information about stored functions (see Section 13.7.5.19, “SHOW FUNCTION STATUS Syntax”).

The LIKE clause, if present, indicates which procedure or function names to match. The WHERE clause can be given to select rows using more general conditions, as discussed in Section 19.31, “Extensions to SHOW Statements”.

```
mysql> SHOW PROCEDURE STATUS LIKE 'sp1'\G
*************************** 1. row ***************************
                  Db: test
                Name: sp1
                Type: PROCEDURE
             Definer: testuser@localhost
            Modified: 2004-08-03 15:29:37
             Created: 2004-08-03 15:29:37
       Security_type: DEFINER
             Comment:
character_set_client: latin1
collation_connection: latin1_swedish_ci
  Database Collation: latin1_swedish_ci
```

character_set_client is the session value of the character_set_client system variable when the routine was created. collation_connection is the session value of the collation_connection system variable when the routine was created. Database Collation is the collation of the database with which the routine is associated.

You can also get information about stored routines from the ROUTINES table in INFORMATION_SCHEMA. See Section 19.19, "The INFORMATION_SCHEMA ROUTINES Table".

13.7.5.28 SHOW PROCESSLIST Syntax

```
SHOW [FULL] PROCESSLIST
```

SHOW PROCESSLIST shows you which threads are running. You can also get this information from the INFORMATION_SCHEMA PROCESSLIST table or the mysqladmin processlist command. If you have the PROCESS privilege, you can see all threads. Otherwise, you can see only your own threads (that is, threads associated with the MySQL account that you are using). If you do not use the FULL keyword, only the first 100 characters of each statement are shown in the Info field.

Process information is also available from the performance_schema.threads table. However, access to threads does not require a mutex and has minimal impact on server performance. INFORMATION_SCHEMA.PROCESSLIST and SHOW PROCESSLIST have negative performance consequences because they require a mutex. threads also shows information about background threads, which INFORMATION_SCHEMA.PROCESSLIST and SHOW PROCESSLIST do not. This means that threads can be used to monitor activity the other thread information sources cannot.

The SHOW PROCESSLIST statement is very useful if you get the "too many connections" error message and want to find out what is going on. MySQL reserves one extra connection to be used by accounts that have the SUPER privilege, to ensure that administrators should always be able to connect and check the system (assuming that you are not giving this privilege to all your users).

Threads can be killed with the KILL statement. See Section 13.7.6.4, "KILL Syntax".

Here is an example of SHOW PROCESSLIST output:

```
mysql> SHOW FULL PROCESSLIST\G
*************************** 1. row ***************************
Id: 1
User: system user
Host:
db: NULL
Command: Connect
Time: 1030455
State: Waiting for master to send event
Info: NULL
```

```
*************************** 2. row ***************************
Id: 2
User: system user
Host:
db: NULL
Command: Connect
Time: 1004
State: Has read all relay log; waiting for the slave
       I/O thread to update it
Info: NULL
*************************** 3. row ***************************
Id: 3112
User: replikator
Host: artemis:2204
db: NULL
Command: Binlog Dump
Time: 2144
State: Has sent all binlog to slave; waiting for binlog to be updated
Info: NULL
*************************** 4. row ***************************
Id: 3113
User: replikator
Host: iconnect2:45781
db: NULL
Command: Binlog Dump
Time: 2086
State: Has sent all binlog to slave; waiting for binlog to be updated
Info: NULL
*************************** 5. row ***************************
Id: 3123
User: stefan
Host: localhost
db: apollon
Command: Query
Time: 0
State: NULL
Info: SHOW FULL PROCESSLIST
5 rows in set (0.00 sec)
```

The columns produced by SHOW PROCESSLIST have the following meanings:

- Id

  The connection identifier.

- User

  The MySQL user who issued the statement. If this is system user, it refers to a nonclient thread spawned by the server to handle tasks internally. This could be the I/O or SQL thread used on replication slaves or a delayed-row handler. unauthenticated user refers to a thread that has become associated with a client connection but for which authentication of the client user has not yet been done. event_scheduler refers to the thread that monitors scheduled events. For system user, there is no host specified in the Host column.

- Host

  The host name of the client issuing the statement (except for system user where there is no host). SHOW PROCESSLIST reports the host name for TCP/IP connections in *host_name*:*client_port* format to make it easier to determine which client is doing what.

- db

  The default database, if one is selected, otherwise NULL.

- Command

  The type of command the thread is executing. For descriptions for thread commands, see Section 8.12.5, “Examining Thread Information”. The value of this column corresponds to the COM_*xxx* commands of the client/server protocol and Com_*xxx* status variables. See Section 5.1.6, “Server Status Variables”

- Time

  The time in seconds that the thread has been in its current state. For a slave SQL thread, the value is the number of seconds between the timestamp of the last replicated event and the real time of the slave machine. See Section 16.2.1, “Replication Implementation Details”.

- State

  An action, event, or state that indicates what the thread is doing. Descriptions for State values can be found at Section 8.12.5, “Examining Thread Information”.

  Most states correspond to very quick operations. If a thread stays in a given state for many seconds, there might be a problem that needs to be investigated.

  For the SHOW PROCESSLIST statement, the value of State is NULL.

- Info

  The statement the thread is executing, or NULL if it is not executing any statement. The statement might be the one sent to the server, or an innermost statement if the statement executes other statements. For example, if a CALL statement executes a stored procedure that is executing a SELECT statement, the Info value shows the SELECT statement.

13.7.5.29 SHOW PROFILE Syntax

```
SHOW PROFILE [type [, type] ... ]
    [FOR QUERY n]
    [LIMIT row_count [OFFSET offset]]

type:
    ALL
  | BLOCK IO
  | CONTEXT SWITCHES
  | CPU
  | IPC
  | MEMORY
  | PAGE FAULTS
  | SOURCE
  | SWAPS
```

The SHOW PROFILE and SHOW PROFILES statements display profiling information that indicates resource usage for statements executed during the course of the current session.

Note

These statements are deprecated and will be removed in a future MySQL release. Use the Performance Schema instead; see Chapter 20, *MySQL Performance Schema*.

Profiling is controlled by the profiling session variable, which has a default value of 0 (OFF). Profiling is enabled by setting profiling to 1 or ON:

```
mysql> SET profiling = 1;
```

SHOW PROFILES displays a list of the most recent statements sent to the server. The size of the list is controlled by the profiling_history_size session variable, which has a default value of 15. The maximum value is 100. Setting the value to 0 has the practical effect of disabling profiling.

All statements are profiled except SHOW PROFILE and SHOW PROFILES, so you will find neither of those statements in the profile list. Malformed statements are profiled. For example, SHOW PROFILING is an illegal statement, and a syntax error occurs if you try to execute it, but it will show up in the profiling list.

SHOW PROFILE displays detailed information about a single statement. Without the FOR QUERY *n* clause, the output pertains to the most recently executed statement. If FOR QUERY *n* is included, SHOW PROFILE displays information for statement *n*. The values of *n* correspond to the Query_ID values displayed by SHOW PROFILES.

The LIMIT *row_count* clause may be given to limit the output to *row_count* rows. If LIMIT is given, OFFSET *offset* may be added to begin the output *offset* rows into the full set of rows.

By default, SHOW PROFILE displays Status and Duration columns. The Status values are like the State values displayed by SHOW PROCESSLIST, although there might be some minor differences in interpretion for the two statements for some status values (see Section 8.12.5, “Examining Thread Information”).

Optional *type* values may be specified to display specific additional types of information:

- ALL displays all information
- BLOCK IO displays counts for block input and output operations
- CONTEXT SWITCHES displays counts for voluntary and involuntary context switches
- CPU displays user and system CPU usage times
- IPC displays counts for messages sent and received
- MEMORY is not currently implemented
- PAGE FAULTS displays counts for major and minor page faults
- SOURCE displays the names of functions from the source code, together with the name and line number of the file in which the function occurs
- SWAPS displays swap counts

Profiling is enabled per session. When a session ends, its profiling information is lost.

```
mysql> SELECT @@profiling;
+-------------+
| @@profiling |
+-------------+
|           0 |
+-------------+
1 row in set (0.00 sec)

mysql> SET profiling = 1;
Query OK, 0 rows affected (0.00 sec)

mysql> DROP TABLE IF EXISTS t1;
```

```
Query OK, 0 rows affected, 1 warning (0.00 sec)

mysql> CREATE TABLE T1 (id INT);
Query OK, 0 rows affected (0.01 sec)

mysql> SHOW PROFILES;
+----------+----------+--------------------------+
| Query_ID | Duration | Query                    |
+----------+----------+--------------------------+
|        0 | 0.000088 | SET PROFILING = 1        |
|        1 | 0.000136 | DROP TABLE IF EXISTS t1  |
|        2 | 0.011947 | CREATE TABLE t1 (id INT) |
+----------+----------+--------------------------+
3 rows in set (0.00 sec)

mysql> SHOW PROFILE;
+----------------------+----------+
| Status               | Duration |
+----------------------+----------+
| checking permissions | 0.000040 |
| creating table       | 0.000056 |
| After create         | 0.011363 |
| query end            | 0.000375 |
| freeing items        | 0.000089 |
| logging slow query   | 0.000019 |
| cleaning up          | 0.000005 |
+----------------------+----------+
7 rows in set (0.00 sec)

mysql> SHOW PROFILE FOR QUERY 1;
+--------------------+----------+
| Status             | Duration |
+--------------------+----------+
| query end          | 0.000107 |
| freeing items      | 0.000008 |
| logging slow query | 0.000015 |
| cleaning up        | 0.000006 |
+--------------------+----------+
4 rows in set (0.00 sec)

mysql> SHOW PROFILE CPU FOR QUERY 2;
+----------------------+----------+----------+------------+
| Status               | Duration | CPU_user | CPU_system |
+----------------------+----------+----------+------------+
| checking permissions | 0.000040 | 0.000038 |   0.000002 |
| creating table       | 0.000056 | 0.000028 |   0.000028 |
| After create         | 0.011363 | 0.000217 |   0.001571 |
| query end            | 0.000375 | 0.000013 |   0.000028 |
| freeing items        | 0.000089 | 0.000010 |   0.000014 |
| logging slow query   | 0.000019 | 0.000009 |   0.000010 |
| cleaning up          | 0.000005 | 0.000003 |   0.000002 |
+----------------------+----------+----------+------------+
7 rows in set (0.00 sec)
```

Note

Profiling is only partially functional on some architectures. For values that depend on the getrusage() system call, NULL is returned on systems such as Windows that do not support the call. In addition, profiling is per process and not per thread. This means that activity on threads within the server other than your own may affect the timing information that you see.

You can also get profiling information from the PROFILING table in INFORMATION_SCHEMA. See Section 19.17, “The INFORMATION_SCHEMA PROFILING Table”. For example, the following queries produce the same result:

```
SHOW PROFILE FOR QUERY 2;

SELECT STATE, FORMAT(DURATION, 6) AS DURATION
FROM INFORMATION_SCHEMA.PROFILING
WHERE QUERY_ID = 2 ORDER BY SEQ;
```

13.7.5.30 SHOW PROFILES Syntax

```
SHOW PROFILES
```

The SHOW PROFILES statement, together with SHOW PROFILE, displays profiling information that indicates resource usage for statements executed during the course of the current session. For more information, see Section 13.7.5.29, "SHOW PROFILE Syntax".

> **Note**
>
> These statements are deprecated and will be removed in a future MySQL release. Use the Performance Schema instead; see Chapter 20, *MySQL Performance Schema*.

13.7.5.31 SHOW RELAYLOG EVENTS Syntax

```
SHOW RELAYLOG EVENTS
   [IN 'log_name'] [FROM pos] [LIMIT [offset,] row_count]
```

Shows the events in the relay log of a replication slave. If you do not specify '*log_name*', the first relay log is displayed. This statement has no effect on the master.

The LIMIT clause has the same syntax as for the SELECT statement. See Section 13.2.9, "SELECT Syntax".

> **Note**
>
> Issuing a SHOW RELAYLOG EVENTS with no LIMIT clause could start a very time- and resource-consuming process because the server returns to the client the complete contents of the relay log (including all statements modifying data that have been received by the slave).

> **Note**
>
> Some events relating to the setting of user and system variables are not included in the output from SHOW RELAYLOG EVENTS. To get complete coverage of events within a relay log, use mysqlbinlog.

13.7.5.32 SHOW SLAVE HOSTS Syntax

```
SHOW SLAVE HOSTS
```

Displays a list of replication slaves currently registered with the master.

SHOW SLAVE HOSTS should be executed on a server that acts as a replication master. The statement displays information about servers that are or have been connected as replication slaves, with each row of the result corresponding to one slave server, as shown here:

```
mysql> SHOW SLAVE HOSTS;
+-----------+-----------+-------+-----------+--------------------------------------+
| Server_id | Host      | Port  | Master_id | Slave_UUID                           |
+-----------+-----------+-------+-----------+--------------------------------------+
|  192168010 | iconnect2 | 3306 | 192168011 | 14cb6624-7f93-11e0-b2c0-c80aa9429562 |
| 1921680101 | athena    | 3306 | 192168011 | 07af4990-f41f-11df-a566-7ac56fdaf645 |
+------------+-----------+------+-----------+--------------------------------------+
```

- Server_id: The unique server ID of the slave server, as configured in the slave server's option file, or on the command line with --server-id=*value* [2162].
- Host: The host name of the slave server, as configured in the slave server's option file, or on the command line with --report-host=*host_name*. Note that this can differ from the machine name as configured in the operating system.
- Port: The port on the master to which the slave server is listening.

  A zero in this column means that the slave port (--report-port) was not set.
- Master_id: The unique server ID of the master server that the slave server is replicating from. This is the server ID of the server on which SHOW SLAVE HOSTS is executed, so this same value is listed for each row in the result.
- Slave_UUID: The globally unique ID of this slave, as generated on the slave and found in the slave's auto.cnf file.

13.7.5.33 SHOW SLAVE STATUS Syntax

```
SHOW SLAVE STATUS [NONBLOCKING]
```

This statement provides status information on essential parameters of the slave threads. It requires either the SUPER or REPLICATION CLIENT privilege.

The NONBLOCKING option causes SHOW SLAVE STATUS, when run concurrently with STOP SLAVE, to return without waiting for STOP SLAVE to finish shutting down the slave SQL thread or slave I/O thread (or both). This option is intended for use in monitoring and other applications where getting an immediate response from SHOW SLAVE STATUS is more important than ensuring that it returns the latest data.

If you issue this statement using the mysql client, you can use a \G statement terminator rather than a semicolon to obtain a more readable vertical layout:

```
mysql> SHOW SLAVE STATUS\G
*************************** 1. row ***************************
               Slave_IO_State: Waiting for master to send event
                  Master_Host: localhost
                  Master_User: root
                  Master_Port: 13000
                Connect_Retry: 60
              Master_Log_File: master-bin.000002
          Read_Master_Log_Pos: 1307
               Relay_Log_File: slave-relay-bin.000003
                Relay_Log_Pos: 1508
        Relay_Master_Log_File: master-bin.000002
             Slave_IO_Running: Yes
            Slave_SQL_Running: Yes
              Replicate_Do_DB:
          Replicate_Ignore_DB:
           Replicate_Do_Table:
       Replicate_Ignore_Table:
```

```
      Replicate_Wild_Do_Table:
  Replicate_Wild_Ignore_Table:
                   Last_Errno: 0
                   Last_Error:
                 Skip_Counter: 0
          Exec_Master_Log_Pos: 1307
              Relay_Log_Space: 1858
              Until_Condition: None
               Until_Log_File:
                Until_Log_Pos: 0
           Master_SSL_Allowed: No
           Master_SSL_CA_File:
           Master_SSL_CA_Path:
              Master_SSL_Cert:
            Master_SSL_Cipher:
               Master_SSL_Key:
        Seconds_Behind_Master: 0
Master_SSL_Verify_Server_Cert: No
                Last_IO_Errno: 0
                Last_IO_Error:
               Last_SQL_Errno: 0
               Last_SQL_Error:
  Replicate_Ignore_Server_Ids:
             Master_Server_Id: 1
                  Master_UUID: 3e11fa47-71ca-11e1-9e33-c80aa9429562
             Master_Info_File: /var/mysqld.2/data/master.info
                    SQL_Delay: 0
          SQL_Remaining_Delay: NULL
      Slave_SQL_Running_State: Slave has read all relay log; waiting for the slave I/O thread to update it
           Master_Retry_Count: 10
                  Master_Bind:
      Last_IO_Error_Timestamp:
     Last_SQL_Error_Timestamp:
               Master_SSL_Crl:
           Master_SSL_Crlpath:
           Retrieved_Gtid_Set: 3e11fa47-71ca-11e1-9e33-c80aa9429562:1-5
            Executed_Gtid_Set: 3e11fa47-71ca-11e1-9e33-c80aa9429562:1-5
                Auto_Position: 1
1 row in set (0.00 sec)
```

As of MySQL 5.7.2, the Performance Schema provides tables that expose replication information. This is similar to the information available from the SHOW SLAVE STATUS statement, but represented in table form. For details, see Section 20.9.10, “Performance Schema Replication Tables”.

The following list describes the fields returned by SHOW SLAVE STATUS. For additional information about interpreting their meanings, see Section 16.1.5.1, “Checking Replication Status”.

- Slave_IO_State

  A copy of the State field of the SHOW PROCESSLIST output for the slave I/O thread. This tells you what the thread is doing: trying to connect to the master, waiting for events from the master, reconnecting to the master, and so on. For a listing of possible states, see Section 8.12.5.5, “Replication Slave I/O Thread States”.

- Master_Host

  The master host that the slave is connected to.

- Master_User

  The user name of the account used to connect to the master.

- Master_Port

The port used to connect to the master.

- Connect_Retry

  The number of seconds between connect retries (default 60). This can be set with the CHANGE MASTER TO statement.

- Master_Log_File

  The name of the master binary log file from which the I/O thread is currently reading.

- Read_Master_Log_Pos

  The position in the current master binary log file up to which the I/O thread has read.

- Relay_Log_File

  The name of the relay log file from which the SQL thread is currently reading and executing.

- Relay_Log_Pos

  The position in the current relay log file up to which the SQL thread has read and executed.

- Relay_Master_Log_File

  The name of the master binary log file containing the most recent event executed by the SQL thread.

- Slave_IO_Running

  Whether the I/O thread is started and has connected successfully to the master. Internally, the state of this thread is represented by one of the following three values:

  - **MYSQL_SLAVE_NOT_RUN.** The slave I/O thread is not running. For this state, Slave_IO_Running is No.

  - **MYSQL_SLAVE_RUN_NOT_CONNECT.** The slave I/O thread is running, but is not connected to a replication master. For this state, Slave_IO_Running depends on the server version as shown in the following table.

MySQL Version	Slave_IO_Running
4.1 (4.1.13 and earlier); 5.0 (5.0.11 and earlier)	Yes
4.1 (4.1.14 and later); 5.0 (5.0.12 and later)	No
5.1 (5.1.45 and earlier)	No
5.1 (5.1.46 and later); 5.5; 5.6	Connecting

  - **MYSQL_SLAVE_RUN_CONNECT.** The slave I/O thread is running, and is connected to a replication master. For this state, Slave_IO_Running is Yes.

  The value of the Slave_running system status variable corresponds with this value.

- Slave_SQL_Running

  Whether the SQL thread is started.

- Replicate_Do_DB, Replicate_Ignore_DB

The lists of databases that were specified with the --replicate-do-db and --replicate-ignore-db options, if any.

- Replicate_Do_Table, Replicate_Ignore_Table, Replicate_Wild_Do_Table, Replicate_Wild_Ignore_Table

  The lists of tables that were specified with the --replicate-do-table, --replicate-ignore-table, --replicate-wild-do-table, and --replicate-wild-ignore-table options, if any.

- Last_Errno, Last_Error

  These columns are aliases for Last_SQL_Errno and Last_SQL_Error.

  Issuing RESET MASTER or RESET SLAVE resets the values shown in these columns.

  Note

  When the slave SQL thread receives an error, it reports the error first, then stops the SQL thread. This means that there is a small window of time during which SHOW SLAVE STATUS shows a nonzero value for Last_SQL_Errno even though Slave_SQL_Running still displays Yes.

- Skip_Counter

  The current value of the sql_slave_skip_counter system variable. See Section 13.4.2.5, “SET GLOBAL sql_slave_skip_counter Syntax”.

- Exec_Master_Log_Pos

  The position in the current master binary log file to which the SQL thread has read and executed, marking the start of the next transaction or event to be processed. You can use this value with the CHANGE MASTER TO statement's MASTER_LOG_POS option when starting a new slave from an existing slave, so that the new slave reads from this point. The coordinates given by (Relay_Master_Log_File, Exec_Master_Log_Pos) in the master's binary log correspond to the coordinates given by (Relay_Log_File, Relay_Log_Pos) in the relay log.

  When using a multi-threaded slave (by setting slave_parallel_workers to a nonzero value), the value in this column actually represents a “low-water” mark, before which no uncommitted transactions remain. Because the current implementation allows execution of transactions on different databases in a different order on the slave than on the master, this is not necessarily the position of the most recently executed transaction.

- Relay_Log_Space

  The total combined size of all existing relay log files.

- Until_Condition, Until_Log_File, Until_Log_Pos

  The values specified in the UNTIL clause of the START SLAVE statement.

  Until_Condition has these values:

  - None if no UNTIL clause was specified
  - Master if the slave is reading until a given position in the master's binary log
  - Relay if the slave is reading until a given position in its relay log

Until_Log_File and Until_Log_Pos indicate the log file name and position that define the coordinates at which the SQL thread stops executing.

- Master_SSL_Allowed, Master_SSL_CA_File, Master_SSL_CA_Path, Master_SSL_Cert, Master_SSL_Cipher, Master_SSL_CRL_File, Master_SSL_CRL_Path, Master_SSL_Key, Master_SSL_Verify_Server_Cert

  These fields show the SSL parameters used by the slave to connect to the master, if any.

  Master_SSL_Allowed has these values:

  - Yes if an SSL connection to the master is permitted
  - No if an SSL connection to the master is not permitted
  - Ignored if an SSL connection is permitted but the slave server does not have SSL support enabled

  The values of the other SSL-related fields correspond to the values of the MASTER_SSL_CA, MASTER_SSL_CAPATH, MASTER_SSL_CERT, MASTER_SSL_CIPHER, MASTER_SSL_CRL, MASTER_SSL_CRLPATH, MASTER_SSL_KEY, and MASTER_SSL_VERIFY_SERVER_CERT options to the CHANGE MASTER TO statement. See Section 13.4.2.1, “CHANGE MASTER TO Syntax”.

- Seconds_Behind_Master

  This field is an indication of how “late” the slave is:

  - When the slave is actively processing updates, this field shows the difference between the current timestamp on the slave and the original timestamp logged on the master for the most event currently being processed on the slave.
  - When no event is currently being processed on the slave, this value is 0.

  In essence, this field measures the time difference in seconds between the slave SQL thread and the slave I/O thread. If the network connection between master and slave is fast, the slave I/O thread is very close to the master, so this field is a good approximation of how late the slave SQL thread is compared to the master. If the network is slow, this is *not* a good approximation; the slave SQL thread may quite often be caught up with the slow-reading slave I/O thread, so Seconds_Behind_Master often shows a value of 0, even if the I/O thread is late compared to the master. In other words, *this column is useful only for fast networks*.

  This time difference computation works even if the master and slave do not have identical clock times, provided that the difference, computed when the slave I/O thread starts, remains constant from then on. Any changes—including NTP updates—can lead to clock skews that can make calculation of Seconds_Behind_Master less reliable.

  In MySQL 5.7, this field is NULL (undefined or unknown) if the slave SQL thread is not running, or if the SQL thread has consumed all of the relay log and the slave I/O thread is not running. (In older versions of MySQL, this field was NULL if the slave SQL thread or the slave I/O thread was not running or was not connected to the master.) If the I/O thread is running but the relay log is exhausted, Seconds_Behind_Master is set to 0.

  The value of Seconds_Behind_Master is based on the timestamps stored in events, which are preserved through replication. This means that if a master M1 is itself a slave of M0, any event from M1's binary log that originates from M0's binary log has M0's timestamp for that event. This enables MySQL to replicate TIMESTAMP successfully. However, the problem for Seconds_Behind_Master is that if M1 also receives direct updates from clients, the Seconds_Behind_Master value randomly fluctuates because sometimes the last event from M1 originates from M0 and sometimes is the result of a direct update on M1.

When using a multi-threaded slave, you should keep in mind that this value is based on Exec_Master_Log_Pos, and so may not reflect the position of the most recently committed transaction.

- Last_IO_Errno, Last_IO_Error

  The error number and error message of the most recent error that caused the I/O thread to stop. An error number of 0 and message of the empty string mean “no error.” If the Last_IO_Error value is not empty, the error values also appear in the slave's error log.

  I/O error information includes a timestamp showing when the most recent I/O thread error occurred. This timestamp uses the format YYMMDD HH:MM:SS, and appears in the Last_SQL_Error_Timestamp column.

  Issuing RESET MASTER or RESET SLAVE resets the values shown in these columns.

- Last_SQL_Errno, Last_SQL_Error

  The error number and error message of the most recent error that caused the SQL thread to stop. An error number of 0 and message of the empty string mean “no error.” If the Last_SQL_Error value is not empty, the error values also appear in the slave's error log.

  If the slave is multi-threaded, the SQL thread is the coordinator for worker threads. In this case, as of MySQL 5.7.2, the Last_SQL_Error field shows exactly what the Last_Error_Message column in the Performance Schema replication_execute_status_by_coordinator table shows. The field value is modified to suggest that there may be more failures in the other worker threads which can be seen in the replication_execute_status_by_worker table that shows each worker thread's status. If that table is not available, the slave error log can be used. The log or the replication_execute_status_by_worker table should also be used to learn more about the failure shown by SHOW SLAVE STATUS or the coordinator table.

  SQL error information includes a timestamp showing when the most recent SQL thread error occurred. This timestamp uses the format YYMMDD HH:MM:SS, and appears in the Last_SQL_Error_Timestamp column.

  Issuing RESET MASTER or RESET SLAVE resets the values shown in these columns.

  In MySQL 5.7, all error codes and messages displayed in the Last_SQL_Errno and Last_SQL_Error columns correspond to error values listed in Section C.3, “Server Error Codes and Messages”. This was not always true in previous versions. (Bug #11760365, Bug #52768)

- Replicate_Ignore_Server_Ids

  In MySQL 5.7, you can tell a slave to ignore events from 0 or more masters using the IGNORE_SERVER_IDS option of the CHANGE MASTER TO statement. This is normally of interest only when using a circular or other multi-master replication setup.

  The message shown for Replicate_Ignore_Server_Ids consists of a space-delimited list of one or more numbers, the first value indicating the number of servers to be ignored; if not 0 (the default), this server-count value is followed by the actual server IDs. For example, if a CHANGE MASTER TO statement containing the IGNORE_SERVER_IDS = (2,6,9) option has been issued to tell a slave to ignore masters having the server ID 2, 6, or 9, that information appears as shown here:

```
 Replicate_Ignore_Server_Ids: 3 2 6 9
```

`Replicate_Ignore_Server_Ids` filtering is performed by the I/O thread, rather than by the SQL thread, which means that events which are filtered out are not written to the relay log. This differs from the filtering actions taken by server options such `--replicate-do-table`, which apply to the SQL thread.

- `Master_Server_Id`

  The `server_id` value from the master.

- `Master_UUID`

  The `server_uuid` [2162] value from the master.

- `Master_Info_File`

  The location of the `master.info` file.

- `SQL_Delay`

  The number of seconds that the slave must lag the master.

- `SQL_Remaining_Delay`

  When `Slave_SQL_Running_State` is `Waiting until MASTER_DELAY seconds after master executed event`, this field contains the number of delay seconds remaining. At other times, this field is `NULL`.

- `Slave_SQL_Running_State`

  The state of the SQL thread (analogous to `Slave_IO_State`). The value is identical to the `State` value of the SQL thread as displayed by `SHOW PROCESSLIST`. Section 8.12.5.6, “Replication Slave SQL Thread States”, provides a listing of possible states

- `Master_Retry_Count`

  The number of times the slave can attempt to reconnect to the master in the event of a lost connection. This value can be set using the `MASTER_RETRY_COUNT` option of the `CHANGE MASTER TO` statement (preferred) or the older `--master-retry-count` server option (still supported for backward compatibility).

- `Master_Bind`

  The network interface that the slave is bound to, if any. This is set using the `MASTER_BIND` option for the `CHANGE MASTER TO` statement.

- `Last_IO_Error_Timestamp`

  A timestamp in `YYMMDD HH:MM:SS` format that shows when the most recent I/O error took place.

- `Last_SQL_Error_Timestamp`

  A timestamp in `YYMMDD HH:MM:SS` format that shows when the last SQL error occurred.

- `Retrieved_Gtid_Set`

  The set of global transaction IDs corresponding to all transactions received by this slave. Empty if GTIDs are not in use.

This is the set of all GTIDs that exist or have existed in the relay logs. Each GTID is added as soon as the `Gtid_log_event` is received. This can cause partially transmitted transactions to have their GTIDs included in the set.

When all relay logs are lost due to executing `RESET SLAVE` or `CHANGE MASTER TO`, or due to the effects of the `--relay-log-recovery` option, the set is cleared. When `relay_log_purge = 1`, the newest relay log is always kept, and the set is not cleared.

Prior to MySQL 5.7.1, this value was printed using uppercase. In MySQL 5.7.1 and later, it is always printed using lowercase. (Bug #15869441)

- `Executed_Gtid_Set`

  The set of global transaction IDs written in the binary log. This is the same as the master's value for the global `gtid_executed` system variable, as well as the master's value for `Executed_Gtid_Set` in the output of `SHOW MASTER STATUS`. Empty if GTIDs are not in use.

  Prior to MySQL 5.7.1, this value was printed using uppercase. In MySQL 5.7.1 and later, it is always printed using lowercase. (Bug #15869441)

- `Auto_Position`

  1 if autopositioning is in use; otherwise 0.

  This column was added in MySQL 5.7.1. (Bug #15992220)

13.7.5.34 `SHOW STATUS` Syntax

```
SHOW [GLOBAL | SESSION] STATUS
    [LIKE 'pattern' | WHERE expr]
```

`SHOW STATUS` provides server status information. This information also can be obtained using the `mysqladmin extended-status` command. The `LIKE` clause, if present, indicates which variable names to match. The `WHERE` clause can be given to select rows using more general conditions, as discussed in Section 19.31, “Extensions to `SHOW` Statements”. This statement does not require any privilege. It requires only the ability to connect to the server.

Partial output is shown here. The list of names and values may be different for your server. The meaning of each variable is given in Section 5.1.6, “Server Status Variables”.

```
mysql> SHOW STATUS;
+--------------------------+------------+
| Variable_name            | Value      |
+--------------------------+------------+
| Aborted_clients          | 0          |
| Aborted_connects         | 0          |
| Bytes_received           | 155372598  |
| Bytes_sent               | 1176560426 |
| Connections              | 30023      |
| Created_tmp_disk_tables  | 0          |
| Created_tmp_tables       | 8340       |
| Created_tmp_files        | 60         |
...
| Open_tables              | 1          |
| Open_files               | 2          |
| Open_streams             | 0          |
| Opened_tables            | 44600      |
| Questions                 | 2026873    |
...
| Table_locks_immediate     | 1920382    |
| Table_locks_waited        | 0          |
| Threads_cached            | 0          |
| Threads_created           | 30022      |
| Threads_connected         | 1          |
| Threads_running           | 1          |
| Uptime                    | 80380      |
+--------------------------+------------+
```

With a LIKE clause, the statement displays only rows for those variables with names that match the pattern:

```
mysql> SHOW STATUS LIKE 'Key%';
+--------------------+----------+
| Variable_name      | Value    |
+--------------------+----------+
| Key_blocks_used    | 14955    |
| Key_read_requests  | 96854827 |
| Key_reads          | 162040   |
| Key_write_requests | 7589728  |
| Key_writes         | 3813196  |
+--------------------+----------+
```

With the GLOBAL modifier, SHOW STATUS displays the status values for all connections to MySQL. With SESSION, it displays the status values for the current connection. If no modifier is present, the default is SESSION. LOCAL is a synonym for SESSION.

Some status variables have only a global value. For these, you get the same value for both GLOBAL and SESSION. The scope for each status variable is listed at Section 5.1.6, “Server Status Variables”.

Each invocation of the SHOW STATUS statement uses an internal temporary table and increments the global Created_tmp_tables value.

13.7.5.35 SHOW TABLE STATUS Syntax

```
SHOW TABLE STATUS [{FROM | IN} db_name]
    [LIKE 'pattern' | WHERE expr]
```

SHOW TABLE STATUS works likes SHOW TABLES, but provides a lot of information about each non-TEMPORARY table. You can also get this list using the mysqlshow --status *db_name* command. The LIKE clause, if present, indicates which table names to match. The WHERE clause can be given to select rows using more general conditions, as discussed in Section 19.31, “Extensions to SHOW Statements”.

This statement also displays information about views.

SHOW TABLE STATUS output has the following columns:

- Name

  The name of the table.

- Engine

  The storage engine for the table. See Chapter 14, *Storage Engines*.

- Version

The version number of the table's .frm file.

- Row_format

  The row-storage format (Fixed, Dynamic, Compressed, Redundant, Compact). For MyISAM tables, (Dynamic corresponds to what myisamchk -dvv reports as Packed. The format of InnoDB tables is reported as Redundant or Compact. For the Barracuda file format of the InnoDB Plugin, the format may be Compressed or Dynamic.

- Rows

  The number of rows. Some storage engines, such as MyISAM, store the exact count. For other storage engines, such as InnoDB, this value is an approximation, and may vary from the actual value by as much as 40 to 50%. In such cases, use SELECT COUNT(*) to obtain an accurate count.

  The Rows value is NULL for tables in the INFORMATION_SCHEMA database.

- Avg_row_length

  The average row length.

- Data_length

  The length of the data file.

- Max_data_length

  The maximum length of the data file. This is the total number of bytes of data that can be stored in the table, given the data pointer size used.

- Index_length

  The length of the index file.

- Data_free

  The number of allocated but unused bytes.

  This information is also shown for InnoDB tables (previously, it was in the Comment value). InnoDB tables report the free space of the tablespace to which the table belongs. For a table located in the shared tablespace, this is the free space of the shared tablespace. If you are using multiple tablespaces and the table has its own tablespace, the free space is for only that table. Free space means the number of completely free 1MB extents minus a safety margin. Even if free space displays as 0, it may be possible to insert rows as long as new extents need not be allocated.

  For partitioned tables, this value is only an estimate and may not be absolutely correct. A more accurate method of obtaining this information in such cases is to query the INFORMATION_SCHEMA.PARTITIONS table, as shown in this example:

```
SELECT    SUM(DATA_FREE)
    FROM  INFORMATION_SCHEMA.PARTITIONS
    WHERE TABLE_SCHEMA = 'mydb'
    AND   TABLE_NAME   = 'mytable';
```

  For more information, see Section 19.14, “The INFORMATION_SCHEMA PARTITIONS Table”.

- Auto_increment

The next AUTO_INCREMENT value.

- Create_time

  When the table was created.

- Update_time

  When the data file was last updated. For some storage engines, this value is NULL. For example, InnoDB stores multiple tables in its system tablespace and the data file timestamp does not apply. Even with file-per-table mode with each InnoDB table in a separate .ibd file, change buffering can delay the write to the data file, so the file modification time is different from the time of the last insert, update, or delete. For MyISAM, the data file timestamp is used; however, on Windows the timestamp is not updated by updates so the value is inaccurate.

- Check_time

  When the table was last checked. Not all storage engines update this time, in which case the value is always NULL.

- Collation

  The table's character set and collation.

- Checksum

  The live checksum value (if any).

- Create_options

  Extra options used with CREATE TABLE. The original options supplied when CREATE TABLE is called are retained and the options reported here may differ from the active table settings and options.

- Comment

  The comment used when creating the table (or information as to why MySQL could not access the table information).

For MEMORY tables, the Data_length, Max_data_length, and Index_length values approximate the actual amount of allocated memory. The allocation algorithm reserves memory in large amounts to reduce the number of allocation operations.

For views, all the fields displayed by SHOW TABLE STATUS are NULL except that Name indicates the view name and Comment says view.

13.7.5.36 SHOW TABLES Syntax

```
SHOW [FULL] TABLES [{FROM | IN} db_name]
    [LIKE 'pattern' | WHERE expr]
```

SHOW TABLES lists the non-TEMPORARY tables in a given database. You can also get this list using the mysqlshow *db_name* command. The LIKE clause, if present, indicates which table names to match. The WHERE clause can be given to select rows using more general conditions, as discussed in Section 19.31, “Extensions to SHOW Statements”.

Matching performed by the LIKE clause is dependent on the setting of the lower_case_table_names system variable.

This statement also lists any views in the database. The FULL modifier is supported such that SHOW FULL TABLES displays a second output column. Values for the second column are BASE TABLE for a table and VIEW for a view.

If you have no privileges for a base table or view, it does not show up in the output from SHOW TABLES or mysqlshow db_name.

13.7.5.37 SHOW TRIGGERS Syntax

```
SHOW TRIGGERS [{FROM | IN} db_name]
    [LIKE 'pattern' | WHERE expr]
```

SHOW TRIGGERS lists the triggers currently defined for tables in a database (the default database unless a FROM clause is given). This statement returns results only for databases and tables for which you have the TRIGGER privilege. The LIKE clause, if present, indicates which table names to match (not trigger names) and causes the statement to display triggers for those tables. The WHERE clause can be given to select rows using more general conditions, as discussed in Section 19.31, "Extensions to SHOW Statements".

For the trigger ins_sum as defined in Section 18.3, "Using Triggers", the output of this statement is as shown here:

```
mysql> SHOW TRIGGERS LIKE 'acc%'\G
*************************** 1. row ***************************
             Trigger: ins_sum
               Event: INSERT
               Table: account
           Statement: SET @sum = @sum + NEW.amount
              Timing: BEFORE
             Created: 2013-07-09 10:39:34.96
            sql_mode: NO_ENGINE_SUBSTITUTION
             Definer: me@localhost
character_set_client: utf8
collation_connection: utf8_general_ci
  Database Collation: latin1_swedish_ci
```

SHOW TRIGGERS output has the following columns:

- Trigger: The trigger name.
- Event: The type of operation that causes trigger activation. The value is 'INSERT', 'UPDATE', or 'DELETE'.
- Table: The table for which the trigger is defined.
- Statement: The trigger body; that is, the statement executed when the trigger activates.
- Timing: Whether the trigger activates before or after the triggering event. The value is 'BEFORE' or 'AFTER'.
- Created: The date and time when the trigger was created. This is a TIMESTAMP(2) value (with a fractional part in hundredths of seconds) for triggers created in MySQL 5.7.2 or later, NULL for triggers created prior to 5.7.2.
- sql_mode: The SQL mode in effect when the trigger executes.
- Definer: The account of the user who created the trigger, in '*user_name*'@'*host_name*' format.
- character_set_client: The session value of the character_set_client system variable when the trigger was created.

- collation_connection: The session value of the collation_connection system variable when the trigger was created.
- Database Collation: The collation of the database with which the trigger is associated.

You can also obtain information about trigger objects from INFORMATION_SCHEMA, which contains a TRIGGERS table. See Section 19.27, "The INFORMATION_SCHEMA TRIGGERS Table".

13.7.5.38 SHOW VARIABLES Syntax

```
SHOW [GLOBAL | SESSION] VARIABLES
    [LIKE 'pattern' | WHERE expr]
```

SHOW VARIABLES shows the values of MySQL system variables. This information also can be obtained using the mysqladmin variables command. The LIKE clause, if present, indicates which variable names to match. The WHERE clause can be given to select rows using more general conditions, as discussed in Section 19.31, "Extensions to SHOW Statements". This statement does not require any privilege. It requires only the ability to connect to the server.

With the GLOBAL modifier, SHOW VARIABLES displays the values that are used for new connections to MySQL. In MySQL 5.7, if a variable has no global value, no value is displayed. With SESSION, SHOW VARIABLES displays the values that are in effect for the current connection. If no modifier is present, the default is SESSION. LOCAL is a synonym for SESSION.

SHOW VARIABLES is subject to a version-dependent display-width limit. For variables with very long values that are not completely displayed, use SELECT as a workaround. For example:

```
SELECT @@GLOBAL.innodb_data_file_path;
```

If the default system variable values are unsuitable, you can set them using command options when mysqld starts, and most can be changed at runtime with the SET statement. See Section 5.1.5, "Using System Variables", and Section 13.7.4, "SET Syntax".

Partial output is shown here. The list of names and values may be different for your server. Section 5.1.4, "Server System Variables", describes the meaning of each variable, and Section 8.11.2, "Tuning Server Parameters", provides information about tuning them.

```
mysql> SHOW VARIABLES;
+-----------------------------------------+---------------------------+
| Variable_name                           | Value                     |
+-----------------------------------------+---------------------------+
| auto_increment_increment                | 1                         |
| auto_increment_offset                   | 1                         |
| autocommit                              | ON                        |
| automatic_sp_privileges                 | ON                        |
| back_log                                | 50                        |
| basedir                                 | /home/jon/bin/mysql-5.5   |
| big_tables                              | OFF                       |
| binlog_cache_size                       | 32768                     |
| binlog_direct_non_transactional_updates | OFF                       |
| binlog_format                           | STATEMENT                 |
| binlog_stmt_cache_size                  | 32768                     |
| bulk_insert_buffer_size                 | 8388608                   |
...
| max_allowed_packet                      | 1048576                   |
| max_binlog_cache_size                   | 18446744073709547520      |
| max_binlog_size                         | 1073741824                |
| max_binlog_stmt_cache_size              | 18446744073709547520      |
| max_connect_errors                      | 10                        |
| max_connections                         | 151                       |
| max_delayed_threads                     | 20                        |
| max_error_count                         | 64                        |
| max_heap_table_size                     | 16777216                  |
| max_insert_delayed_threads              | 20                        |
| max_join_size                           | 18446744073709551615      |
...

| thread_handling                         | one-thread-per-connection |
| thread_stack                            | 262144                    |
| time_format                             | %H:%i:%s                  |
| time_zone                               | SYSTEM                    |
| timed_mutexes                           | OFF                       |
| timestamp                               | 1316689732                |
| tmp_table_size                          | 16777216                  |
| tmpdir                                  | /tmp                      |
| transaction_alloc_block_size            | 8192                      |
| transaction_prealloc_size               | 4096                      |
| tx_isolation                            | REPEATABLE-READ           |
| unique_checks                           | ON                        |
| updatable_views_with_limit              | YES                       |
| version                                 | 5.5.17-log                |
| version_comment                         | Source distribution       |
| version_compile_machine                 | x86_64                    |
| version_compile_os                      | Linux                     |
| wait_timeout                            | 28800                     |
| warning_count                           | 0                         |
+-----------------------------------------+---------------------------+
```

With a LIKE clause, the statement displays only rows for those variables with names that match the pattern. To obtain the row for a specific variable, use a LIKE clause as shown:

```
SHOW VARIABLES LIKE 'max_join_size';
SHOW SESSION VARIABLES LIKE 'max_join_size';
```

To get a list of variables whose name match a pattern, use the “%” wildcard character in a LIKE clause:

```
SHOW VARIABLES LIKE '%size%';
SHOW GLOBAL VARIABLES LIKE '%size%';
```

Wildcard characters can be used in any position within the pattern to be matched. Strictly speaking, because “_” is a wildcard that matches any single character, you should escape it as “_” to match it literally. In practice, this is rarely necessary.

13.7.5.39 SHOW WARNINGS Syntax

```
SHOW WARNINGS [LIMIT [offset,] row_count]
SHOW COUNT(*) WARNINGS
```

SHOW WARNINGS is a diagnostic statement that displays information about the conditions (errors, warnings, and notes) resulting from executing a statement in the current session. Warnings are generated for DML statements such as INSERT, UPDATE, and LOAD DATA INFILE as well as DDL statements such as CREATE TABLE and ALTER TABLE.

The LIMIT clause has the same syntax as for the SELECT statement. See Section 13.2.9, “SELECT Syntax”.

SHOW WARNINGS is also used following EXPLAIN EXTENDED, to display the extra information generated by EXPLAIN when the EXTENDED keyword is used. See Section 8.8.4, “EXPLAIN EXTENDED Output Format”.

As of MySQL 5.7.2, `SHOW WARNINGS` displays information about the conditions resulting from execution of the most recent nondiagnostic statement in the current session. If the most recent statement resulted in an error during parsing, `SHOW WARNINGS` shows the resulting conditions, regardless of statement type (diagnostic or nondiagnostic).

Before MySQL 5.7.2, `SHOW WARNINGS` displays information about the conditions resulting from the most recent statement in the current session that generated messages. It shows nothing if the most recent statement used a table and generated no messages. (That is, statements that use a table but generate no messages clear the message list.) Statements that do not use tables and do not generate messages have no effect on the message list.

The `SHOW COUNT(*) WARNINGS` diagnostic statement displays the total number of errors, warnings, and notes. You can also retrieve this number from the `warning_count` system variable:

```
SHOW COUNT(*) WARNINGS;
SELECT @@warning_count;
```

A difference in these statements is that the first is a diagnostic statement that does not clear the message list. The second, because it is a `SELECT` statement is considered nondiagnostic and, as of MySQL 5.7.2, does clear the message list.

A related diagnostic statement, `SHOW ERRORS`, shows only error conditions (it excludes warnings and notes), and `SHOW COUNT(*) ERRORS` statement displays the total number of errors. See Section 13.7.5.16, “`SHOW ERRORS` Syntax”. `GET DIAGNOSTICS` can be used to examine information for individual conditions. See Section 13.6.7.3, “`GET DIAGNOSTICS` Syntax”.

Here is a simple example that shows data-conversion warnings for `INSERT`:

```
mysql> CREATE TABLE t1 (a TINYINT NOT NULL, b CHAR(4));
Query OK, 0 rows affected (0.05 sec)

mysql> INSERT INTO t1 VALUES(10,'mysql'), (NULL,'test'), (300,'xyz');
Query OK, 3 rows affected, 3 warnings (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 3

mysql> SHOW WARNINGS\G
*************************** 1. row ***************************
  Level: Warning
   Code: 1265
Message: Data truncated for column 'b' at row 1
*************************** 2. row ***************************
  Level: Warning
   Code: 1048
Message: Column 'a' cannot be null
*************************** 3. row ***************************
  Level: Warning
   Code: 1264
Message: Out of range value for column 'a' at row 3
3 rows in set (0.00 sec)
```

The `max_error_count` system variable controls the maximum number of error, warning, and note messages for which the server stores information, and thus the number of messages that `SHOW WARNINGS` displays. To change the number of messages the server can store, change the value of `max_error_count`. The default is 64.

`max_error_count` controls only how many messages are stored, not how many are counted. The value of `warning_count` is not limited by `max_error_count`, even if the number of messages generated exceeds `max_error_count`. The following example demonstrates this. The `ALTER TABLE` statement produces three warning messages (strict SQL mode is disabled for the example to prevent an error from occuring after a single conversion issue). Only one message is stored and displayed because max_error_count has been set to 1, but all three are counted (as shown by the value of warning_count):

```
mysql> SHOW VARIABLES LIKE 'max_error_count';
+-----------------+-------+
| Variable_name   | Value |
+-----------------+-------+
| max_error_count | 64    |
+-----------------+-------+
1 row in set (0.00 sec)

mysql> SET max_error_count=1, sql_mode = '';
Query OK, 0 rows affected (0.00 sec)

mysql> ALTER TABLE t1 MODIFY b CHAR;
Query OK, 3 rows affected, 3 warnings (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 3

mysql> SHOW WARNINGS;
+---------+------+----------------------------------------+
| Level   | Code | Message                                |
+---------+------+----------------------------------------+
| Warning | 1263 | Data truncated for column 'b' at row 1 |
+---------+------+----------------------------------------+
1 row in set (0.00 sec)

mysql> SELECT @@warning_count;
+-----------------+
| @@warning_count |
+-----------------+
|               3 |
+-----------------+
1 row in set (0.01 sec)
```

To disable message storage, set max_error_count to 0. In this case, warning_count still indicates how many warnings occurred, but messages are not stored and cannot be displayed.

The sql_notes system variable controls whether note messages increment warning_count and whether the server stores them. By default, sql_notes is 1, but if set to 0, notes do not increment warning_count and the server does not store them:

```
mysql> SET sql_notes = 1;
mysql> DROP TABLE IF EXISTS test.no_such_table;
Query OK, 0 rows affected, 1 warning (0.00 sec)
mysql> SHOW WARNINGS;
+-------+------+------------------------------------+
| Level | Code | Message                            |
+-------+------+------------------------------------+
| Note  | 1051 | Unknown table 'test.no_such_table' |
+-------+------+------------------------------------+
1 row in set (0.00 sec)

mysql> SET sql_notes = 0;
mysql> DROP TABLE IF EXISTS test.no_such_table;
Query OK, 0 rows affected (0.00 sec)
mysql> SHOW WARNINGS;
Empty set (0.00 sec)
```

The MySQL server sends to each client a count indicating the total number of errors, warnings, and notes resulting from the most recent statement executed by that client. From the C API, this value can be obtained by calling mysql_warning_count(). See Section 21.8.7.77, “mysql_warning_count()”.

13.7.6 Other Administrative Statements

13.7.6.1 BINLOG Syntax

```
BINLOG 'str'
```

BINLOG is an internal-use statement. It is generated by the mysqlbinlog program as the printable representation of certain events in binary log files. (See Section 4.6.7, “mysqlbinlog — Utility for Processing Binary Log Files”.) The 'str' value is a base 64-encoded string the that server decodes to determine the data change indicated by the corresponding event. This statement requires the SUPER privilege.

As of MySQL 5.6, this statement can execute only format description events and row events. Previously it could execute all types of events.

13.7.6.2 CACHE INDEX Syntax

```
CACHE INDEX
  tbl_index_list [, tbl_index_list] ...
  [PARTITION (partition_list | ALL)]
  IN key_cache_name

tbl_index_list:
  tbl_name [[INDEX|KEY] (index_name[, index_name] ...)]

partition_list:
  partition_name[, partition_name][, ...]
```

The CACHE INDEX statement assigns table indexes to a specific key cache. It is used only for MyISAM tables. After the indexes have been assigned, they can be preloaded into the cache if desired with LOAD INDEX INTO CACHE.

The following statement assigns indexes from the tables t1, t2, and t3 to the key cache named hot_cache:

```
mysql> CACHE INDEX t1, t2, t3 IN hot_cache;
+---------+--------------------+----------+----------+
| Table   | Op                 | Msg_type | Msg_text |
+---------+--------------------+----------+----------+
| test.t1 | assign_to_keycache | status   | OK       |
| test.t2 | assign_to_keycache | status   | OK       |
| test.t3 | assign_to_keycache | status   | OK       |
+---------+--------------------+----------+----------+
```

The syntax of CACHE INDEX enables you to specify that only particular indexes from a table should be assigned to the cache. The current implementation assigns all the table's indexes to the cache, so there is no reason to specify anything other than the table name.

The key cache referred to in a CACHE INDEX statement can be created by setting its size with a parameter setting statement or in the server parameter settings. For example:

```
mysql> SET GLOBAL keycache1.key_buffer_size=128*1024;
```

Key cache parameters can be accessed as members of a structured system variable. See Section 5.1.5.1, “Structured System Variables”.

A key cache must exist before you can assign indexes to it:

```
mysql> CACHE INDEX t1 IN non_existent_cache;
ERROR 1284 (HY000): Unknown key cache 'non_existent_cache'
```

By default, table indexes are assigned to the main (default) key cache created at the server startup. When a key cache is destroyed, all indexes assigned to it become assigned to the default key cache again.

Index assignment affects the server globally: If one client assigns an index to a given cache, this cache is used for all queries involving the index, no matter which client issues the queries.

In MySQL 5.7, this statement is also supported for partitioned `MyISAM` tables. You can assign one or more indexes for one, several, or all partitions to a given key cache. For example, you can do the following:

```
CREATE TABLE pt (c1 INT, c2 VARCHAR(50), INDEX i(c1))
    PARTITION BY HASH(c1)
    PARTITIONS 4;

SET GLOBAL kc_fast.key_buffer_size = 128 * 1024;
SET GLOBAL kc_slow.key_buffer_size = 128 * 1024;

CACHE INDEX pt PARTITION (p0) IN kc_fast;
CACHE INDEX pt PARTITION (p1, p3) IN kc_slow;
```

The previous set of statements performs the following actions:

- Creates a partitioned table with 4 partitions; these partitions are automatically named `p0`, ..., `p3`; this table has an index named `i` on column `c1`.
- Creates 2 key caches named `kc_fast` and `kc_slow`
- Assigns the index for partition `p0` to the `kc_fast` key cache and the index for partitions `p1` and `p3` to the `kc_slow` key cache; the index for the remaining partition (`p2`) uses the server's default key cache.

If you wish instead to assign the indexes for all partitions in table `pt` to a single key cache named `kc_all`, you can use either one of the following 2 statements:

```
CACHE INDEX pt PARTITION (ALL) IN kc_all;

CACHE INDEX pt IN kc_all;
```

The two statements just shown are equivalent, and issuing either one of them has exactly the same effect. In other words, if you wish to assign indexes for all partitions of a partitioned table to the same key cache, then the `PARTITION (ALL)` clause is optional.

When assigning indexes for multiple partitions to a key cache, the partitions do not have to be contiguous, and you are not required to list their names in any particular order. Indexes for any partitions that are not explicitly assigned to a key cache automatically use the server's default key cache.

In MySQL 5.7, index preloading is also supported for partitioned `MyISAM` tables. For more information, see Section 13.7.6.5, “`LOAD INDEX INTO CACHE` Syntax”.

In MySQL 5.7.1, `gtid_next` must be set to `AUTOMATIC` before issuing this statement. This restriction does not apply in MySQL 5.7.2 or later. (Bug #16062608, Bug #16715809, Bug #69045)

13.7.6.3 `FLUSH` Syntax

```
FLUSH [NO_WRITE_TO_BINLOG | LOCAL]
    flush_option [, flush_option] ...
```

The FLUSH statement has several variant forms that clear or reload various internal caches, flush tables, or acquire locks. To execute FLUSH, you must have the RELOAD privilege. Specific flush options might require additional privileges, as described later.

By default, the server writes FLUSH statements to the binary log so that they replicate to replication slaves. To suppress logging, specify the optional NO_WRITE_TO_BINLOG keyword or its alias LOCAL.

Note

FLUSH LOGS, FLUSH TABLES WITH READ LOCK (with or without a table list), and FLUSH TABLES *tbl_name* ... FOR EXPORT are not written to the binary log in any case because they would cause problems if replicated to a slave.

Sending a SIGHUP signal to the server causes several flush operations to occur that are similar to various forms of the FLUSH statement. See Section 5.1.11, “Server Response to Signals”.

The FLUSH statement causes an implicit commit. See Section 13.3.3, “Statements That Cause an Implicit Commit”.

The RESET statement is similar to FLUSH. See Section 13.7.6.6, “RESET Syntax”, for information about using the RESET statement with replication.

flush_option can be any of the following items.

- DES_KEY_FILE

  Reloads the DES keys from the file that was specified with the --des-key-file option at server startup time.

- HOSTS

  Empties the host cache. You should flush the host cache if some of your hosts change IP address or if the error message Host '*host_name*' is blocked occurs. (See Section C.5.2.6, “Host '*host_name*' is blocked”.) When more than max_connect_errors errors occur successively for a given host while connecting to the MySQL server, MySQL assumes that something is wrong and blocks the host from further connection requests. Flushing the host cache enables further connection attempts from the host. The default value of max_connect_errors is 10. To avoid this error message, start the server with max_connect_errors set to a large value.

- [*log_type*] LOGS

  With no *log_type* option, FLUSH LOGS closes and reopens all log files. If binary logging is enabled, the sequence number of the binary log file is incremented by one relative to the previous file.

  With a *log_type* option, only the specified log type is flushed. These *log_type* options are permitted:

  - BINARY closes and reopens the binary log files.
  - ENGINE closes and reopens any flushable logs for installed storage engines. Currently, this causes InnoDB to flush its logs to disk.
  - ERROR closes and reopens the error log file.
  - GENERAL closes and reopens the general query log file.
  - RELAY closes and reopens the relay log files.
  - SLOW closes and reopens the slow query log file.

- PRIVILEGES

  Reloads the privileges from the grant tables in the mysql database.

  The server caches information in memory as a result of GRANT, CREATE USER, CREATE SERVER, and INSTALL PLUGIN statements. This memory is not released by the corresponding REVOKE, DROP USER, DROP SERVER, and UNINSTALL PLUGIN statements, so for a server that executes many instances of the statements that cause caching, there will be an increase in memory use. This cached memory can be freed with FLUSH PRIVILEGES.

- QUERY CACHE

  Defragment the query cache to better utilize its memory. FLUSH QUERY CACHE does not remove any queries from the cache, unlike FLUSH TABLES or RESET QUERY CACHE.

- STATUS

  This option adds the current thread's session status variable values to the global values and resets the session values to zero. Some global variables may be reset to zero as well. It also resets the counters for key caches (default and named) to zero and sets Max_used_connections to the current number of open connections. This is something you should use only when debugging a query. See Section 1.7, “How to Report Bugs or Problems”.

- TABLES

  FLUSH TABLES flushes tables, and, depending on the variant used, acquires locks. The permitted syntax is discussed later in this section.

- USER_RESOURCES

  Resets all per-hour user resources to zero. This enables clients that have reached their hourly connection, query, or update limits to resume activity immediately. FLUSH USER_RESOURCES does not apply to the limit on maximum simultaneous connections. See Section 6.3.4, “Setting Account Resource Limits”.

The mysqladmin utility provides a command-line interface to some flush operations, using commands such as flush-hosts, flush-logs, flush-privileges, flush-status, and flush-tables. See Section 4.5.2, “mysqladmin — Client for Administering a MySQL Server”.

> **Note**
>
> It is not possible to issue FLUSH statements within stored functions or triggers. However, you may use FLUSH in stored procedures, so long as these are not called from stored functions or triggers. See Section E.1, “Restrictions on Stored Programs”.

In MySQL 5.7.1, gtid_next must be set to AUTOMATIC before issuing this statement. This restriction does not apply in MySQL 5.7.2 or later. (Bug #16062608, Bug #16715809, Bug #69045)

FLUSH TABLES Syntax

FLUSH TABLES has several forms, described following. If any variant of the TABLES option is used in a FLUSH statement, it must be the only option used. FLUSH TABLE is a synonym for FLUSH TABLES.

- FLUSH TABLES

  Closes all open tables, forces all tables in use to be closed, and flushes the query cache. FLUSH TABLES also removes all query results from the query cache, like the RESET QUERY CACHE statement.

In MySQL 5.7, `FLUSH TABLES` is not permitted when there is an active `LOCK TABLES ... READ`. To flush and lock tables, use `FLUSH TABLES tbl_name ... WITH READ LOCK` instead.

- `FLUSH TABLES tbl_name [, tbl_name] ...`

  With a list of one or more comma-separated table names, this statement is like `FLUSH TABLES` with no names except that the server flushes only the named tables. No error occurs if a named table does not exist.

- `FLUSH TABLES WITH READ LOCK`

  Closes all open tables and locks all tables for all databases with a global read lock. This is a very convenient way to get backups if you have a file system such as Veritas or ZFS that can take snapshots in time. Use `UNLOCK TABLES` to release the lock.

  `FLUSH TABLES WITH READ LOCK` acquires a global read lock and not table locks, so it is not subject to the same behavior as `LOCK TABLES` and `UNLOCK TABLES` with respect to table locking and implicit commits:

  - `UNLOCK TABLES` implicitly commits any active transaction only if any tables currently have been locked with `LOCK TABLES`. The commit does not occur for `UNLOCK TABLES` following `FLUSH TABLES WITH READ LOCK` because the latter statement does not acquire table locks.

  - Beginning a transaction causes table locks acquired with `LOCK TABLES` to be released, as though you had executed `UNLOCK TABLES`. Beginning a transaction does not release a global read lock acquired with `FLUSH TABLES WITH READ LOCK`.

  `FLUSH TABLES WITH READ LOCK` does not prevent the server from inserting rows into the log tables (see Section 5.2.1, “Selecting General Query and Slow Query Log Output Destinations”).

- `FLUSH TABLES tbl_name [, tbl_name] ... WITH READ LOCK`

  This statement flushes and acquires read locks for the named tables. The statement first acquires exclusive metadata locks for the tables, so it waits for transactions that have those tables open to complete. Then the statement flushes the tables from the table cache, reopens the tables, acquires table locks (like `LOCK TABLES ... READ`), and downgrades the metadata locks from exclusive to shared. After the statement acquires locks and downgrades the metadata locks, other sessions can read but not modify the tables.

  Because this statement acquires table locks, you must have the `LOCK TABLES` privilege for each table, in addition to the `RELOAD` privilege that is required to use any `FLUSH` statement.

  This statement applies only to existing base tables. If a name refers to a base table, that table is used. If it refers to a `TEMPORARY` table, it is ignored. If a name applies to a view, an `ER_WRONG_OBJECT` error occurs. Otherwise, an `ER_NO_SUCH_TABLE` error occurs.

  Use `UNLOCK TABLES` to release the locks, `LOCK TABLES` to release the locks and acquire other locks, or `START TRANSACTION` to release the locks and begin a new transaction.

  This variant of `FLUSH` enables tables to be flushed and locked in a single operation. It provides a workaround for the restriction in MySQL 5.7 that `FLUSH TABLES` is not permitted when there is an active `LOCK TABLES ... READ`.

  This statement does not perform an implicit `UNLOCK TABLES`, so an error results if you use the statement while there is any active `LOCK TABLES` or use it a second time without first releasing the locks acquired.

If a flushed table was opened with HANDLER, the handler is implicitly flushed and loses its position.

- FLUSH TABLES *tbl_name* [, *tbl_name*] ... FOR EXPORT

  This FLUSH TABLES variant applies to InnoDB tables. It ensures that changes to the named tables have been flushed to disk so that binary table copies can be made while the server is running.

  The statement works like this:

  1. It acquires shared metadata locks for the named tables. The statement blocks as long as other sessions have active transactions that have modified those tables or hold table locks for them. When the locks have been acquired, the statement blocks transactions that attempt to update the tables while permitting read-only operations to continue.

  2. It checks whether all storage engines for the tables support FOR EXPORT. If any do not, an ER_ILLEGAL_HA error occurs and the statement fails.

  3. The statement notifies the storage engine for each table to make the table ready for export. The storage engine must ensure that any pending changes are written to disk.

  4. The statement puts the session in lock-tables mode so that the metadata locks acquired earlier are not released when the FOR EXPORT statement completes.

  The FLUSH TABLES ... FOR EXPORT statement requires that you have the SELECT privilege for each table. Because this statement acquires table locks, you must also have the LOCK TABLES privilege for each table, in addition to the RELOAD privilege that is required to use any FLUSH statement.

  This statement applies only to existing base tables. If a name refers to a base table, that table is used. If it refers to a TEMPORARY table, it is ignored. If a name applies to a view, an ER_WRONG_OBJECT error occurs. Otherwise, an ER_NO_SUCH_TABLE error occurs.

  InnoDB supports FOR EXPORT for tables that have their own .ibd file file (that is, tables that were created with the innodb_file_per_table setting enabled). InnoDB ensures when notified by the FOR EXPORT statement that any changes have been flushed to disk. This permits a binary copy of table contents to be made while the FOR EXPORT statement is in effect because the .ibd file is transaction consistent and can be copied while the server is running. FOR EXPORT does not apply to InnoDB system tablespace files, or to InnoDB tables that have any FULLTEXT indexes.

  FLUSH TABLES ...FOR EXPORT does not work with partitioned InnoDB tables prior to MySQL 5.7.4, but is supported for such tables in MySQL 5.7.4 and later. (Bug #16943907)

  When notified by FOR EXPORT, InnoDB writes to disk certain kinds of data that is normally held in memory or in separate disk buffers outside the tablespace files. For each table, InnoDB also produces a file named *table_name*.cfg in the same database directory as the table. The .cfg file contains metadata needed to reimport the tablespace files later, into the same or different server.

  When the FOR EXPORT statement completes, InnoDB will have flushed all dirty pages to the table data files. Any change buffer entries are merged prior to flushing. At this point, the tables are locked and quiescent: The tables are in a transactionally consistent state on disk and you can copy the .ibd tablespace files along with the corresponding .cfg files to get a consistent snapshot of those tables.

  For the procedure to reimport the copied table data into a MySQL instance, see Section 14.2.5.5, “Copying Tablespaces to Another Server (Transportable Tablespaces)”.

After you are done with the tables, use `UNLOCK TABLES` to release the locks, `LOCK TABLES` to release the locks and acquire other locks, or `START TRANSACTION` to release the locks and begin a new transaction.

While any of these statements is in effect within the session, attempts to use `FLUSH TABLES ... FOR EXPORT` produce an error:

```
FLUSH TABLES ... WITH READ LOCK
FLUSH TABLES ... FOR EXPORT
LOCK TABLES ... READ
LOCK TABLES ... WRITE
```

While `FLUSH TABLES ... FOR EXPORT` is in effect within the session, attempts to use any of these statements produce an error:

```
FLUSH TABLES WITH READ LOCK
FLUSH TABLES ... WITH READ LOCK
FLUSH TABLES ... FOR EXPORT
```

13.7.6.4 `KILL` Syntax

```
KILL [CONNECTION | QUERY] thread_id
```

Each connection to `mysqld` runs in a separate thread. You can see which threads are running with the `SHOW PROCESSLIST` statement and kill a thread with the `KILL thread_id` statement.

`KILL` permits an optional `CONNECTION` or `QUERY` modifier:

- `KILL CONNECTION` is the same as `KILL` with no modifier: It terminates the connection associated with the given `thread_id`.

- `KILL QUERY` terminates the statement that the connection is currently executing, but leaves the connection itself intact.

If you have the `PROCESS` privilege, you can see all threads. If you have the `SUPER` privilege, you can kill all threads and statements. Otherwise, you can see and kill only your own threads and statements.

You can also use the `mysqladmin processlist` and `mysqladmin kill` commands to examine and kill threads.

> **Note**
>
> You cannot use `KILL` with the Embedded MySQL Server library because the embedded server merely runs inside the threads of the host application. It does not create any connection threads of its own.

When you use `KILL`, a thread-specific kill flag is set for the thread. In most cases, it might take some time for the thread to die because the kill flag is checked only at specific intervals:

- In `SELECT`, `ORDER BY` and `GROUP BY` loops, the flag is checked after reading a block of rows. If the kill flag is set, the statement is aborted.

- During `ALTER TABLE`, the kill flag is checked before each block of rows are read from the original table. If the kill flag was set, the statement is aborted and the temporary table is deleted.

- During UPDATE or DELETE operations, the kill flag is checked after each block read and after each updated or deleted row. If the kill flag is set, the statement is aborted. Note that if you are not using transactions, the changes are not rolled back.
- GET_LOCK() aborts and returns NULL.
- If the thread is in the table lock handler (state: Locked), the table lock is quickly aborted.
- If the thread is waiting for free disk space in a write call, the write is aborted with a “disk full” error message.
- **Warning**

  Killing a REPAIR TABLE or OPTIMIZE TABLE operation on a MyISAM table results in a table that is corrupted and unusable. Any reads or writes to such a table fail until you optimize or repair it again (without interruption).

13.7.6.5 LOAD INDEX INTO CACHE Syntax

```
LOAD INDEX INTO CACHE
  tbl_index_list [, tbl_index_list] ...

tbl_index_list:
  tbl_name
    [PARTITION (partition_list | ALL)]
    [[INDEX|KEY] (index_name[, index_name] ...)]
    [IGNORE LEAVES]

partition_list:
    partition_name[, partition_name][, ...]
```

The LOAD INDEX INTO CACHE statement preloads a table index into the key cache to which it has been assigned by an explicit CACHE INDEX statement, or into the default key cache otherwise.

LOAD INDEX INTO CACHE is used only for MyISAM tables. In MySQL 5.7, it is also supported for partitioned MyISAM tables; in addition, indexes on partitioned tables can be preloaded for one, several, or all partitions.

The IGNORE LEAVES modifier causes only blocks for the nonleaf nodes of the index to be preloaded.

IGNORE LEAVES is also supported for partitioned MyISAM tables.

The following statement preloads nodes (index blocks) of indexes for the tables t1 and t2:

```
mysql> LOAD INDEX INTO CACHE t1, t2 IGNORE LEAVES;
+---------+--------------+----------+----------+
| Table   | Op           | Msg_type | Msg_text |
+---------+--------------+----------+----------+
| test.t1 | preload_keys | status   | OK       |
| test.t2 | preload_keys | status   | OK       |
+---------+--------------+----------+----------+
```

This statement preloads all index blocks from t1. It preloads only blocks for the nonleaf nodes from t2.

The syntax of LOAD INDEX INTO CACHE enables you to specify that only particular indexes from a table should be preloaded. The current implementation preloads all the table's indexes into the cache, so there is no reason to specify anything other than the table name.

In MySQL 5.7.1, gtid_next must be set to AUTOMATIC before issuing this statement. This restriction does not apply in MySQL 5.7.2 or later. (Bug #16062608, Bug #16715809, Bug #69045)

In MySQL 5.7, it is possible to preload indexes on specific partitions of partitioned MyISAM tables. For example, of the following 2 statements, the first preloads indexes for partition p0 of a partitioned table pt, while the second preloads the indexes for partitions p1 and p3 of the same table:

```
LOAD INDEX INTO CACHE pt PARTITION (p0);
LOAD INDEX INTO CACHE pt PARTITION (p1, p3);
```

To preload the indexes for all partitions in table pt, you can use either one of the following 2 statements:

```
LOAD INDEX INTO CACHE pt PARTITION (ALL);

LOAD INDEX INTO CACHE pt;
```

The two statements just shown are equivalent, and issuing either one of them has exactly the same effect. In other words, if you wish to preload indexes for all partitions of a partitioned table, then the PARTITION (ALL) clause is optional.

When preloading indexes for multiple partitions, the partitions do not have to be contiguous, and you are not required to list their names in any particular order.

LOAD INDEX INTO CACHE ... IGNORE LEAVES fails unless all indexes in a table have the same block size. You can determine index block sizes for a table by using myisamchk -dv and checking the Blocksize column.

13.7.6.6 RESET Syntax

```
RESET reset_option [, reset_option] ...
```

The RESET statement is used to clear the state of various server operations. You must have the RELOAD privilege to execute RESET.

RESET acts as a stronger version of the FLUSH statement. See Section 13.7.6.3, “FLUSH Syntax”.

The RESET statement causes an implicit commit. See Section 13.3.3, “Statements That Cause an Implicit Commit”.

In MySQL 5.7.1, gtid_next must be set to AUTOMATIC before issuing this statement. This restriction does not apply in MySQL 5.7.2 or later. (Bug #16062608, Bug #16715809, Bug #69045)

reset_option can be any of the following:

- MASTER

  Deletes all binary logs listed in the index file, resets the binary log index file to be empty, and creates a new binary log file.

- QUERY CACHE

  Removes all query results from the query cache.

- SLAVE

  Makes the slave forget its replication position in the master binary logs. Also resets the relay log by deleting any existing relay log files and beginning a new one.

13.8 MySQL Utility Statements

13.8.1 DESCRIBE Syntax

The DESCRIBE and EXPLAIN statements are synonyms, used either to obtain information about table structure or query execution plans. For more information, see Section 13.7.5.5, "SHOW COLUMNS Syntax", and Section 13.8.2, "EXPLAIN Syntax".

13.8.2 EXPLAIN Syntax

```
{EXPLAIN | DESCRIBE | DESC}
    tbl_name [col_name | wild]

{EXPLAIN | DESCRIBE | DESC}
    [explain_type]
    {explainable_stmt | FOR CONNECTION connection_id}

explain_type: {
    EXTENDED
  | PARTITIONS
  | FORMAT = format_name
}

format_name: {
    TRADITIONAL
  | JSON
}

explainable_stmt: {
    SELECT statement
  | DELETE statement
  | INSERT statement
  | REPLACE statement
  | UPDATE statement
}
```

The DESCRIBE and EXPLAIN statements are synonyms. In practice, the DESCRIBE keyword is more often used to obtain information about table structure, whereas EXPLAIN is used to obtain a query execution plan (that is, an explanation of how MySQL would execute a query). The following discussion uses the DESCRIBE and EXPLAIN keywords in accordance with those uses, but the MySQL parser treats them as completely synonymous.

Obtaining Table Structure Information

DESCRIBE provides information about the columns in a table:

```
mysql> DESCRIBE City;
+------------+----------+------+-----+---------+----------------+
| Field      | Type     | Null | Key | Default | Extra          |
+------------+----------+------+-----+---------+----------------+
| Id         | int(11)  | NO   | PRI | NULL    | auto_increment |
| Name       | char(35) | NO   |     |         |                |
| Country    | char(3)  | NO   | UNI |         |                |
| District   | char(20) | YES  | MUL |         |                |
| Population | int(11)  | NO   |     | 0       |                |
+------------+----------+------+-----+---------+----------------+
```

DESCRIBE is a shortcut for SHOW COLUMNS. These statements also display information for views. The description for SHOW COLUMNS provides more information about the output columns. See Section 13.7.5.5, "SHOW COLUMNS Syntax".

By default, DESCRIBE displays information about all columns in the table. *col_name*, if given, is the name of a column in the table. In this case, the statement displays information only for the named column. *wild*, if given, is a pattern string. It can contain the SQL "%" and "_" wildcard characters. In this case, the statement displays output only for the columns with names matching the string. There is no need to enclose the string within quotation marks unless it contains spaces or other special characters.

The DESCRIBE statement is provided for compatibility with Oracle.

The SHOW CREATE TABLE, SHOW TABLE STATUS, and SHOW INDEX statements also provide information about tables. See Section 13.7.5, "SHOW Syntax".

Obtaining Execution Plan Information

The EXPLAIN statement provides information about how MySQL executes statements:

- In MySQL 5.7, permitted explainable statements for EXPLAIN are SELECT, DELETE, INSERT, REPLACE, and UPDATE.

- When EXPLAIN is used with an explainable statement, MySQL displays information from the optimizer about the statement execution plan. That is, MySQL explains how it would process the statement, including information about how tables are joined and in which order. For information about using EXPLAIN to obtain execution plan information, see Section 8.8.2, "EXPLAIN Output Format".

- When EXPLAIN is used with FOR CONNECTION *connection_id* rather than an explainable statement, it displays the execution plan for the statement executing in the named connection. See Section 8.8.3, "Obtaining Execution Plan Information for a Named Connection".

- EXPLAIN EXTENDED can be used to obtain additional execution plan information. See Section 8.8.4, "EXPLAIN EXTENDED Output Format".

  As of MySQL 5.7.3, the EXPLAIN statement is changed so that the effect of the EXTENDED keyword is always enabled. EXTENDED is still recognized for backward compatibility, but is superfluous and is deprecated; its use results in a warning. It will be removed from EXPLAIN syntax in a future MySQL release.

- EXPLAIN PARTITIONS is useful for examining queries involving partitioned tables. See Section 17.3.5, "Obtaining Information About Partitions".

  As of MySQL 5.7.3, the EXPLAIN statement is changed so that the effect of the PARTITIONS keyword is always enabled. PARTITIONS is still recognized for backward compatibility, but is superfluous and is deprecated; its use results in a warning. It will be removed from EXPLAIN syntax in a future MySQL release.

- The FORMAT option can be used to select the output format. TRADITIONAL presents the output in tabular format. This is the default if no FORMAT option is present. JSON format displays the information in JSON format. With FORMAT = JSON, the output includes extended and partition information.

With the help of EXPLAIN, you can see where you should add indexes to tables so that the statement executes faster by using indexes to find rows. You can also use EXPLAIN to check whether the optimizer joins the tables in an optimal order. To give a hint to the optimizer to use a join order corresponding to the order in which the tables are named in a SELECT statement, begin the statement with SELECT STRAIGHT_JOIN rather than just SELECT. (See Section 13.2.9, "SELECT Syntax".)

The optimizer trace may sometimes provide information complementary to that of EXPLAIN. However, the optimizer trace format and content are subject to change between versions. For details, see MySQL Internals: Tracing the Optimizer.

If you have a problem with indexes not being used when you believe that they should be, run ANALYZE TABLE to update table statistics, such as cardinality of keys, that can affect the choices the optimizer makes. See Section 13.7.2.1, “ANALYZE TABLE Syntax”.

13.8.3 HELP Syntax

```
HELP 'search_string'
```

The HELP statement returns online information from the MySQL Reference manual. Its proper operation requires that the help tables in the mysql database be initialized with help topic information (see Section 5.1.10, “Server-Side Help”).

The HELP statement searches the help tables for the given search string and displays the result of the search. The search string is not case sensitive.

The search string can contain the the wildcard characters “%” and “_”. These have the same meaning as for pattern-matching operations performed with the LIKE operator. For example, HELP 'rep%' returns a list of topics that begin with rep.

The HELP statement understands several types of search strings:

- At the most general level, use contents to retrieve a list of the top-level help categories:

  ```
  HELP 'contents'
  ```

- For a list of topics in a given help category, such as Data Types, use the category name:

  ```
  HELP 'data types'
  ```

- For help on a specific help topic, such as the ASCII() function or the CREATE TABLE statement, use the associated keyword or keywords:

  ```
  HELP 'ascii'
  HELP 'create table'
  ```

In other words, the search string matches a category, many topics, or a single topic. You cannot necessarily tell in advance whether a given search string will return a list of items or the help information for a single help topic. However, you can tell what kind of response HELP returned by examining the number of rows and columns in the result set.

The following descriptions indicate the forms that the result set can take. Output for the example statements is shown using the familiar “tabular” or “vertical” format that you see when using the mysql client, but note that mysql itself reformats HELP result sets in a different way.

- Empty result set

  No match could be found for the search string.

- Result set containing a single row with three columns

  This means that the search string yielded a hit for the help topic. The result has three columns:

  - name: The topic name.
  - description: Descriptive help text for the topic.
  - example: Usage example or examples. This column might be blank.

Example: HELP 'replace'

Yields:

```
name: REPLACE
description: Syntax:
REPLACE(str,from_str,to_str)

Returns the string str with all occurrences of the string from_str
replaced by the string to_str. REPLACE() performs a case-sensitive
match when searching for from_str.
example: mysql> SELECT REPLACE('www.mysql.com', 'w', 'Ww');
        -> 'WwWwWw.mysql.com'
```

- Result set containing multiple rows with two columns

  This means that the search string matched many help topics. The result set indicates the help topic names:

  - name: The help topic name.
  - is_it_category: Y if the name represents a help category, N if it does not. If it does not, the name value when specified as the argument to the HELP statement should yield a single-row result set containing a description for the named item.

  Example: HELP 'status'

  Yields:

```
+-----------------------+----------------+
| name                  | is_it_category |
+-----------------------+----------------+
| SHOW                  | N              |
| SHOW ENGINE           | N              |
| SHOW MASTER STATUS    | N              |
| SHOW PROCEDURE STATUS | N              |
| SHOW SLAVE STATUS     | N              |
| SHOW STATUS           | N              |
| SHOW TABLE STATUS     | N              |
+-----------------------+----------------+
```

- Result set containing multiple rows with three columns

  This means the search string matches a category. The result set contains category entries:

  - source_category_name: The help category name.
  - name: The category or topic name
  - is_it_category: Y if the name represents a help category, N if it does not. If it does not, the name value when specified as the argument to the HELP statement should yield a single-row result set containing a description for the named item.

  Example: HELP 'functions'

  Yields:

```
+----------------------+-------------------------+----------------+
| source_category_name | name                   | is_it_category |
+----------------------+------------------------+----------------+
| Functions            | CREATE FUNCTION        | N              |
| Functions            | DROP FUNCTION          | N              |
| Functions            | Bit Functions          | Y              |
| Functions            | Comparison operators   | Y              |
| Functions            | Control flow functions | Y              |
| Functions            | Date and Time Functions | Y              |
| Functions            | Encryption Functions   | Y              |
| Functions            | Information Functions  | Y              |
| Functions            | Logical operators      | Y              |
| Functions            | Miscellaneous Functions | Y              |
| Functions            | Numeric Functions      | Y              |
| Functions            | String Functions       | Y              |
+----------------------+------------------------+----------------+
```

13.8.4 USE Syntax

```
USE db_name
```

The USE *db_name* statement tells MySQL to use the *db_name* database as the default (current) database for subsequent statements. The database remains the default until the end of the session or another USE statement is issued:

```
USE db1;
SELECT COUNT(*) FROM mytable;   # selects from db1.mytable
USE db2;
SELECT COUNT(*) FROM mytable;   # selects from db2.mytable
```

Making a particular database the default by means of the USE statement does not preclude you from accessing tables in other databases. The following example accesses the author table from the db1 database and the editor table from the db2 database:

```
USE db1;
SELECT author_name,editor_name FROM author,db2.editor
  WHERE author.editor_id = db2.editor.editor_id;
```

Chapter 14 Storage Engines

Table of Contents

Storage engines are MySQL components that handle the SQL operations for different table types. InnoDB is the most general-purpose storage engine, and Oracle recommends using it for tables except for specialized use cases. (The CREATE TABLE statement in MySQL 5.7 creates InnoDB tables by default.)

MySQL Server uses a pluggable storage engine architecture that enables storage engines to be loaded into and unloaded from a running MySQL server.

To determine which storage engines your server supports, use the SHOW ENGINES statement. The value in the Support column indicates whether an engine can be used. A value of YES, NO, or DEFAULT indicates that an engine is available, not available, or available and currently set as the default storage engine.

```
mysql> SHOW ENGINES\G
*************************** 1. row ***************************
      Engine: PERFORMANCE_SCHEMA
     Support: YES
     Comment: Performance Schema
Transactions: NO
          XA: NO
  Savepoints: NO
*************************** 2. row ***************************
      Engine: InnoDB
     Support: DEFAULT
     Comment: Supports transactions, row-level locking, and foreign keys
Transactions: YES
          XA: YES
  Savepoints: YES
*************************** 3. row ***************************
      Engine: MRG_MYISAM
     Support: YES
     Comment: Collection of identical MyISAM tables
Transactions: NO
          XA: NO
  Savepoints: NO
*************************** 4. row ***************************
      Engine: BLACKHOLE
     Support: YES
     Comment: /dev/null storage engine (anything you write to it disappears)
Transactions: NO
          XA: NO
  Savepoints: NO
*************************** 5. row ***************************
      Engine: MyISAM
     Support: YES
     Comment: MyISAM storage engine
Transactions: NO
          XA: NO
  Savepoints: NO
...
```

This chapter primarily describes the features and performance characteristics of InnoDB tables. It also covers the use cases for the special-purpose MySQL storage engines, except for NDBCLUSTER which is covered in MySQL Cluster NDB 7.2. For advanced users, it also contains a description of the pluggable storage engine architecture (see Section 14.12, “Overview of MySQL Storage Engine Architecture”).

For information about storage engine support offered in commercial MySQL Server binaries, see *MySQL Enterprise Server 5.6*, on the MySQL Web site. The storage engines available might depend on which edition of Enterprise Server you are using.

For answers to some commonly asked questions about MySQL storage engines, see Section B.2, “MySQL 5.7 FAQ: Storage Engines”.

MySQL 5.7 Supported storage Engines

- InnoDB: A transaction-safe (ACID compliant) storage engine for MySQL that has commit, rollback, and crash-recovery capabilities to protect user data. InnoDB row-level locking (without escalation to coarser granularity locks) and Oracle-style consistent nonlocking reads increase multi-user concurrency and performance. InnoDB stores user data in clustered indexes to reduce I/O for common queries based on primary keys. To maintain data integrity, InnoDB also supports FOREIGN KEY referential-integrity constraints. InnoDB is the default storage engine in MySQL 5.7.

- MyISAM: These tables have a small footprint. Table-level locking limits the performance in read/write workloads, so it is often used in read-only or read-mostly workloads in Web and data warehousing configurations.

- Memory: Stores all data in RAM, for fast access in environments that require quick lookups of non-critical data. This engine was formerly known as the HEAP engine. Its use cases are decreasing; InnoDB with its buffer pool memory area provides a general-purpose and durable way to keep most or all data in memory, and NDBCLUSTER provides fast key-value lookups for huge distributed data sets.

- CSV: Its tables are really text files with comma-separated values. CSV tables let you import or dump data in CSV format, to exchange data with scripts and applications that read and write that same format. Because CSV tables are not indexed, you typically keep the data in InnoDB tables during normal operation, and only use CSV tables during the import or export stage.

- Archive: These compact, unindexed tables are intended for storing and retrieving large amounts of seldom-referenced historical, archived, or security audit information.

- Blackhole: The Blackhole storage engine accepts but does not store data, similar to the Unix /dev/null device. Queries always return an empty set. These tables can be used in replication configurations where DML statements are sent to slave servers, but the master server does not keep its own copy of the data.

- Merge: Enables a MySQL DBA or developer to logically group a series of identical MyISAM tables and reference them as one object. Good for VLDB environments such as data warehousing.

- Federated: Offers the ability to link separate MySQL servers to create one logical database from many physical servers. Very good for distributed or data mart environments.

- Example: This engine serves as an example in the MySQL source code that illustrates how to begin writing new storage engines. It is primarily of interest to developers. The storage engine is a “stub” that does nothing. You can create tables with this engine, but no data can be stored in them or retrieved from them.

You are not restricted to using the same storage engine for an entire server or schema. You can specify the storage engine for any table. For example, an application might use mostly InnoDB tables, with one CSV table for exporting data to a spreadsheet and a few MEMORY tables for temporary workspaces.

Choosing a Storage Engine

The various storage engines provided with MySQL are designed with different use cases in mind. The following table provides an overview of some storage engines provided with MySQL:

Table 14.1 Storage Engines Feature Summary

Feature	MyISAM	Memory	InnoDB	Archive	NDB
Storage limits	256TB	RAM	64TB	None	384EB
Transactions	No	No	Yes	No	Yes
Locking granularity	Table	Table	Row	Table	Row
MVCC	No	No	Yes	No	No

Feature	MyISAM	Memory	InnoDB	Archive	NDB
Geospatial data type support	Yes	No	Yes	Yes	Yes
Geospatial indexing support	Yes	No	No	No	No
B-tree indexes	Yes	Yes	Yes	No	No
T-tree indexes	No	No	No	No	Yes
Hash indexes	No	Yes	No[a]	No	Yes
Full-text search indexes	Yes	No	Yes[b]	No	No
Clustered indexes	No	No	Yes	No	No
Data caches	No	N/A	Yes	No	Yes
Index caches	Yes	N/A	Yes	No	Yes
Compressed data	Yes[c]	No	Yes[d]	Yes	No
Encrypted data[e]	Yes	Yes	Yes	Yes	Yes
Cluster database support	No	No	No	No	Yes
Replication support[f]	Yes	Yes	Yes	Yes	Yes
Foreign key support	No	No	Yes	No	No
Backup / point-in-time recovery[g]	Yes	Yes	Yes	Yes	Yes
Query cache support	Yes	Yes	Yes	Yes	Yes
Update statistics for data dictionary	Yes	Yes	Yes	Yes	Yes

[a] InnoDB utilizes hash indexes internally for its Adaptive Hash Index feature.

[b] InnoDB support for FULLTEXT indexes is available in MySQL 5.6.4 and higher.

[c] Compressed MyISAM tables are supported only when using the compressed row format. Tables using the compressed row format with MyISAM are read only.

[d] Compressed InnoDB tables require the InnoDB Barracuda file format.

[e] Implemented in the server (via encryption functions), rather than in the storage engine.

[f] Implemented in the server, rather than in the storage engine.

[g] Implemented in the server, rather than in the storage engine.

14.1 Setting the Storage Engine

When you create a new table, you can specify which storage engine to use by adding an `ENGINE` table option to the `CREATE TABLE` statement:

```
-- ENGINE=INNODB not needed unless you have set a different
-- default storage engine.
CREATE TABLE t1 (i INT) ENGINE = INNODB;
-- Simple table definitions can be switched from one to another.
CREATE TABLE t2 (i INT) ENGINE = CSV;
CREATE TABLE t3 (i INT) ENGINE = MEMORY;
```

When you omit the `ENGINE` option, the default storage engine is used. The default engine is `InnoDB` in MySQL 5.7. You can specify the default engine by using the `--default-storage-engine` server startup option, or by setting the `default-storage-engine` option in the `my.cnf` configuration file.

You can set the default storage engine for the current session by setting the `default_storage_engine` variable:

```
SET default_storage_engine=NDBCLUSTER;
```

The storage engine for `TEMPORARY` tables created with `CREATE TEMPORARY TABLE` can be set separately from the engine for permanent tables by setting the `default_tmp_storage_engine`, either at startup or at runtime.

When MySQL is installed on Windows using the MySQL Configuration Wizard, the `InnoDB` or `MyISAM` storage engine can be selected as the default. See The Database Usage Dialog.

To convert a table from one storage engine to another, use an `ALTER TABLE` statement that indicates the new engine:

```
ALTER TABLE t ENGINE = InnoDB;
```

See Section 13.1.14, “`CREATE TABLE` Syntax”, and Section 13.1.6, “`ALTER TABLE` Syntax”.

If you try to use a storage engine that is not compiled in or that is compiled in but deactivated, MySQL instead creates a table using the default storage engine. For example, in a replication setup, perhaps your master server uses `InnoDB` tables for maximum safety, but the slave servers use other storage engines for speed at the expense of durability or concurrency.

By default, a warning is generated whenever `CREATE TABLE` or `ALTER TABLE` cannot use the default storage engine. To prevent confusing, unintended behavior if the desired engine is unavailable, enable the `NO_ENGINE_SUBSTITUTION` SQL mode. If the desired engine is unavailable, this setting produces an error instead of a warning, and the table is not created or altered. See Section 5.1.7, “Server SQL Modes”.

For new tables, MySQL always creates an `.frm` file to hold the table and column definitions. The table's index and data may be stored in one or more other files, depending on the storage engine. The server creates the `.frm` file above the storage engine level. Individual storage engines create any additional files required for the tables that they manage. If a table name contains special characters, the names for the table files contain encoded versions of those characters as described in Section 9.2.3, “Mapping of Identifiers to File Names”.

14.2 The InnoDB Storage Engine

14.2.1 Introduction to InnoDB

InnoDB is a general-purpose storage engine that balances high reliability and high performance. As of MySQL 5.5, it is the default MySQL storage engine. In MySQL 5.7, issuing the CREATE TABLE statement without an ENGINE= clause creates an InnoDB table.

Key Advantages of InnoDB

Key advantages of InnoDB tables include:

- Its DML operations follow the ACID model, with transactions featuring commit, rollback, and crash-recovery capabilities to protect user data.
- Row-level locking and Oracle-style consistent reads increase multi-user concurrency and performance.
- InnoDB tables arrange your data on disk to optimize queries based on primary keys.
- To maintain data integrity, InnoDB also supports FOREIGN KEY constraints. Inserts, updates, and deletes are all checked to ensure they do not result in inconsistencies across different tables.
- You can freely mix InnoDB tables with tables from other MySQL storage engines, even within the same statement. For example, you can use a join operation to combine data from InnoDB and MEMORY tables in a single query.
- InnoDB has been designed for maximum performance when processing large data volumes. Its CPU efficiency is probably not matched by any other disk-based relational database engine.

InnoDB Storage Engine Features

The InnoDB storage engine maintains its own buffer pool for caching data and indexes in main memory. By default, with the innodb_file_per_table setting enabled, each new InnoDB table and its associated indexes are stored in a separate file. When the innodb_file_per_table option is disabled, InnoDB stores all its tables and indexes in the single system tablespace, which may consist of several files (or raw disk partitions). InnoDB tables can handle large quantities of data, even on operating systems where file size is limited to 2GB.

InnoDB Enhancements and New Features

For information about InnoDB enhancements and new features in MySQL 5.7, refer to:

- The InnoDB enhancements list in Section 1.4, “What Is New in MySQL 5.7”, which provides an overview of the features added in MySQL 5.7.
- The Release Notes, which provide information about changes in each version.

Additional Resources

- For InnoDB-related terms and definitions, see MySQL Glossary.
- A forum dedicated to the InnoDB storage engine is available at http://forums.mysql.com/list.php?22.
- InnoDB is published under the same GNU GPL License Version 2 (of June 1991) as MySQL. For more information on MySQL licensing, see http://www.mysql.com/company/legal/licensing/.

14.2.1.1 InnoDB as the Default MySQL Storage Engine

MySQL has a well-earned reputation for being easy-to-use and delivering performance and scalability. Prior to MySQL 5.5, MyISAM was the default storage engine. In our experience, most users never changed the default settings. In MySQL 5.5 and higher, InnoDB is the default storage engine. Again, we expect most users will not change the default settings. But, because of InnoDB, the default settings deliver the benefits users expect from their RDBMS: ACID Transactions, Referential Integrity, and Crash Recovery. Let's explore how using InnoDB tables improves your life as a MySQL user, DBA, or developer.

Trends in Storage Engine Usage

In the first years of MySQL growth, early web-based applications didn't push the limits of concurrency and availability. In recent years, hard drive and memory capacity and the performance/price ratio have all gone through the roof. Users pushing the performance boundaries of MySQL care a lot about reliability and crash recovery. MySQL databases are big, busy, robust, distributed, and important.

InnoDB addresses these top user priorities. The trend of storage engine usage has shifted in favor of the more scalable InnoDB. Thus MySQL 5.5 was the logical transition release to make InnoDB the default storage engine.

MySQL continues to work on addressing use cases that formerly required MyISAM tables. In MySQL 5.6 and higher:

- InnoDB can perform full-text search using the FULLTEXT index type. See FULLTEXT Indexes for details.

- InnoDB now performs better with read-only or read-mostly workloads. Automatic optimizations apply to InnoDB queries in autocommit mode, and you can explicitly mark transactions as read-only with the syntax START TRANSACTION READ ONLY. See Optimizations for Read-Only Transactions for details.

- Applications distributed on read-only media can now use InnoDB tables. See Section 14.2.3.1, “Configuring InnoDB for Read-Only Operation” for details.

Consequences of InnoDB as Default MySQL Storage Engine

Starting from MySQL 5.5.5, the default storage engine for new tables is InnoDB. This change applies to newly created tables that don't specify a storage engine with a clause such as ENGINE=MyISAM. Given this change of default behavior, MySQL 5.5 might be a logical point to evaluate whether your tables that do use MyISAM could benefit from switching to InnoDB.

The mysql and information_schema databases, that implement some of the MySQL internals, still use MyISAM. In particular, you cannot switch the grant tables to use InnoDB.

Benefits of InnoDB Tables

If you use MyISAM tables but aren't tied to them for technical reasons, you'll find many things more convenient when you use InnoDB tables:

- If your server crashes because of a hardware or software issue, regardless of what was happening in the database at the time, you don't need to do anything special after restarting the database. InnoDB crash recovery automatically finalizes any changes that were committed before the time of the crash, and undoes any changes that were in process but not committed. Just restart and continue where you left off. This process is now much faster than in MySQL 5.1 and earlier.

- The InnoDB buffer pool caches table and index data as the data is accessed. Frequently used data is processed directly from memory. This cache applies to so many types of information, and speeds up processing so much, that dedicated database servers assign up to 80% of their physical memory to the InnoDB buffer pool.

- If you split up related data into different tables, you can set up foreign keys that enforce referential integrity. Update or delete data, and the related data in other tables is updated or deleted automatically. Try to insert data into a secondary table without corresponding data in the primary table, and the bad data gets kicked out automatically.

- If data becomes corrupted on disk or in memory, a checksum mechanism alerts you to the bogus data before you use it.

- When you design your database with appropriate primary key columns for each table, operations involving those columns are automatically optimized. It is very fast to reference the primary key columns in `WHERE` clauses, `ORDER BY` clauses, `GROUP BY` clauses, and join operations.

- Inserts, updates, deletes are optimized by an automatic mechanism called change buffering. `InnoDB` not only allows concurrent read and write access to the same table, it caches changed data to streamline disk I/O.

- Performance benefits are not limited to giant tables with long-running queries. When the same rows are accessed over and over from a table, a feature called the Adaptive Hash Index takes over to make these lookups even faster, as if they came out of a hash table.

Best Practices for `InnoDB` Tables

If you have been using `InnoDB` for a long time, you already know about features like transactions and foreign keys. If not, read about them throughout this chapter. To make a long story short:

- Specify a primary key for every table using the most frequently queried column or columns, or anauto-increment value if there is no obvious primary key.

- Embrace the idea of joins, where data is pulled from multiple tables based on identical ID values from those tables. For fast join performance, define foreign keys on the join columns, and declare those columns with the same data type in each table. The foreign keys also propagate deletes or updates to all affected tables, and prevent insertion of data in a child table if the corresponding IDs are not present in the parent table.

- Turn off autocommit. Committing hundreds of times a second puts a cap on performance (limited by the write speed of your storage device).

- Group sets of related DML operations into transactions, by bracketing them with `START TRANSACTION` and `COMMIT` statements. While you don't want to commit too often, you also don't want to issue huge batches of `INSERT`, `UPDATE`, or `DELETE` statements that run for hours without committing.

- Stop using `LOCK TABLE` statements. `InnoDB` can handle multiple sessions all reading and writing to the same table at once, without sacrificing reliability or high performance. To get exclusive write access to a set of rows, use the `SELECT ... FOR UPDATE` syntax to lock just the rows you intend to update.

- Enable the `innodb_file_per_table` option to put the data and indexes for individual tables into separate files, instead of in a single giant system tablespace. This setting is required to use some of the other features, such as table compression and fast truncation.

- Evaluate whether your data and access patterns benefit from the new `InnoDB` table compression feature (`ROW_FORMAT=COMPRESSED`) on the `CREATE TABLE` statement. You can compress `InnoDB` tables without sacrificing read/write capability.

- Run your server with the option `--sql_mode=NO_ENGINE_SUBSTITUTION` to prevent tables being created with a different storage engine if there is an issue with the one specified in the `ENGINE=` clause of `CREATE TABLE`.

Recent Improvements for InnoDB Tables

- You can compress tables and associated indexes.
- You can create and drop indexes with much less performance or availability impact than before.
- Truncating a table is very fast, and can free up disk space for the operating system to reuse, rather than freeing up space within the system tablespace that only InnoDB could reuse.
- The storage layout for table data is more efficient for BLOBs and long text fields, with the DYNAMIC row format.
- You can monitor the internal workings of the storage engine by querying INFORMATION_SCHEMA tables.
- You can monitor the performance details of the storage engine by querying performance_schema tables.
- There are many performance improvements. In particular, crash recovery, the automatic process that makes all data consistent when the database is restarted, is fast and reliable (much faster than long-time InnoDB users are used to). The bigger the database, the more dramatic the speedup.

  Most new performance features are automatic, or at most require setting a value for a configuration option. For details, see Section 14.2.12.2, “InnoDB Performance and Scalability Enhancements”. For InnoDB-specific tuning techniques you can apply in your application code, see Section 8.5, “Optimizing for InnoDB Tables”. Advanced users can review Section 14.2.13, “InnoDB Startup Options and System Variables”.

Testing and Benchmarking with InnoDB as Default Storage Engine

Even before completing your upgrade from MySQL 5.1 or earlier to MySQL 5.5 or higher, you can preview whether your database server or application works correctly with InnoDB as the default storage engine. To set up InnoDB as the default storage engine with an earlier MySQL release, either specify on the command line --default-storage-engine=InnoDB, or add to your my.cnf file default-storage-engine=innodb in the [mysqld] section, then restart the server.

Since changing the default storage engine only affects new tables as they are created, run all your application installation and setup steps to confirm that everything installs properly. Then exercise all the application features to make sure all the data loading, editing, and querying features work. If a table relies on some MyISAM-specific feature, you'll receive an error; add the ENGINE=MyISAM clause to the CREATE TABLE statement to avoid the error (for example, tables that rely on full-text search must be MyISAM tables rather than InnoDB ones).

If you did not make a deliberate decision about the storage engine, and you just want to preview how certain tables work when they're created under InnoDB, issue the command ALTER TABLE table_name ENGINE=InnoDB; for each table. Or, to run test queries and other statements without disturbing the original table, make a copy like so:

```
CREATE TABLE InnoDB_Table (...) ENGINE=InnoDB AS SELECT * FROM MyISAM_Table;
```

Since there are so many performance enhancements in InnoDB in MySQL 5.5 and higher, to get a true idea of the performance with a full application under a realistic workload, install the latest MySQL server and run benchmarks.

Test the full application lifecycle, from installation, through heavy usage, and server restart. Kill the server process while the database is busy to simulate a power failure, and verify that the data is recovered successfully when you restart the server.

Test any replication configurations, especially if you use different MySQL versions and options on the master and the slaves.

Verifying that **InnoDB** is the Default Storage Engine

To know what the status of InnoDB is, whether you're doing what-if testing with an older MySQL or comprehensive testing with the latest MySQL:

- Issue the command SHOW ENGINES; to see all the different MySQL storage engines. Look for DEFAULT in the InnoDB line.
- If InnoDB is not present at all, you have a mysqld binary that was compiled without InnoDB support and you need to get a different one.
- If InnoDB is present but disabled, go back through your startup options and configuration file and get rid of any skip-innodb option.

14.2.1.2 Checking InnoDB Availability

To determine whether your server supports InnoDB, use the SHOW ENGINES statement. (Now that InnoDB is the default MySQL storage engine, only very specialized environments might not support it.)

14.2.1.3 Turning Off InnoDB

Oracle recommends InnoDB as the preferred storage engine for typical database applications, from single-user wikis and blogs running on a local system, to high-end applications pushing the limits of performance. In MySQL 5.7, InnoDB is the default storage engine for new tables.

If you do not want to use InnoDB tables:

- Start the server with the --innodb=OFF or --skip-innodb option to disable the InnoDB storage engine.
- Because the default storage engine is InnoDB, the server will not start unless you also use --default-storage-engine and --default-tmp-storage-engine to set the default to some other engine for both permanent and TEMPORARY tables.
- To prevent the server from crashing when the InnoDB-related information_schema tables are queried, also disable the plugins associated with those tables. Specify in the [mysqld] section of the MySQL configuration file:

```
loose-innodb-trx=0
loose-innodb-locks=0
loose-innodb-lock-waits=0
loose-innodb-cmp=0
loose-innodb-cmp-per-index=0
loose-innodb-cmp-per-index-reset=0
loose-innodb-cmp-reset=0
loose-innodb-cmpmem=0
loose-innodb-cmpmem-reset=0
loose-innodb-buffer-page=0
loose-innodb-buffer-page-lru=0
loose-innodb-buffer-pool-stats=0
loose-innodb-metrics=0
loose-innodb-ft-default-stopword=0
loose-innodb-ft-inserted=0
loose-innodb-ft-deleted=0
loose-innodb-ft-being-deleted=0
loose-innodb-ft-config=0
loose-innodb-ft-index-cache=0
```

```
loose-innodb-ft-index-table=0
loose-innodb-sys-tables=0
loose-innodb-sys-tablestats=0
loose-innodb-sys-indexes=0
loose-innodb-sys-columns=0
loose-innodb-sys-fields=0
loose-innodb-sys-foreign=0
loose-innodb-sys-foreign-cols=0
```

14.2.2 InnoDB Concepts and Architecture

The information in this section provides background to help you get the most performance and functionality from using InnoDB tables. It is intended for:

- Anyone switching to MySQL from another database system, to explain what things might seem familiar and which might be all-new.

- Anyone moving from MyISAM tables to InnoDB, now that InnoDB is the default MySQL storage engine.

- Anyone considering their application architecture or software stack, to understand the design considerations, performance characteristics, and scalability of InnoDB tables at a detailed level.

In this section, you will learn:

- How InnoDB closely adheres to ACID principles.

- How InnoDB implements transactions, and how the inner workings of transactions compare with other database systems you might be familiar with.

- How InnoDB implements row-level locking to allow queries and DML statements to read and write the same table simultaneously.

- How multi-version concurrency control (MVCC) keeps transactions from viewing or modifying each others' data before the appropriate time.

- The physical layout of InnoDB-related objects on disk, such as tables, indexes, tablespaces, undo logs, and the redo log.

14.2.2.1 MySQL and the ACID Model

The ACID model is a set of database design principles that emphasize aspects of reliability that are important for business data and mission-critical applications. MySQL includes components such as the InnoDB storage engine that adhere closely to the ACID model, so that data is not corrupted and results are not distorted by exceptional conditions such as software crashes and hardware malfunctions. When you rely on ACID-compliant features, you do not need to reinvent the wheel of consistency checking and crash recovery mechanisms. In cases where you have additional software safeguards, ultra-reliable hardware, or an application that can tolerate a small amount of data loss or inconsistency, you can adjust MySQL settings to trade some of the ACID reliability for greater performance or throughput.

The following sections discuss how MySQL features, in particular the InnoDB storage engine, interact with the categories of the ACID model:

- **A**: atomicity.

- **C**: consistency.

- **I:**: isolation.

- **D**: durability.

Atomicity

The **atomicity** aspect of the ACID model mainly involves InnoDB transactions. Related MySQL features include:

- Autocommit setting.
- COMMIT statement.
- ROLLBACK statement.
- Operational data from the INFORMATION_SCHEMA tables.

Consistency

The **consistency** aspect of the ACID model mainly involves internal InnoDB processing to protect data from crashes. Related MySQL features include:

- InnoDB doublewrite buffer.
- InnoDB crash recovery.

Isolation

The **isolation** aspect of the ACID model mainly involves InnoDB transactions, in particular the isolation level that applies to each transaction. Related MySQL features include:

- Autocommit setting.
- SET ISOLATION LEVEL statement.
- The low-level details of InnoDB locking. During performance tuning, you see these details through INFORMATION_SCHEMA tables.

Durability

The **durability** aspect of the ACID model involves MySQL software features interacting with your particular hardware configuration. Because of the many possibilities depending on the capabilities of your CPU, network, and storage devices, this aspect is the most complicated to provide concrete guidelines for. (And those guidelines might take the form of buy “new hardware”.) Related MySQL features include:

- InnoDB doublewrite buffer, turned on and off by the innodb_doublewrite configuration option.
- Configuration option innodb_flush_log_at_trx_commit.
- Configuration option sync_binlog.
- Configuration option innodb_file_per_table.
- Write buffer in a storage device, such as a disk drive, SSD, or RAID array.
- Battery-backed cache in a storage device.
- The operating system used to run MySQL, in particular its support for the fsync() system call.
- Uninterruptible power supply (UPS) protecting the electrical power to all computer servers and storage devices that run MySQL servers and store MySQL data.
- Your backup strategy, such as frequency and types of backups, and backup retention periods.
- For distributed or hosted data applications, the particular characteristics of the data centers where the hardware for the MySQL servers is located, and network connections between the data centers.

14.2.2.2 The InnoDB Transaction Model and Locking

To implement a large-scale, busy, or highly reliable database application, to port substantial code from a different database system, or to push MySQL performance to the limits of the laws of physics, you must understand the notions of transactions and locking as they relate to the InnoDB storage engine.

In the InnoDB transaction model, the goal is to combine the best properties of a multi-versioning database with traditional two-phase locking. InnoDB does locking on the row level and runs queries as nonlocking consistent reads by default, in the style of Oracle. The lock information in InnoDB is stored so space-efficiently that lock escalation is not needed: Typically, several users are permitted to lock every row in InnoDB tables, or any random subset of the rows, without causing InnoDB memory exhaustion.

In InnoDB, all user activity occurs inside a transaction. If autocommit mode is enabled, each SQL statement forms a single transaction on its own. By default, MySQL starts the session for each new connection with autocommit enabled, so MySQL does a commit after each SQL statement if that statement did not return an error. If a statement returns an error, the commit or rollback behavior depends on the error. See Section 14.2.17.4, "InnoDB Error Handling".

A session that has autocommit enabled can perform a multiple-statement transaction by starting it with an explicit START TRANSACTION or BEGIN statement and ending it with a COMMIT or ROLLBACK statement. See Section 13.3.1, "START TRANSACTION, COMMIT, and ROLLBACK Syntax".

If autocommit mode is disabled within a session with SET autocommit = 0, the session always has a transaction open. A COMMIT or ROLLBACK statement ends the current transaction and a new one starts.

A COMMIT means that the changes made in the current transaction are made permanent and become visible to other sessions. A ROLLBACK statement, on the other hand, cancels all modifications made by the current transaction. Both COMMIT and ROLLBACK release all InnoDB locks that were set during the current transaction.

In terms of the SQL:1992 transaction isolation levels, the default InnoDB level is REPEATABLE READ. InnoDB offers all four transaction isolation levels described by the SQL standard: READ UNCOMMITTED, READ COMMITTED, REPEATABLE READ, and SERIALIZABLE.

A user can change the isolation level for a single session or for all subsequent connections with the SET TRANSACTION statement. To set the server's default isolation level for all connections, use the --transaction-isolation option on the command line or in an option file. For detailed information about isolation levels and level-setting syntax, see Section 13.3.6, "SET TRANSACTION Syntax".

In row-level locking, InnoDB normally uses next-key locking. That means that besides index records, InnoDB can also lock the gap preceding an index record to block insertions by other sessions where the indexed values would be inserted in that gap within the tree data structure. A next-key lock refers to a lock that locks an index record and the gap before it. A gap lock refers to a lock that locks only the gap before some index record.

For more information about row-level locking, and the circumstances under which gap locking is disabled, see Section 14.2.2.6, "InnoDB Record, Gap, and Next-Key Locks".

14.2.2.3 InnoDB Lock Modes

InnoDB implements standard row-level locking where there are two types of locks, shared (*S*) locks and exclusive (*X*) locks. For information about record, gap, and next-key lock types, see Section 14.2.2.6, "InnoDB Record, Gap, and Next-Key Locks".

- A shared (*S*) lock permits the transaction that holds the lock to read a row.
- An exclusive (*X*) lock permits the transaction that holds the lock to update or delete a row.

If transaction T1 holds a shared (*S*) lock on row r, then requests from some distinct transaction T2 for a lock on row r are handled as follows:

- A request by T2 for an *S* lock can be granted immediately. As a result, both T1 and T2 hold an *S* lock on r.
- A request by T2 for an *X* lock cannot be granted immediately.

If a transaction T1 holds an exclusive (*X*) lock on row r, a request from some distinct transaction T2 for a lock of either type on r cannot be granted immediately. Instead, transaction T2 has to wait for transaction T1 to release its lock on row r.

Intention Locks

Additionally, InnoDB supports *multiple granularity locking* which permits coexistence of record locks and locks on entire tables. To make locking at multiple granularity levels practical, additional types of locks called intention locks are used. Intention locks are table locks in InnoDB that indicate which type of lock (shared or exclusive) a transaction will require later for a row in that table. There are two types of intention locks used in InnoDB (assume that transaction T has requested a lock of the indicated type on table t):

- Intention shared (*IS*): Transaction T intends to set *S* locks on individual rows in table t.
- Intention exclusive (*IX*): Transaction T intends to set *X* locks on those rows.

For example, SELECT ... LOCK IN SHARE MODE sets an *IS* lock and SELECT ... FOR UPDATE sets an *IX* lock.

The intention locking protocol is as follows:

- Before a transaction can acquire an *S* lock on a row in table t, it must first acquire an *IS* or stronger lock on t.
- Before a transaction can acquire an *X* lock on a row, it must first acquire an *IX* lock on t.

These rules can be conveniently summarized by means of the following *lock type compatibility matrix*.

	X	***IX***	***S***	***IS***
X	Conflict	Conflict	Conflict	Conflict
IX	Conflict	Compatible	Conflict	Compatible
S	Conflict	Conflict	Compatible	Compatible
IS	Conflict	Compatible	Compatible	Compatible

A lock is granted to a requesting transaction if it is compatible with existing locks, but not if it conflicts with existing locks. A transaction waits until the conflicting existing lock is released. If a lock request conflicts with an existing lock and cannot be granted because it would cause deadlock, an error occurs.

Thus, intention locks do not block anything except full table requests (for example, LOCK TABLES ... WRITE). The main purpose of *IX* and *IS* locks is to show that someone is locking a row, or going to lock a row in the table.

Deadlock Example

The following example illustrates how an error can occur when a lock request would cause a deadlock. The example involves two clients, A and B.

First, client A creates a table containing one row, and then begins a transaction. Within the transaction, A obtains an *S* lock on the row by selecting it in share mode:

```
mysql> CREATE TABLE t (i INT) ENGINE = InnoDB;
Query OK, 0 rows affected (1.07 sec)

mysql> INSERT INTO t (i) VALUES(1);
Query OK, 1 row affected (0.09 sec)

mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM t WHERE i = 1 LOCK IN SHARE MODE;
+------+
| i    |
+------+
|    1 |
+------+
1 row in set (0.10 sec)
```

Next, client B begins a transaction and attempts to delete the row from the table:

```
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql> DELETE FROM t WHERE i = 1;
```

The delete operation requires an *X* lock. The lock cannot be granted because it is incompatible with the *S* lock that client A holds, so the request goes on the queue of lock requests for the row and client B blocks.

Finally, client A also attempts to delete the row from the table:

```
mysql> DELETE FROM t WHERE i = 1;
ERROR 1213 (40001): Deadlock found when trying to get lock;
try restarting transaction
```

Deadlock occurs here because client A needs an *X* lock to delete the row. However, that lock request cannot be granted because client B already has a request for an *X* lock and is waiting for client A to release its *S* lock. Nor can the *S* lock held by A be upgraded to an *X* lock because of the prior request by B for an *X* lock. As a result, InnoDB generates an error for one of the clients and releases its locks. The client returns this error:

```
ERROR 1213 (40001): Deadlock found when trying to get lock;
try restarting transaction
```

At that point, the lock request for the other client can be granted and it deletes the row from the table.

Note

If the LATEST DETECTED DEADLOCK section of InnoDB Monitor output includes a message stating, "TOO DEEP OR LONG SEARCH IN THE LOCK TABLE WAITS-FOR GRAPH, WE WILL ROLL BACK FOLLOWING TRANSACTION," this indicates that the number of transactions on the wait-for list has reached a limit of 200, which is defined by LOCK_MAX_DEPTH_IN_DEADLOCK_CHECK. A wait-for list that exceeds 200 transactions is treated as a deadlock and the transaction attempting to check the wait-for list is rolled back.

The same error may also occur if the locking thread must look at more than 1,000,000 locks owned by the transactions on the wait-for list. The limit of 1,000,000 locks is defined by LOCK_MAX_N_STEPS_IN_DEADLOCK_CHECK.

14.2.2.4 Consistent Nonlocking Reads

A consistent read means that InnoDB uses multi-versioning to present to a query a snapshot of the database at a point in time. The query sees the changes made by transactions that committed before that point of time, and no changes made by later or uncommitted transactions. The exception to this rule is that the query sees the changes made by earlier statements within the same transaction. This exception causes the following anomaly: If you update some rows in a table, a SELECT sees the latest version of the updated rows, but it might also see older versions of any rows. If other sessions simultaneously update the same table, the anomaly means that you might see the table in a state that never existed in the database.

If the transaction isolation level is REPEATABLE READ (the default level), all consistent reads within the same transaction read the snapshot established by the first such read in that transaction. You can get a fresher snapshot for your queries by committing the current transaction and after that issuing new queries.

With READ COMMITTED isolation level, each consistent read within a transaction sets and reads its own fresh snapshot.

Consistent read is the default mode in which InnoDB processes SELECT statements in READ COMMITTED and REPEATABLE READ isolation levels. A consistent read does not set any locks on the tables it accesses, and therefore other sessions are free to modify those tables at the same time a consistent read is being performed on the table.

Suppose that you are running in the default REPEATABLE READ isolation level. When you issue a consistent read (that is, an ordinary SELECT statement), InnoDB gives your transaction a timepoint according to which your query sees the database. If another transaction deletes a row and commits after your timepoint was assigned, you do not see the row as having been deleted. Inserts and updates are treated similarly.

> **Note**
>
> The snapshot of the database state applies to SELECT statements within a transaction, not necessarily to DML statements. If you insert or modify some rows and then commit that transaction, a DELETE or UPDATE statement issued from another concurrent REPEATABLE READ transaction could affect those just-committed rows, even though the session could not query them. If a transaction does update or delete rows committed by a different transaction, those changes do become visible to the current transaction. For example, you might encounter a situation like the following:
>
> ```
> SELECT COUNT(c1) FROM t1 WHERE c1 = 'xyz'; -- Returns 0: no rows match.
> DELETE FROM t1 WHERE c1 = 'xyz'; -- Deletes several rows recently committed by other transact
>
> SELECT COUNT(c2) FROM t1 WHERE c2 = 'abc'; -- Returns 0: no rows match.
> UPDATE t1 SET c2 = 'cba' WHERE c2 = 'abc'; -- Affects 10 rows: another txn just committed 10
> SELECT COUNT(c2) FROM t1 WHERE c2 = 'cba'; -- Returns 10: this txn can now see the rows it ju
> ```

You can advance your timepoint by committing your transaction and then doing another SELECT or START TRANSACTION WITH CONSISTENT SNAPSHOT.

This is called *multi-versioned concurrency control*.

In the following example, session A sees the row inserted by B only when B has committed the insert and A has committed as well, so that the timepoint is advanced past the commit of B.

```
             Session A              Session B

           SET autocommit=0;      SET autocommit=0;
time
|          SELECT * FROM t;
|          empty set
```

```
|                                 INSERT INTO t VALUES (1, 2);
|
v          SELECT * FROM t;
           empty set
                                  COMMIT;

           SELECT * FROM t;
           empty set

           COMMIT;

           SELECT * FROM t;
           ---------------------
           |    1    |    2    |
           ---------------------
           1 row in set
```

If you want to see the “freshest” state of the database, use either the READ COMMITTED isolation level or a locking read:

```
SELECT * FROM t LOCK IN SHARE MODE;
```

With READ COMMITTED isolation level, each consistent read within a transaction sets and reads its own fresh snapshot. With LOCK IN SHARE MODE, a locking read occurs instead: A SELECT blocks until the transaction containing the freshest rows ends (see Section 14.2.2.5, “Locking Reads (SELECT ... FOR UPDATE and SELECT ... LOCK IN SHARE MODE)”).

Consistent read does not work over certain DDL statements:

- Consistent read does not work over DROP TABLE, because MySQL cannot use a table that has been dropped and InnoDB destroys the table.

- Consistent read does not work over ALTER TABLE, because that statement makes a temporary copy of the original table and deletes the original table when the temporary copy is built. When you reissue a consistent read within a transaction, rows in the new table are not visible because those rows did not exist when the transaction's snapshot was taken. In this case, the transaction returns an error: ER_TABLE_DEF_CHANGED, “Table definition has changed, please retry transaction”.

The type of read varies for selects in clauses like INSERT INTO ... SELECT, UPDATE ... (SELECT), and CREATE TABLE ... SELECT that do not specify FOR UPDATE or LOCK IN SHARE MODE:

- By default, InnoDB uses stronger locks and the SELECT part acts like READ COMMITTED, where each consistent read, even within the same transaction, sets and reads its own fresh snapshot.

- To use a consistent read in such cases, enable the innodb_locks_unsafe_for_binlog option and set the isolation level of the transaction to READ UNCOMMITTED, READ COMMITTED, or REPEATABLE READ (that is, anything other than SERIALIZABLE). In this case, no locks are set on rows read from the selected table.

14.2.2.5 Locking Reads (SELECT ... FOR UPDATE and SELECT ... LOCK IN SHARE MODE)

If you query data and then insert or update related data within the same transaction, the regular SELECT statement does not give enough protection. Other transactions can update or delete the same rows you just queried. InnoDB supports two types of locking reads that offer extra safety:

- SELECT ... LOCK IN SHARE MODE sets a shared mode lock on any rows that are read. Other sessions can read the rows, but cannot modify them until your transaction commits. If any of these rows were changed by another transaction that has not yet committed, your query waits until that transaction ends and then uses the latest values.

- For index records the search encounters, `SELECT ... FOR UPDATE` locks the rows and any associated index entries, the same as if you issued an `UPDATE` statement for those rows. Other transactions are blocked from updating those rows, from doing `SELECT ... LOCK IN SHARE MODE`, or from reading the data in certain transaction isolation levels. Consistent reads ignore any locks set on the records that exist in the read view. (Old versions of a record cannot be locked; they are reconstructed by applying undo logs on an in-memory copy of the record.)

These clauses are primarily useful when dealing with tree-structured or graph-structured data, either in a single table or split across multiple tables. You traverse edges or tree branches from one place to another, while reserving the right to come back and change any of these “pointer” values.

All locks set by `LOCK IN SHARE MODE` and `FOR UPDATE` queries are released when the transaction is committed or rolled back.

Note

Locking of rows for update using `SELECT FOR UPDATE` only applies when autocommit is disabled (either by beginning transaction with `START TRANSACTION` or by setting `autocommit` to 0. If autocommit is enabled, the rows matching the specification are not locked.

Usage Examples

Suppose that you want to insert a new row into a table `child`, and make sure that the child row has a parent row in table `parent`. Your application code can ensure referential integrity throughout this sequence of operations.

First, use a consistent read to query the table `PARENT` and verify that the parent row exists. Can you safely insert the child row to table `CHILD`? No, because some other session could delete the parent row in the moment between your `SELECT` and your `INSERT`, without you being aware of it.

To avoid this potential issue, perform the `SELECT` using `LOCK IN SHARE MODE`:

```
SELECT * FROM parent WHERE NAME = 'Jones' LOCK IN SHARE MODE;
```

After the `LOCK IN SHARE MODE` query returns the parent `'Jones'`, you can safely add the child record to the `CHILD` table and commit the transaction. Any transaction that tries to read or write to the applicable row in the `PARENT` table waits until you are finished, that is, the data in all tables is in a consistent state.

For another example, consider an integer counter field in a table `CHILD_CODES`, used to assign a unique identifier to each child added to table `CHILD`. Do not use either consistent read or a shared mode read to read the present value of the counter, because two users of the database could see the same value for the counter, and a duplicate-key error occurs if two transactions attempt to add rows with the same identifier to the `CHILD` table.

Here, `LOCK IN SHARE MODE` is not a good solution because if two users read the counter at the same time, at least one of them ends up in deadlock when it attempts to update the counter.

To implement reading and incrementing the counter, first perform a locking read of the counter using `FOR UPDATE`, and then increment the counter. For example:

```
SELECT counter_field FROM child_codes FOR UPDATE;
UPDATE child_codes SET counter_field = counter_field + 1;
```

A `SELECT ... FOR UPDATE` reads the latest available data, setting exclusive locks on each row it reads. Thus, it sets the same locks a searched SQL `UPDATE` would set on the rows.

The preceding description is merely an example of how SELECT ... FOR UPDATE works. In MySQL, the specific task of generating a unique identifier actually can be accomplished using only a single access to the table:

```
UPDATE child_codes SET counter_field = LAST_INSERT_ID(counter_field + 1);
SELECT LAST_INSERT_ID();
```

The SELECT statement merely retrieves the identifier information (specific to the current connection). It does not access any table.

14.2.2.6 InnoDB Record, Gap, and Next-Key Locks

InnoDB has several types of record-level locks including record locks, gap locks, and next-key locks. For information about shared locks, exclusive locks, and intention locks, see Section 14.2.2.3, “InnoDB Lock Modes”.

- Record lock: This is a lock on an index record.
- Gap lock: This is a lock on a gap between index records, or a lock on the gap before the first or after the last index record.
- Next-key lock: This is a combination of a record lock on the index record and a gap lock on the gap before the index record.

Record Locks

Record locks always lock index records, even if a table is defined with no indexes. For such cases, InnoDB creates a hidden clustered index and uses this index for record locking. See Clustered and Secondary Indexes.

Next-key Locks

By default, InnoDB operates in REPEATABLE READ transaction isolation level. In this case, InnoDB uses next-key locks for searches and index scans, which prevents phantom rows (see Section 14.2.2.7, “Avoiding the Phantom Problem Using Next-Key Locking”).

Next-key locking combines index-row locking with gap locking. InnoDB performs row-level locking in such a way that when it searches or scans a table index, it sets shared or exclusive locks on the index records it encounters. Thus, the row-level locks are actually index-record locks. In addition, a next-key lock on an index record also affects the “gap” before that index record. That is, a next-key lock is an index-record lock plus a gap lock on the gap preceding the index record. If one session has a shared or exclusive lock on record R in an index, another session cannot insert a new index record in the gap immediately before R in the index order.

Suppose that an index contains the values 10, 11, 13, and 20. The possible next-key locks for this index cover the following intervals, where (or) denote exclusion of the interval endpoint and [or] denote inclusion of the endpoint:

```
(negative infinity, 10]
(10, 11]
(11, 13]
(13, 20]
(20, positive infinity)
```

For the last interval, the next-key lock locks the gap above the largest value in the index and the “supremum” pseudo-record having a value higher than any value actually in the index. The supremum is not a real index record, so, in effect, this next-key lock locks only the gap following the largest index value.

Gap Locks

The next-key locking example in the previous section shows that a gap might span a single index value, multiple index values, or even be empty.

Gap locking is not needed for statements that lock rows using a unique index to search for a unique row. (This does not include the case that the search condition includes only some columns of a multiple-column unique index; in that case, gap locking does occur.) For example, if the id column has a unique index, the following statement uses only an index-record lock for the row having id value 100 and it does not matter whether other sessions insert rows in the preceding gap:

```
SELECT * FROM child WHERE id = 100;
```

If id is not indexed or has a nonunique index, the statement does lock the preceding gap.

A type of gap lock called an insertion intention gap lock is set by INSERT operations prior to row insertion. This lock signals the intent to insert in such a way that multiple transactions inserting into the same index gap need not wait for each other if they are not inserting at the same position within the gap. Suppose that there are index records with values of 4 and 7. Separate transactions that attempt to insert values of 5 and 6 each lock the gap between 4 and 7 with insert intention locks prior to obtaining the exclusive lock on the inserted row, but do not block each other because the rows are nonconflicting. For more information about intention locks, see Section 14.2.2.3, "InnoDB Lock Modes".

It is also worth noting here that conflicting locks can be held on a gap by different transactions. For example, transaction A can hold a shared gap lock (gap S-lock) on a gap while transaction B holds an exclusive gap lock (gap X-lock) on the same gap. The reason conflicting gap locks are allowed is that if a record is purged from an index, the gap locks held on the record by different transactions must be merged.

Gap locks in InnoDB are "purely inhibitive", which means they only stop other transactions from inserting to the gap. Thus, a gap X-lock has the same effect as a gap S-lock.

Disabling Gap Locking

Gap locking can be disabled explicitly. This occurs if you change the transaction isolation level to READ COMMITTED or enable the innodb_locks_unsafe_for_binlog system variable (which is now deprecated). Under these circumstances, gap locking is disabled for searches and index scans and is used only for foreign-key constraint checking and duplicate-key checking.

There are also other effects of using the READ COMMITTED isolation level or enabling innodb_locks_unsafe_for_binlog: Record locks for nonmatching rows are released after MySQL has evaluated the WHERE condition. For UPDATE statements, InnoDB does a "semi-consistent" read, such that it returns the latest committed version to MySQL so that MySQL can determine whether the row matches the WHERE condition of the UPDATE.

14.2.2.7 Avoiding the Phantom Problem Using Next-Key Locking

The so-called *phantom* problem occurs within a transaction when the same query produces different sets of rows at different times. For example, if a SELECT is executed twice, but returns a row the second time that was not returned the first time, the row is a "phantom" row.

Suppose that there is an index on the id column of the child table and that you want to read and lock all rows from the table having an identifier value larger than 100, with the intention of updating some column in the selected rows later:

```
SELECT * FROM child WHERE id > 100 FOR UPDATE;
```

The query scans the index starting from the first record where id is bigger than 100. Let the table contain rows having id values of 90 and 102. If the locks set on the index records in the scanned range do not lock out inserts made in the gaps (in this case, the gap between 90 and 102), another session can insert a new row into the table with an id of 101. If you were to execute the same SELECT within the same transaction, you would see a new row with an id of 101 (a “phantom”) in the result set returned by the query. If we regard a set of rows as a data item, the new phantom child would violate the isolation principle of transactions that a transaction should be able to run so that the data it has read does not change during the transaction.

To prevent phantoms, InnoDB uses an algorithm called *next-key locking* that combines index-row locking with gap locking. InnoDB performs row-level locking in such a way that when it searches or scans a table index, it sets shared or exclusive locks on the index records it encounters. Thus, the row-level locks are actually index-record locks. In addition, a next-key lock on an index record also affects the “gap” before that index record. That is, a next-key lock is an index-record lock plus a gap lock on the gap preceding the index record. If one session has a shared or exclusive lock on record R in an index, another session cannot insert a new index record in the gap immediately before R in the index order.

When InnoDB scans an index, it can also lock the gap after the last record in the index. Just that happens in the preceding example: To prevent any insert into the table where id would be bigger than 100, the locks set by InnoDB include a lock on the gap following id value 102.

You can use next-key locking to implement a uniqueness check in your application: If you read your data in share mode and do not see a duplicate for a row you are going to insert, then you can safely insert your row and know that the next-key lock set on the successor of your row during the read prevents anyone meanwhile inserting a duplicate for your row. Thus, the next-key locking enables you to “lock” the nonexistence of something in your table.

Gap locking can be disabled as discussed in Section 14.2.2.6, “InnoDB Record, Gap, and Next-Key Locks”. This may cause phantom problems because other sessions can insert new rows into the gaps when gap locking is disabled.

14.2.2.8 Locks Set by Different SQL Statements in InnoDB

A locking read, an UPDATE, or a DELETE generally set record locks on every index record that is scanned in the processing of the SQL statement. It does not matter whether there are WHERE conditions in the statement that would exclude the row. InnoDB does not remember the exact WHERE condition, but only knows which index ranges were scanned. The locks are normally next-key locks that also block inserts into the “gap” immediately before the record. However, gap locking can be disabled explicitly, which causes next-key locking not to be used. For more information, see Section 14.2.2.6, “InnoDB Record, Gap, and Next-Key Locks”. The transaction isolation level also can affect which locks are set; see Section 13.3.6, “SET TRANSACTION Syntax”.

If a secondary index is used in a search and index record locks to be set are exclusive, InnoDB also retrieves the corresponding clustered index records and sets locks on them.

Differences between shared and exclusive locks are described in Section 14.2.2.3, “InnoDB Lock Modes”.

If you have no indexes suitable for your statement and MySQL must scan the entire table to process the statement, every row of the table becomes locked, which in turn blocks all inserts by other users to the table. It is important to create good indexes so that your queries do not unnecessarily scan many rows.

For SELECT ... FOR UPDATE or SELECT ... LOCK IN SHARE MODE, locks are acquired for scanned rows, and expected to be released for rows that do not qualify for inclusion in the result set (for example, if they do not meet the criteria given in the WHERE clause). However, in some cases, rows might not be unlocked immediately because the relationship between a result row and its original source is lost during query execution. For example, in a UNION, scanned (and locked) rows from a table might be inserted into a temporary table before evaluation whether they qualify for the result set. In this circumstance, the relationship of the rows in the temporary table to the rows in the original table is lost and the latter rows are not unlocked until the end of query execution.

InnoDB sets specific types of locks as follows.

- SELECT ... FROM is a consistent read, reading a snapshot of the database and setting no locks unless the transaction isolation level is set to SERIALIZABLE. For SERIALIZABLE level, the search sets shared next-key locks on the index records it encounters.
- SELECT ... FROM ... LOCK IN SHARE MODE sets shared next-key locks on all index records the search encounters.
- For index records the search encounters, SELECT ... FROM ... FOR UPDATE blocks other sessions from doing SELECT ... FROM ... LOCK IN SHARE MODE or from reading in certain transaction isolation levels. Consistent reads will ignore any locks set on the records that exist in the read view.
- UPDATE ... WHERE ... sets an exclusive next-key lock on every record the search encounters.
- DELETE FROM ... WHERE ... sets an exclusive next-key lock on every record the search encounters.
- INSERT sets an exclusive lock on the inserted row. This lock is an index-record lock, not a next-key lock (that is, there is no gap lock) and does not prevent other sessions from inserting into the gap before the inserted row.

  Prior to inserting the row, a type of gap lock called an insertion intention gap lock is set. This lock signals the intent to insert in such a way that multiple transactions inserting into the same index gap need not wait for each other if they are not inserting at the same position within the gap. Suppose that there are index records with values of 4 and 7. Separate transactions that attempt to insert values of 5 and 6 each lock the gap between 4 and 7 with insert intention locks prior to obtaining the exclusive lock on the inserted row, but do not block each other because the rows are nonconflicting.

  If a duplicate-key error occurs, a shared lock on the duplicate index record is set. This use of a shared lock can result in deadlock should there be multiple sessions trying to insert the same row if another session already has an exclusive lock. This can occur if another session deletes the row. Suppose that an InnoDB table t1 has the following structure:

  ```
  CREATE TABLE t1 (i INT, PRIMARY KEY (i)) ENGINE = InnoDB;
  ```

  Now suppose that three sessions perform the following operations in order:

  Session 1:

  ```
  START TRANSACTION;
  INSERT INTO t1 VALUES(1);
  ```

  Session 2:

  ```
  START TRANSACTION;
  INSERT INTO t1 VALUES(1);
  ```

  Session 3:

  ```
  START TRANSACTION;
  ```

```
INSERT INTO t1 VALUES(1);
```

Session 1:

```
ROLLBACK;
```

The first operation by session 1 acquires an exclusive lock for the row. The operations by sessions 2 and 3 both result in a duplicate-key error and they both request a shared lock for the row. When session 1 rolls back, it releases its exclusive lock on the row and the queued shared lock requests for sessions 2 and 3 are granted. At this point, sessions 2 and 3 deadlock: Neither can acquire an exclusive lock for the row because of the shared lock held by the other.

A similar situation occurs if the table already contains a row with key value 1 and three sessions perform the following operations in order:

Session 1:

```
START TRANSACTION;
DELETE FROM t1 WHERE i = 1;
```

Session 2:

```
START TRANSACTION;
INSERT INTO t1 VALUES(1);
```

Session 3:

```
START TRANSACTION;
INSERT INTO t1 VALUES(1);
```

Session 1:

```
COMMIT;
```

The first operation by session 1 acquires an exclusive lock for the row. The operations by sessions 2 and 3 both result in a duplicate-key error and they both request a shared lock for the row. When session 1 commits, it releases its exclusive lock on the row and the queued shared lock requests for sessions 2 and 3 are granted. At this point, sessions 2 and 3 deadlock: Neither can acquire an exclusive lock for the row because of the shared lock held by the other.

- `INSERT ... ON DUPLICATE KEY UPDATE` differs from a simple `INSERT` in that an exclusive next-key lock rather than a shared lock is placed on the row to be updated when a duplicate-key error occurs.
- `REPLACE` is done like an `INSERT` if there is no collision on a unique key. Otherwise, an exclusive next-key lock is placed on the row to be replaced.
- `INSERT INTO T SELECT ... FROM S WHERE ...` sets an exclusive index record without a gap lock on each row inserted into `T`. If the transaction isolation level is `READ COMMITTED` or the transaction isolation level is not `SERIALIZABLE`, `InnoDB` does the search on `S` as a consistent read (no locks). Otherwise, `InnoDB` sets shared next-key locks on rows from `S`. `InnoDB` has to set locks in the latter case: In roll-forward recovery from a backup, every SQL statement must be executed in exactly the same way it was done originally.

  `CREATE TABLE ... SELECT ...` performs the `SELECT` with shared next-key locks or as a consistent read, as for `INSERT ... SELECT`.

When a SELECT is used in the constructs REPLACE INTO t SELECT ... FROM s WHERE ... or UPDATE t ... WHERE col IN (SELECT ... FROM s ...), InnoDB sets shared next-key locks on rows from table s.

- While initializing a previously specified AUTO_INCREMENT column on a table, InnoDB sets an exclusive lock on the end of the index associated with the AUTO_INCREMENT column. In accessing the auto-increment counter, InnoDB uses a specific AUTO-INC table lock mode where the lock lasts only to the end of the current SQL statement, not to the end of the entire transaction. Other sessions cannot insert into the table while the AUTO-INC table lock is held; see Section 14.2.2.2, "The InnoDB Transaction Model and Locking".

  InnoDB fetches the value of a previously initialized AUTO_INCREMENT column without setting any locks.

- If a FOREIGN KEY constraint is defined on a table, any insert, update, or delete that requires the constraint condition to be checked sets shared record-level locks on the records that it looks at to check the constraint. InnoDB also sets these locks in the case where the constraint fails.

- LOCK TABLES sets table locks, but it is the higher MySQL layer above the InnoDB layer that sets these locks. InnoDB is aware of table locks if innodb_table_locks = 1 (the default) and autocommit = 0, and the MySQL layer above InnoDB knows about row-level locks.

  Otherwise, InnoDB's automatic deadlock detection cannot detect deadlocks where such table locks are involved. Also, because in this case the higher MySQL layer does not know about row-level locks, it is possible to get a table lock on a table where another session currently has row-level locks. However, this does not endanger transaction integrity, as discussed in Section 14.2.2.10, "Deadlock Detection and Rollback". See also Section 14.2.6.7, "Limits on InnoDB Tables".

14.2.2.9 Implicit Transaction Commit and Rollback

By default, MySQL starts the session for each new connection with autocommit mode enabled, so MySQL does a commit after each SQL statement if that statement did not return an error. If a statement returns an error, the commit or rollback behavior depends on the error. See Section 14.2.17.4, "InnoDB Error Handling".

If a session that has autocommit disabled ends without explicitly committing the final transaction, MySQL rolls back that transaction.

Some statements implicitly end a transaction, as if you had done a COMMIT before executing the statement. For details, see Section 13.3.3, "Statements That Cause an Implicit Commit".

14.2.2.10 Deadlock Detection and Rollback

InnoDB automatically detects transaction deadlocks and rolls back a transaction or transactions to break the deadlock. InnoDB tries to pick small transactions to roll back, where the size of a transaction is determined by the number of rows inserted, updated, or deleted.

InnoDB is aware of table locks if innodb_table_locks = 1 (the default) and autocommit = 0, and the MySQL layer above it knows about row-level locks. Otherwise, InnoDB cannot detect deadlocks where a table lock set by a MySQL LOCK TABLES statement or a lock set by a storage engine other than InnoDB is involved. Resolve these situations by setting the value of the innodb_lock_wait_timeout system variable.

When InnoDB performs a complete rollback of a transaction, all locks set by the transaction are released. However, if just a single SQL statement is rolled back as a result of an error, some of the locks set by the statement may be preserved. This happens because InnoDB stores row locks in a format such that it cannot know afterward which lock was set by which statement.

If a SELECT calls a stored function in a transaction, and a statement within the function fails, that statement rolls back. Furthermore, if ROLLBACK is executed after that, the entire transaction rolls back.

For techniques to organize database operations to avoid deadlocks, see Section 14.2.2.11, "How to Cope with Deadlocks".

14.2.2.11 How to Cope with Deadlocks

This section builds on the conceptual information about deadlocks in Section 14.2.2.10, "Deadlock Detection and Rollback". It explains how to organize database operations to minimize deadlocks and the subsequent error handling required in applications.

Deadlocks are a classic problem in transactional databases, but they are not dangerous unless they are so frequent that you cannot run certain transactions at all. Normally, you must write your applications so that they are always prepared to re-issue a transaction if it gets rolled back because of a deadlock.

InnoDB uses automatic row-level locking. You can get deadlocks even in the case of transactions that just insert or delete a single row. That is because these operations are not really "atomic"; they automatically set locks on the (possibly several) index records of the row inserted or deleted.

You can cope with deadlocks and reduce the likelihood of their occurrence with the following techniques:

- At any time, issue the SHOW ENGINE INNODB STATUS command to determine the cause of the most recent deadlock. That can help you to tune your application to avoid deadlocks.

- If frequent deadlock warnings cause concern, collect more extensive debugging information by restarting the server with the innodb_print_all_deadlocks configuration option enabled. Information about each deadlock, not just the latest one, is recorded in the MySQL error log. Remove this option and restart the server again once the debugging is finished.

- Always be prepared to re-issue a transaction if it fails due to deadlock. Deadlocks are not dangerous. Just try again.

- Commit your transactions immediately after making a set of related changes. Small transactions are less prone to collision. In particular, do not leave an interactive mysql session open for a long time with an uncommitted transaction.

- If you use locking reads (SELECT ... FOR UPDATE or SELECT ... LOCK IN SHARE MODE), try using a lower isolation level such as READ COMMITTED.

- When modifying multiple tables within a transaction, or different sets of rows in the same table, do those operations in a consistent order each time. Then transactions form well-defined queues and do not deadlock. For example, organize database operations into functions within your application, or call stored routines, rather than coding multiple similar sequences of INSERT, UPDATE, and DELETE statements in different places.

- Add well-chosen indexes to your tables. Then your queries need to scan fewer index records and consequently set fewer locks. Use EXPLAIN SELECT to determine which indexes the MySQL server regards as the most appropriate for your queries.

- Use less locking. If you can afford to permit a SELECT to return data from an old snapshot, do not add the clause FOR UPDATE or LOCK IN SHARE MODE to it. Using the READ COMMITTED isolation level is good here, because each consistent read within the same transaction reads from its own fresh snapshot.

- If nothing else helps, serialize your transactions with table-level locks. The correct way to use LOCK TABLES with transactional tables, such as InnoDB tables, is to begin a transaction with SET autocommit = 0 (not START TRANSACTION) followed by LOCK TABLES, and to not call UNLOCK TABLES until you commit the transaction explicitly. For example, if you need to write to table t1 and read from table t2, you can do this:

```
SET autocommit=0;
LOCK TABLES t1 WRITE, t2 READ, ...;
... do something with tables t1 and t2 here ...
COMMIT;
UNLOCK TABLES;
```

Table-level locks prevent concurrent updates to the table, avoiding deadlocks at the expense of less responsiveness for a busy system.

- Another way to serialize transactions is to create an auxiliary “semaphore” table that contains just a single row. Have each transaction update that row before accessing other tables. In that way, all transactions happen in a serial fashion. Note that the InnoDB instant deadlock detection algorithm also works in this case, because the serializing lock is a row-level lock. With MySQL table-level locks, the timeout method must be used to resolve deadlocks.

14.2.2.12 InnoDB Multi-Versioning

InnoDB is a multi-versioned storage engine: it keeps information about old versions of changed rows, to support transactional features such as concurrency and rollback. This information is stored in the tablespace in a data structure called a rollback segment (after an analogous data structure in Oracle). InnoDB uses the information in the rollback segment to perform the undo operations needed in a transaction rollback. It also uses the information to build earlier versions of a row for a consistent read.

Internal Details of Multi-Versioning

Internally, InnoDB adds three fields to each row stored in the database. A 6-byte DB_TRX_ID field indicates the transaction identifier for the last transaction that inserted or updated the row. Also, a deletion is treated internally as an update where a special bit in the row is set to mark it as deleted. Each row also contains a 7-byte DB_ROLL_PTR field called the roll pointer. The roll pointer points to an undo log record written to the rollback segment. If the row was updated, the undo log record contains the information necessary to rebuild the content of the row before it was updated. A 6-byte DB_ROW_ID field contains a row ID that increases monotonically as new rows are inserted. If InnoDB generates a clustered index automatically, the index contains row ID values. Otherwise, the DB_ROW_ID column does not appear in any index.

Undo logs in the rollback segment are divided into insert and update undo logs. Insert undo logs are needed only in transaction rollback and can be discarded as soon as the transaction commits. Update undo logs are used also in consistent reads, but they can be discarded only after there is no transaction present for which InnoDB has assigned a snapshot that in a consistent read could need the information in the update undo log to build an earlier version of a database row.

Rollback Segments and Concurrent Transaction Limits

In MySQL 5.5, the limit on concurrent data-modifying transactions was significantly increased by removing a bottleneck that resulted from a single InnoDB rollback segment that supported a maximum of 1023 concurrent data-modifying transactions. The single rollback segment was divided into 128 segments, each supporting up to 1023 transactions, creating a new limit of approximately 128K concurrent transactions.

In MySQL 5.7.2, the 128K transaction limit is reduced to 96K in order to support the introduction of a new type of undo log for normal and compressed temporary tables and related objects. 32 of the 128 rollback segments are now reserved for temporary table transactions.

Each transaction that updates a temporary table, excluding read-only transactions, is assigned two rollback segments, one redo rollback segment and one non-redo rollback segment. Read-only transactions are only assigned non-redo rollback segments, as read-only transactions are only permitted to modify temporary tables.

This change leaves 96 segments, each supporting up to 1023 transactions, for a limit of 96K concurrent data-modifying transactions. The 96K limit assumes that transactions do not modify temporary tables. If all data-modifying transactions also modify temporary tables, the limit would be 32K concurrent transactions.

Guidelines for Managing Rollback Segments

Commit your transactions regularly, including those transactions that issue only consistent reads. Otherwise, `InnoDB` cannot discard data from the update undo logs, and the rollback segment may grow too big, filling up your tablespace.

The physical size of an undo log record in the rollback segment is typically smaller than the corresponding inserted or updated row. You can use this information to calculate the space needed for your rollback segment.

In the `InnoDB` multi-versioning scheme, a row is not physically removed from the database immediately when you delete it with an SQL statement. `InnoDB` only physically removes the corresponding row and its index records when it discards the update undo log record written for the deletion. This removal operation is called a purge, and it is quite fast, usually taking the same order of time as the SQL statement that did the deletion.

If you insert and delete rows in smallish batches at about the same rate in the table, the purge thread can start to lag behind and the table can grow bigger and bigger because of all the “dead” rows, making everything disk-bound and very slow. In such a case, throttle new row operations, and allocate more resources to the purge thread by tuning the `innodb_max_purge_lag` system variable. See Section 14.2.13, “`InnoDB` Startup Options and System Variables” for more information.

14.2.2.13 `InnoDB` Temporary Table Undo Logs

MySQL 5.7.2 introduces a new type of undo log for both normal and compressed temporary tables and related objects. The new type of undo log is not a redo log, as temporary tables are not recovered during crash recovery and do not require redo logs. Temporary table undo logs are, however, required for rollback, MVCC, and purging while the server is running. This special type of non-redo undo log benefits performance by avoiding redo logging I/O for temporary tables and related objects. The new undo log resides in the temporary tablespace. The default temporary tablespace file, `ibtmp1`, is located in the data directory by default and is always recreated on server startup. A user defined location for the temporary tablespace file can be specified by setting `innodb_temp_data_file_path`.

With this change, 32 rollback segments are now reserved for temporary table undo logs for transactions that modify temporary tables and related objects. This reduces the maximum number of rollback segments available for data-modifying transactions that generate undo records from 128 to 96, which reduces the limit on concurrent data-modifying transactions from 128K to 96K. For more information see Section 14.2.2.12, “`InnoDB` Multi-Versioning” and Section 14.2.6.7, “Limits on `InnoDB` Tables”.

14.2.2.14 `InnoDB` Table and Index Structures

This section describes how `InnoDB` tables, indexes, and their associated metadata is represented at the physical level. This information is primarily useful for performance tuning and troubleshooting.

Role of the `.frm` File for `InnoDB` Tables

MySQL stores its data dictionary information for tables in .frm files in database directories. Unlike other MySQL storage engines, `InnoDB` also encodes information about the table in its own internal data dictionary inside the tablespace. When MySQL drops a table or a database, it deletes one or more .frm files as well as the corresponding entries inside the InnoDB data dictionary. You cannot move InnoDB tables between databases simply by moving the .frm files.

Clustered and Secondary Indexes

Every InnoDB table has a special index called the clustered index where the data for the rows is stored. Typically, the clustered index is synonymous with the primary key. To get the best performance from queries, inserts, and other database operations, you must understand how InnoDB uses the clustered index to optimize the most common lookup and DML operations for each table.

- When you define a PRIMARY KEY on your table, InnoDB uses it as the clustered index. Define a primary key for each table that you create. If there is no logical unique and non-null column or set of columns, add a new auto-increment column, whose values are filled in automatically.
- If you do not define a PRIMARY KEY for your table, MySQL locates the first UNIQUE index where all the key columns are NOT NULL and InnoDB uses it as the clustered index.
- If the table has no PRIMARY KEY or suitable UNIQUE index, InnoDB internally generates a hidden clustered index on a synthetic column containing row ID values. The rows are ordered by the ID that InnoDB assigns to the rows in such a table. The row ID is a 6-byte field that increases monotonically as new rows are inserted. Thus, the rows ordered by the row ID are physically in insertion order.

How the Clustered Index Speeds Up Queries

Accessing a row through the clustered index is fast because the index search leads directly to the page with all the row data. If a table is large, the clustered index architecture often saves a disk I/O operation when compared to storage organizations that store row data using a different page from the index record. (For example, MyISAM uses one file for data rows and another for index records.)

How Secondary Indexes Relate to the Clustered Index

All indexes other than the clustered index are known as secondary indexes. In InnoDB, each record in a secondary index contains the primary key columns for the row, as well as the columns specified for the secondary index. InnoDB uses this primary key value to search for the row in the clustered index.

If the primary key is long, the secondary indexes use more space, so it is advantageous to have a short primary key.

For coding guidelines to take advantage of InnoDB clustered and secondary indexes, see Section 8.3.2, “Using Primary Keys” Section 8.3, “Optimization and Indexes” Section 8.5, “Optimizing for InnoDB Tables” Section 8.3.2, “Using Primary Keys”.

FULLTEXT Indexes

A special kind of index, the FULLTEXT index, helps InnoDB deal with queries and DML operations involving text-based columns and the words they contain. These indexes are physically represented as entire InnoDB tables, which are acted upon by SQL keywords such as the FULLTEXT clause of the CREATE INDEX statement, the MATCH() ... AGAINST [1271] syntax in a SELECT statement, and the OPTIMIZE TABLE statement. For usage information, see Section 12.9, “Full-Text Search Functions”.

You can examine FULLTEXT indexes by querying tables in the INFORMATION_SCHEMA database. You can see basic index information for FULLTEXT indexes by querying INNODB_SYS_INDEXES. Although InnoDB FULLTEXT indexes are represented by tables, which show up in INNODB_SYS_TABLES queries, the way to monitor the special text-processing aspects of a FULLTEXT index is to query the tables `INNODB_FT_CONFIG`, `INNODB_FT_INDEX_TABLE`, `INNODB_FT_INDEX_CACHE`, `INNODB_FT_DEFAULT_STOPWORD`, `INNODB_FT_DELETED`, and `INNODB_FT_BEING_DELETED`.

`InnoDB` `FULLTEXT` indexes are updated by the `OPTIMIZE TABLE` command, using a special mode controlled by the configuration options `innodb_ft_num_word_optimize` and `innodb_optimize_fulltext_only`.

Physical Structure of an `InnoDB` Index

All `InnoDB` indexes are B-trees where the index records are stored in the leaf pages of the tree. The default size of an index page is 16KB. When new records are inserted, `InnoDB` tries to leave 1/16 of the page free for future insertions and updates of the index records.

If index records are inserted in a sequential order (ascending or descending), the resulting index pages are about 15/16 full. If records are inserted in a random order, the pages are from 1/2 to 15/16 full. If the fill factor of an index page drops below 1/2, `InnoDB` tries to contract the index tree to free the page.

> **Note**
>
> You can specify the page size for all `InnoDB` tablespaces in a MySQL instance by setting the `innodb_page_size` configuration option before creating the instance. Once the page size for a MySQL instance is set, you cannot change it. Supported sizes are 16KB, 8KB, and 4KB, corresponding to the option values `16k`, `8k`, and `4k`.
>
> A MySQL instance using a particular `InnoDB` page size cannot use data files or log files from an instance that uses a different page size.

Insert Buffering

Database applications often insert new rows in the ascending order of the primary key. In this case, due to the layout of the clustered index in the same order as the primary key, insertions into an InnoDB table do not require random reads from a disk.

On the other hand, secondary indexes are usually nonunique, and insertions into secondary indexes happen in a relatively random order. In the same way, deletes and updates can affect data pages that are not adjacent in secondary indexes. This would cause a lot of random disk I/O operations without a special mechanism used in `InnoDB`.

When an index record is inserted, marked for deletion, or deleted from a nonunique secondary index, `InnoDB` checks whether the secondary index page is in the buffer pool. If that is the case, `InnoDB` applies the change directly to the index page. If the index page is not found in the buffer pool, `InnoDB` records the change in a special structure known as the insert buffer. The insert buffer is kept small so that it fits entirely in the buffer pool, and changes can be applied very quickly. This process is known as change buffering. (Formerly, it applied only to inserts and was called insert buffering. The data structure is still called the insert buffer.)

Disk I/O for Flushing the Insert Buffer

Periodically, the insert buffer is merged into the secondary index trees in the database. Often, it is possible to merge several changes into the same page of the index tree, saving disk I/O operations. It has been measured that the insert buffer can speed up insertions into a table up to 15 times.

The insert buffer merging may continue to happen *after* the transaction has been committed. In fact, it may continue to happen after a server shutdown and restart (see Section 14.2.17.2, “Starting `InnoDB` on a Corrupted Database”).

Insert buffer merging may take many hours when many secondary indexes must be updated and many rows have been inserted. During this time, disk I/O will be increased, which can cause significant slowdown on disk-bound queries. Another significant background I/O operation is the purge thread (see Section 14.2.2.12, "InnoDB Multi-Versioning").

Adaptive Hash Indexes

The feature known as the adaptive hash index (AHI) lets InnoDB perform more like an in-memory database on systems with appropriate combinations of workload and ample memory for the buffer pool, without sacrificing any transactional features or reliability. This feature is enabled by the innodb_adaptive_hash_index option, or turned off by the --skip-innodb_adaptive_hash_index at server startup.

Based on the observed pattern of searches, MySQL builds a hash index using a prefix of the index key. The prefix of the key can be any length, and it may be that only some of the values in the B-tree appear in the hash index. Hash indexes are built on demand for those pages of the index that are often accessed.

If a table fits almost entirely in main memory, a hash index can speed up queries by enabling direct lookup of any element, turning the index value into a sort of pointer. InnoDB has a mechanism that monitors index searches. If InnoDB notices that queries could benefit from building a hash index, it does so automatically.

With some workloads, the speedup from hash index lookups greatly outweighs the extra work to monitor index lookups and maintain the hash index structure. Sometimes, the read/write lock that guards access to the adaptive hash index can become a source of contention under heavy workloads, such as multiple concurrent joins. Queries with LIKE operators and % wildcards also tend not to benefit from the AHI. For workloads where the adaptive hash index is not needed, turning it off reduces unnecessary performance overhead. Because it is difficult to predict in advance whether this feature is appropriate for a particular system, consider running benchmarks with it both enabled and disabled, using a realistic workload. The architectural changes in MySQL 5.6 and higher make more workloads suitable for disabling the adaptive hash index than in earlier releases, although it is still enabled by default.

The hash index is always built based on an existingB-tree index on the table. InnoDB can build a hash index on a prefix of any length of the key defined for the B-tree, depending on the pattern of searches that InnoDB observes for the B-tree index. A hash index can be partial, covering only those pages of the index that are often accessed.

You can monitor the use of the adaptive hash index and the contention for its use in the SEMAPHORES section of the output of the SHOW ENGINE INNODB STATUS command. If you see many threads waiting on an RW-latch created in btr0sea.c, then it might be useful to disable adaptive hash indexing.

For more information about the performance characteristics of hash indexes, see Section 8.3.8, "Comparison of B-Tree and Hash Indexes".

Physical Row Structure

The physical row structure for an InnoDB table depends on the row format specified when the table was created. By default, InnoDB uses the Antelope file format and its COMPACT row format. The REDUNDANT format is available to retain compatibility with older versions of MySQL. When you enable the innodb_file_per_table setting, you can also make use of the newer Barracuda file format, with its DYNAMIC and COMPRESSED row formats, as explained in Section 14.2.9, "InnoDB Row Storage and Row Formats" and Section 14.2.7, "InnoDB Compressed Tables".

To check the row format of an InnoDB table, use SHOW TABLE STATUS.

The COMPACT row format decreases row storage space by about 20% at the cost of increasing CPU use for some operations. If your workload is a typical one that is limited by cache hit rates and disk speed, COMPACT format is likely to be faster. If the workload is a rare case that is limited by CPU speed, COMPACT format might be slower.

Rows in InnoDB tables that use REDUNDANT row format have the following characteristics:

- Each index record contains a 6-byte header. The header is used to link together consecutive records, and also in row-level locking.
- Records in the clustered index contain fields for all user-defined columns. In addition, there is a 6-byte transaction ID field and a 7-byte roll pointer field.
- If no primary key was defined for a table, each clustered index record also contains a 6-byte row ID field.
- Each secondary index record also contains all the primary key fields defined for the clustered index key that are not in the secondary index.
- A record contains a pointer to each field of the record. If the total length of the fields in a record is less than 128 bytes, the pointer is one byte; otherwise, two bytes. The array of these pointers is called the record directory. The area where these pointers point is called the data part of the record.
- Internally, InnoDB stores fixed-length character columns such as CHAR(10) in a fixed-length format. InnoDB does not truncate trailing spaces from VARCHAR columns.
- An SQL NULL value reserves one or two bytes in the record directory. Besides that, an SQL NULL value reserves zero bytes in the data part of the record if stored in a variable length column. In a fixed-length column, it reserves the fixed length of the column in the data part of the record. Reserving the fixed space for NULL values enables an update of the column from NULL to a non-NULL value to be done in place without causing fragmentation of the index page.

Rows in InnoDB tables that use COMPACT row format have the following characteristics:

- Each index record contains a 5-byte header that may be preceded by a variable-length header. The header is used to link together consecutive records, and also in row-level locking.
- The variable-length part of the record header contains a bit vector for indicating NULL columns. If the number of columns in the index that can be NULL is *N*, the bit vector occupies CEILING(*N*/8) bytes. (For example, if there are anywhere from 9 to 15 columns that can be NULL, the bit vector uses two bytes.) Columns that are NULL do not occupy space other than the bit in this vector. The variable-length part of the header also contains the lengths of variable-length columns. Each length takes one or two bytes, depending on the maximum length of the column. If all columns in the index are NOT NULL and have a fixed length, the record header has no variable-length part.
- For each non-NULL variable-length field, the record header contains the length of the column in one or two bytes. Two bytes will only be needed if part of the column is stored externally in overflow pages or the maximum length exceeds 255 bytes and the actual length exceeds 127 bytes. For an externally stored column, the 2-byte length indicates the length of the internally stored part plus the 20-byte pointer to the externally stored part. The internal part is 768 bytes, so the length is 768+20. The 20-byte pointer stores the true length of the column.
- The record header is followed by the data contents of the non-NULL columns.
- Records in the clustered index contain fields for all user-defined columns. In addition, there is a 6-byte transaction ID field and a 7-byte roll pointer field.
- If no primary key was defined for a table, each clustered index record also contains a 6-byte row ID field.
- Each secondary index record also contains all the primary key fields defined for the clustered index key that are not in the secondary index. If any of these primary key fields are variable length, the record

header for each secondary index will have a variable-length part to record their lengths, even if the secondary index is defined on fixed-length columns.

- Internally, InnoDB stores fixed-length, fixed-width character columns such as CHAR(10) in a fixed-length format. InnoDB does not truncate trailing spaces from VARCHAR columns.

- Internally, InnoDB attempts to store UTF-8 CHAR(*N*) columns in *N* bytes by trimming trailing spaces. (With REDUNDANT row format, such columns occupy 3 × *N* bytes.) Reserving the minimum space *N* in many cases enables column updates to be done in place without causing fragmentation of the index page.

14.2.3 InnoDB Configuration

The first decisions to make about InnoDB configuration involve how to lay out InnoDB data files, and how much memory to allocate for the InnoDB storage engine. You record these choices either by recording them in a configuration file that MySQL reads at startup, or by specifying them as command-line options in a startup script. The full list of options, descriptions, and allowed parameter values is at Section 14.2.13, “InnoDB Startup Options and System Variables”.

Overview of InnoDB Tablespace and Log Files

Two important disk-based resources managed by the InnoDB storage engine are its tablespace data files and its log files. If you specify no InnoDB configuration options, MySQL creates an auto-extending data file, slightly larger than 12MB, named ibdata1 and two log files named ib_logfile0 and ib_logfile1 in the MySQL data directory. Their size is given by the size of the innodb_log_file_size system variable. To get good performance, explicitly provide InnoDB parameters as discussed in the following examples. Naturally, edit the settings to suit your hardware and requirements.

The examples shown here are representative. See Section 14.2.13, “InnoDB Startup Options and System Variables” for additional information about InnoDB-related configuration parameters.

Considerations for Storage Devices

In some cases, database performance improves if the data is not all placed on the same physical disk. Putting log files on a different disk from data is very often beneficial for performance. The example illustrates how to do this. It places the two data files on different disks and places the log files on the third disk. InnoDB fills the tablespace beginning with the first data file. You can also use raw disk partitions (raw devices) as InnoDB data files, which may speed up I/O. See Section 14.2.5.8, “Using Raw Disk Partitions for the Shared Tablespace”.

> **Caution**
>
> InnoDB is a transaction-safe (ACID compliant) storage engine for MySQL that has commit, rollback, and crash-recovery capabilities to protect user data. **However, it cannot do so** if the underlying operating system or hardware does not work as advertised. Many operating systems or disk subsystems may delay or reorder write operations to improve performance. On some operating systems, the very fsync() system call that should wait until all unwritten data for a file has been flushed might actually return before the data has been flushed to stable storage. Because of this, an operating system crash or a power outage may destroy recently committed data, or in the worst case, even corrupt the database because of write operations having been reordered. If data integrity is important to you, perform some “pull-the-plug” tests before using anything in production. On Mac OS X 10.3 and up, InnoDB uses a special fcntl() file flush method. Under Linux, it is advisable to **disable the write-back cache**.

On ATA/SATA disk drives, a command such `hdparm -W0 /dev/hda` may work to disable the write-back cache. **Beware that some drives or disk controllers may be unable to disable the write-back cache.**

Caution

If reliability is a consideration for your data, do not configure `InnoDB` to use data files or log files on NFS volumes. Potential problems vary according to OS and version of NFS, and include such issues as lack of protection from conflicting writes, and limitations on maximum file sizes.

Specifying the Location and Size for InnoDB Tablespace Files

To set up the `InnoDB` tablespace files, use the `innodb_data_file_path` option in the `[mysqld]` section of the `my.cnf` option file. On Windows, you can use `my.ini` instead. The value of `innodb_data_file_path` should be a list of one or more data file specifications. If you name more than one data file, separate them by semicolon (“`;`”) characters:

```
innodb_data_file_path=datafile_spec1[;datafile_spec2]...
```

For example, the following setting explicitly creates a minimally sized system tablespace:

```
[mysqld]
innodb_data_file_path=ibdata1:12M:autoextend
```

This setting configures a single 12MB data file named `ibdata1` that is auto-extending. No location for the file is given, so by default, `InnoDB` creates it in the MySQL data directory.

Sizes are specified using `K`, `M`, or `G` suffix letters to indicate units of KB, MB, or GB.

A tablespace containing a fixed-size 50MB data file named `ibdata1` and a 50MB auto-extending file named `ibdata2` in the data directory can be configured like this:

```
[mysqld]
innodb_data_file_path=ibdata1:50M;ibdata2:50M:autoextend
```

The full syntax for a data file specification includes the file name, its size, and several optional attributes:

```
file_name:file_size[:autoextend[:max:max_file_size]]
```

The `autoextend` and `max` attributes can be used only for the last data file in the `innodb_data_file_path` line.

If you specify the `autoextend` option for the last data file, `InnoDB` extends the data file if it runs out of free space in the tablespace. The increment is 8MB at a time by default. To modify the increment, change the `innodb_autoextend_increment` system variable.

If the disk becomes full, you might want to add another data file on another disk. For tablespace reconfiguration instructions, see Section 14.2.5.7, “Changing the Number or Size of `InnoDB` Log Files and Resizing the `InnoDB` Tablespace”.

`InnoDB` is not aware of the file system maximum file size, so be cautious on file systems where the maximum file size is a small value such as 2GB. To specify a maximum size for an auto-extending data file, use the max attribute following the autoextend attribute. Use the max attribute only in cases where constraining disk usage is of critical importance, because exceeding the maximum size causes a fatal error, possibly including a crash. The following configuration permits ibdata1 to grow up to a limit of 500MB:

```
[mysqld]
innodb_data_file_path=ibdata1:12M:autoextend:max:500M
```

InnoDB creates tablespace files in the MySQL data directory by default. To specify a location explicitly, use the innodb_data_home_dir option. For example, to use two files named ibdata1 and ibdata2 but create them in the /ibdata directory, configure InnoDB like this:

```
[mysqld]
innodb_data_home_dir = /ibdata
innodb_data_file_path=ibdata1:50M;ibdata2:50M:autoextend
```

Note

InnoDB does not create directories, so make sure that the /ibdata directory exists before you start the server. This is also true of any log file directories that you configure. Use the Unix or DOS mkdir command to create any necessary directories.

Make sure that the MySQL server has the proper access rights to create files in the data directory. More generally, the server must have access rights in any directory where it needs to create data files or log files.

InnoDB forms the directory path for each data file by textually concatenating the value of innodb_data_home_dir to the data file name, adding a path name separator (slash or backslash) between values if necessary. If the innodb_data_home_dir option is not specified in my.cnf at all, the default value is the “dot” directory ./, which means the MySQL data directory. (The MySQL server changes its current working directory to its data directory when it begins executing.)

If you specify innodb_data_home_dir as an empty string, you can specify absolute paths for the data files listed in the innodb_data_file_path value. The following example is equivalent to the preceding one:

```
[mysqld]
innodb_data_home_dir =
innodb_data_file_path=/ibdata/ibdata1:50M;/ibdata/ibdata2:50M:autoextend
```

Specifying InnoDB Configuration Options

Sample my.cnf file for small systems. Suppose that you have a computer with 512MB RAM and one hard disk. The following example shows possible configuration parameters in my.cnf or my.ini for InnoDB, including the autoextend attribute. The example suits most users, both on Unix and Windows, who do not want to distribute InnoDB data files and log files onto several disks. It creates an auto-extending data file ibdata1 and two InnoDB log files ib_logfile0 and ib_logfile1 in the MySQL data directory.

```
[mysqld]
# You can write your other MySQL server options here
# ...
# Data files must be able to hold your data and indexes.
# Make sure that you have enough free disk space.
innodb_data_file_path = ibdata1:12M:autoextend
#
# Set buffer pool size to 50-80% of your computer's memory
innodb_buffer_pool_size=256M
#
# Set the log file size to about 25% of the buffer pool size
innodb_log_file_size=64M
innodb_log_buffer_size=8M
#
innodb_flush_log_at_trx_commit=1
```

Note that data files must be less than 2GB in some file systems. The combined size of the log files can be up to 512GB. The combined size of data files must be slightly larger than 10MB.

Setting Up the InnoDB System Tablespace

When you create an InnoDB system tablespace for the first time, it is best that you start the MySQL server from the command prompt. InnoDB then prints the information about the database creation to the screen, so you can see what is happening. For example, on Windows, if mysqld is located in C:\Program Files\MySQL\MySQL Server 5.7\bin, you can start it like this:

```
C:\> "C:\Program Files\MySQL\MySQL Server 5.7\bin\mysqld" --console
```

If you do not send server output to the screen, check the server's error log to see what InnoDB prints during the startup process.

For an example of what the information displayed by InnoDB should look like, see Section 14.2.5.1, “Creating the InnoDB Tablespace”.

Editing the MySQL Configuration File

You can place InnoDB options in the [mysqld] group of any option file that your server reads when it starts. The locations for option files are described in Section 4.2.3.3, “Using Option Files”.

If you installed MySQL on Windows using the installation and configuration wizards, the option file will be the my.ini file located in your MySQL installation directory. See Section 2.3.3, “Installing MySQL on Microsoft Windows Using MySQL Installer”.

If your PC uses a boot loader where the C: drive is not the boot drive, your only option is to use the my.ini file in your Windows directory (typically C:\WINDOWS). You can use the SET command at the command prompt in a console window to print the value of WINDIR:

```
C:\> SET WINDIR
windir=C:\WINDOWS
```

To make sure that mysqld reads options only from a specific file, use the --defaults-file option as the first option on the command line when starting the server:

```
mysqld --defaults-file=your_path_to_my_cnf
```

Sample my.cnf file for large systems. Suppose that you have a Linux computer with 2GB RAM and three 60GB hard disks at directory paths /, /dr2 and /dr3. The following example shows possible configuration parameters in my.cnf for InnoDB.

```
[mysqld]
# You can write your other MySQL server options here
# ...
innodb_data_home_dir =
#
# Data files must be able to hold your data and indexes
innodb_data_file_path = /db/ibdata1:2000M;/dr2/db/ibdata2:2000M:autoextend
#
# Set buffer pool size to 50-80% of your computer's memory,
# but make sure on Linux x86 total memory usage is < 2GB
innodb_buffer_pool_size=1G
innodb_log_group_home_dir = /dr3/iblogs
#
# Set the log file size to about 25% of the buffer pool size
innodb_log_file_size=250M
innodb_log_buffer_size=8M
#
innodb_flush_log_at_trx_commit=1
innodb_lock_wait_timeout=50
#
# Uncomment the next line if you want to use it
#innodb_thread_concurrency=5
```

Determining the Maximum Memory Allocation for InnoDB

Warning

On 32-bit GNU/Linux x86, be careful not to set memory usage too high. glibc may permit the process heap to grow over thread stacks, which crashes your server. It is a risk if the value of the following expression is close to or exceeds 2GB:

```
innodb_buffer_pool_size
+ key_buffer_size
+ max_connections*(sort_buffer_size+read_buffer_size+binlog_cache_size)
+ max_connections*2MB
```

Each thread uses a stack (often 2MB, but only 256KB in MySQL binaries provided by Oracle Corporation.) and in the worst case also uses sort_buffer_size + read_buffer_size additional memory.

Tuning other mysqld server parameters. The following values are typical and suit most users:

```
[mysqld]
skip-external-locking
max_connections=200
read_buffer_size=1M
sort_buffer_size=1M
#
# Set key_buffer to 5 - 50% of your RAM depending on how much
# you use MyISAM tables, but keep key_buffer_size + InnoDB
# buffer pool size < 80% of your RAM
key_buffer_size=value
```

On Linux, if the kernel is enabled for large page support, InnoDB can use large pages to allocate memory for its buffer pool and additional memory pool. See Section 8.11.4.2, “Enabling Large Page Support”.

14.2.3.1 Configuring InnoDB for Read-Only Operation

You can now query InnoDB tables where the MySQL data directory is on read-only media, by enabling the --innodb-read-only configuration option at server startup.

How to Enable

To prepare an instance for read-only operation, make sure all the necessary information is flushed to the data files before storing it on the read-only medium. Run the server with change buffering disabled (`innodb_change_buffering=0`) and do a slow shutdown.

To enable read-only mode for an entire MySQL instance, specify the following configuration options at server startup:

- `--innodb-read-only=1`
- If the instance is on read-only media such as a DVD or CD, or the `/var` directory is not writeable by all: `--pid-file=path_on_writeable_media` and `--event-scheduler=disabled`

Usage Scenarios

This mode of operation is appropriate in situations such as:

- Distributing a MySQL application, or a set of MySQL data, on a read-only storage medium such as a DVD or CD.
- Multiple MySQL instances querying the same data directory simultaneously, typically in a data warehousing configuration. You might use this technique to avoid bottlenecks that can occur with a heavily loaded MySQL instance, or you might use different configuration options for the various instances to tune each one for particular kinds of queries.
- Querying data that has been put into a read-only state for security or data integrity reasons, such as archived backup data.

> **Note**
>
> This feature is mainly intended for flexibility in distribution and deployment, rather than raw performance based on the read-only aspect. See Optimizations for Read-Only Transactions for ways to tune the performance of read-only queries, which do not require making the entire server read-only.

How It Works

When the server is run in read-only mode through the `--innodb-read-only` option, certain `InnoDB` features and components are reduced or turned off entirely:

- No change buffering is done, in particular no merges from the change buffer. To make sure the change buffer is empty when you prepare the instance for read-only operation, disable change buffering (`innodb_change_buffering=0`) and do a slow shutdown first.
- There is no crash recovery phase at startup. The instance must have performed a slow shutdown before being put into the read-only state.
- Because the redo log is not used in read-only operation, you can set `innodb_log_file_size` to the smallest size possible (1 MB) before making the instance read-only.
- All background threads other than I/O read threads are turned off. As a consequence, a read-only instance cannot encounter any deadlocks.
- Information about deadlocks, monitor output, and so on is not written to temporary files. As a consequence, `SHOW ENGINE INNODB STATUS` does not produce any output.

- If the MySQL server is started with --innodb-read-only but the data directory is still on writeable media, the root user can still perform DCL operations such as GRANT and REVOKE.
- Changes to configuration option settings that would normally change the the behavior of write operations, have no effect when the server is in read-only mode.
- The MVCC processing to enforce isolation levels is turned off. All queries read the latest version of a record, because update and deletes are not possible.
- The undo log is not used. Disable any settings for the innodb_undo_tablespaces and innodb_undo_directory configuration options.

14.2.4 InnoDB Administration

Administration tasks related to InnoDB mainly involve these aspects:

- Managing the data files that represent the system tablespace, InnoDB tables, and their associated indexes. You can change the way these files are laid out and divided, which affects both performance and the features available for specific tables.
- Managing the redo log files that are used for crash recovery. You can specify the size of these files.
- Making sure that InnoDB is used for the tables where it is intended, rather than a different storage engine.
- General administrative tasks related to performance. You might consult with application developers during the application design phase, monitor performance on an ongoing basis to ensure the system settings are working well, and diagnose and help fix performance and capacity issues that arise suddenly.

Since InnoDB tables are now the default for MySQL, much of the associated administration material is now in the main “Administration” chapter, Chapter 5, *MySQL Server Administration*.

14.2.5 InnoDB Tablespace Management

14.2.5.1 Creating the InnoDB Tablespace

Suppose that you have installed MySQL and have edited your option file so that it contains the necessary InnoDB configuration parameters. Before starting MySQL, verify that the directories you have specified for InnoDB data files and log files exist and that the MySQL server has access rights to those directories. InnoDB does not create directories, only files. Check also that you have enough disk space for the data and log files.

It is best to run the MySQL server mysqld from the command prompt when you first start the server with InnoDB enabled, not from mysqld_safe or as a Windows service. When you run from a command prompt you see what mysqld prints and what is happening. On Unix, just invoke mysqld. On Windows, start mysqld with the --console option to direct the output to the console window.

When you start the MySQL server after initially configuring InnoDB in your option file, InnoDB creates your data files and log files, and prints something like this:

```
2013-09-24T12:55:18.897250Z 0 [Note] InnoDB: The first specified data file "ibdata1" did not exist : a new dat
2013-09-24T12:55:18.897299Z 0 [Note] InnoDB: Need to create new data file "ibdata2"
2013-09-24T12:55:18.897492Z 0 [Note] InnoDB: Setting file "./ibdata1" size to 128 MB
2013-09-24T12:55:18.897509Z 0 [Note] InnoDB: Database physically writes the file full: wait ...
```

```
2013-09-24T12:55:19.013723Z 0 [Note] InnoDB: Setting file "./ibdata2" size to 250 MB
2013-09-24T12:55:19.013766Z 0 [Note] InnoDB: Database physically writes the file full: wait ...
2013-09-24T12:55:19.131808Z 0 [Note] InnoDB: Setting log file ./ib_logfile101 size to 48 MB
2013-09-24T12:55:19.571493Z 0 [Note] InnoDB: Setting log file ./ib_logfile1 size to 48 MB
2013-09-24T12:55:20.226902Z 0 [Note] InnoDB: Renaming log file ./ib_logfile101 to ./ib_logfile0
2013-09-24T12:55:20.227251Z 0 [Warning] InnoDB: New log files created, LSN=45781
2013-09-24T12:55:21.227716Z 0 [Note] InnoDB: Creating shared tablespace for temporary tables
2013-09-24T12:55:21.228286Z 0 [Note] InnoDB: Setting file "./ibtmp1" size to 12 MB
2013-09-24T12:55:21.228334Z 0 [Note] InnoDB: Database physically writes the file full: wait ...
2013-09-24T12:55:21.329536Z 0 [Note] InnoDB: Doublewrite buffer not found: creating new
2013-09-24T12:55:21.476956Z 0 [Note] InnoDB: Doublewrite buffer created
2013-09-24T12:55:22.077524Z 0 [Note] InnoDB: 96 redo rollback segment(s) found. 96 redo rollback segment(s
2013-09-24T12:55:22.077564Z 0 [Note] InnoDB: 32 non-redo rollback segment(s) are active.
2013-09-24T12:55:22.182853Z 0 [Warning] InnoDB: Creating foreign key constraint system tables.
2013-09-24T12:55:22.195621Z 0 [Note] InnoDB: Foreign key constraint system tables created
2013-09-24T12:55:22.195791Z 0 [Note] InnoDB: Creating tablespace and datafile system tables.
2013-09-24T12:55:22.202725Z 0 [Note] InnoDB: Tablespace and datafile system tables created.
2013-09-24T12:55:22.202844Z 0 [Note] InnoDB: Waiting for purge to start
2013-09-24T12:55:22.253342Z 0 [Note] InnoDB: 5.7.5 started; log sequence number 0
2013-09-24T12:55:22.630676Z 0 [Note] mysqld: ready for connections.
```

At this point InnoDB has initialized its tablespace and log files. You can connect to the MySQL server with the usual MySQL client programs like mysql. When you shut down the MySQL server with mysqladmin shutdown, the output is like this:

```
2013-09-24T13:03:08.029127Z 0 [Note] mysqld: Normal shutdown
2013-09-24T13:03:10.057269Z 0 [Note] InnoDB: Starting shutdown...
2013-09-24T13:03:10.857032Z 0 [Note] InnoDB: Shutdown completed; log sequence number 1566036
2013-09-24T13:03:10.863259Z 0 [Note] mysqld: Shutdown complete
```

You can look at the data file and log directories and you see the files created there. When MySQL is started again, the data files and log files have been created already, so the output is much briefer:

```
2013-09-24T13:04:38.639397Z 0 [Note] InnoDB: Creating shared tablespace for temporary tables
2013-09-24T13:04:38.639660Z 0 [Note] InnoDB: Setting file "./ibtmp1" size to 12 MB
2013-09-24T13:04:38.639684Z 0 [Note] InnoDB: Database physically writes the file full: wait ...
2013-09-24T13:04:38.872720Z 0 [Note] InnoDB: 96 redo rollback segment(s) found. 96 redo rollback segment(s
2013-09-24T13:04:38.872760Z 0 [Note] InnoDB: 32 non-redo rollback segment(s) are active.
2013-09-24T13:04:38.906989Z 0 [Note] InnoDB: Waiting for purge to start
2013-09-24T13:04:38.958353Z 0 [Note] InnoDB: 5.7.5 started; log sequence number 1566036
2013-09-24T13:04:39.281825Z 0 [Note] mysqld: ready for connections.
```

If you add the innodb_file_per_table option to my.cnf, InnoDB stores each table in its own .ibd file, in the same MySQL database directory where the .frm file is created. See Section 14.2.5.2, "InnoDB File-Per-Table Mode".

14.2.5.2 InnoDB File-Per-Table Mode

Historically, all InnoDB tables and indexes were stored in the system tablespace. This monolithic approach was targeted at machines dedicated entirely to database processing, with carefully planned data growth, where any disk storage allocated to MySQL would never be needed for other purposes. InnoDB's file-per-table mode is a more flexible alternative, where you store each InnoDB table and its indexes in a separate file. Each such .ibd file represents a separate tablespace. This mode is controlled by the innodb_file_per_table configuration option, and is the default in MySQL 5.6.6 and higher.

Advantages of File-Per-Table Mode

- You can reclaim operating system disk space when truncating or dropping a table. For tables created when file-per-table mode is turned off, truncating or dropping the tables creates free space internally in the ibdata files but the free space can only be used for new InnoDB data.

- The TRUNCATE TABLE operation is faster when run on individual .ibd files.
- You can store specific tables on separate storage devices, for I/O optimization, space management, or backup purposes. In previous releases, you had to move entire database directories to other drives and create symbolic links in the MySQL data directory, as described in Section 8.11.3.1, “Using Symbolic Links”. In MySQL 5.6.6 and higher, you can specify the location of each table using the syntax CREATE TABLE ... DATA DIRECTORY = *absolute_path_to_directory*, as explained in Section 14.2.5.4, “Specifying the Location of a Tablespace”.
- You can run OPTIMIZE TABLE to compact or recreate a tablespace. When you run an OPTIMIZE TABLE, InnoDB will create a new .ibd file with a temporary name, using only the space required to store actual data. When the optimization is complete, InnoDB removes the old .ibd file and replaces it with the new .ibd file. If the previous .ibd file had grown significantly but actual data only accounted for a portion of its size, running OPTIMIZE TABLE allows you to reclaim the unused space.
- You can move individual InnoDB tables rather than entire databases.
- You can copy individual InnoDB tables from one MySQL instance to another (known as the transportable tablespace feature).
- Tables created when innodb_file_per_table is enabled can use the Barracuda file format. The Barracuda file format enables features such as compressed and dynamic row formats. Tables created when innodb_file_per_table is off cannot use these features. To take advantage of these features for an existing table, you can turn on the file-per-table setting and run ALTER TABLE *t* ENGINE=INNODB on the existing table. Before converting tables, refer to Section 14.2.6.4, “Converting Tables from MyISAM to InnoDB”.
- You can enable more efficient storage for tables with large BLOB or text columns using the dynamic row format.
- Using innodb_file_per_table may improve chances for a successful recovery and save time if a corruption occurs, a server cannot be restarted, or backup and binary logs are unavailable.
- You can back up or restore a single table quickly, without interrupting the use of other InnoDB tables, using the MySQL Enterprise Backup product. See Backing Up and Restoring a Single .ibd File for the procedure and restrictions.
- File-per-table mode allows you to excluded tables from a backup. This is beneficial if you have tables that require backup less frequently or on a different schedule.
- File-per-table mode is convenient for per-table status reporting when copying or backing up tables.
- File-per-table mode allows you to monitor table size at a file system level, without accessing MySQL.
- Common Linux file systems do not permit concurrent writes to a single file when innodb_flush_method is set to O_DIRECT. As a result, there are possible performance improvements when using innodb_file_per_table in conjunction with innodb_flush_method.
- If innodb_file_per_table is disabled, there is one shared tablespace (the system tablespace) for tables, the data dictionary, and undo logs. This single tablespace has a 64TB size limit. If innodb_file_per_table is enabled, each table has its own tablespace, each with a 64TB size limit. See Section E.10.3, “Limits on Table Size” for related information.

Potential Disadvantages of File-Per-Table Mode

- With innodb_file_per_table, each table may have unused table space, which can only be utilized by rows of the same table. This could lead to more rather than less wasted table space if not properly managed.

- fsync operations must run on each open table rather than on a single file. Because there is a separate fsync operation for each file, write operations on multiple tables cannot be combined into a single I/O operation. This may require InnoDB to perform a higher total number of fsync operations.
- mysqld must keep 1 open file handle per table, which may impact performance if you have numerous tables.
- More file descriptors are used.
- innodb_file_per_table is on by default in MySQL 5.6.6 and higher. You may want to consider disabling it if backward compatibility with MySQL 5.5 or 5.1 is a concern. Disabling innodb_file_per_table prevents ALTER TABLE from moving an InnoDB table from the system tablespace to an individual .ibd file in cases where ALTER TABLE recreates the table (ALGORITHM=COPY).

  For example, when restructuring the clustered index for an InnoDB table, the table is re-created using the current settings for innodb_file_per_table. This behavior does not apply when adding or dropping InnoDB secondary indexes. When a secondary index is created without rebuilding the table, the index is stored in the same file as the table data, regardless of the current innodb_file_per_table setting.
- If many tables are growing there is potential for more fragmentation which can impede DROP TABLE and table scan performance. However, when fragmentation is managed, having files in their own tablespace can improve performance.
- The buffer pool is scanned when dropping a per-table tablespace, which can take several seconds for buffer pools that are tens of gigabytes in size. The scan is performed with a broad internal lock, which may delay other operations. Tables in the shared tablespace are not affected.
- The innodb_autoextend_increment variable, which defines increment size (in MB) for extending the size of an auto-extending shared tablespace file when it becomes full, does not apply to file-per-table tablespace files. File-per-table tablespace files are auto-extending regardless of the value of innodb_autoextend_increment. The initial extensions are by small amounts, after which extensions occur in increments of 4MB.

14.2.5.3 Enabling and Disabling File-Per-Table Mode

To make file-per-table mode the default for a MySQL server, start the server with the --innodb_file_per_table command-line option, or add this line to the [mysqld] section of my.cnf:

```
[mysqld]
innodb_file_per_table
```

You can also issue the command while the server is running:

```
SET GLOBAL innodb_file_per_table=1;
```

With file-per-table mode enabled, InnoDB stores each newly created table in its own *tbl_name*.ibd file in the appropriate database directory. Unlike the MyISAM storage engine, with its separate *tbl_name*.MYD and *tbl_name*.MYI files for indexes and data, InnoDB stores the data and the indexes together in a single .ibd file. The *tbl_name*.frm file is still created as usual.

If you remove innodb_file_per_table from your startup options and restart the server, or turn it off with the SET GLOBAL command, InnoDB creates any new tables inside the system tablespace.

You can always read and write any InnoDB tables, regardless of the file-per-table setting.

To move a table from the system tablespace to its own tablespace, or vice versa, change the innodb_file_per_table setting and rebuild the table:

```
-- Move table from system tablespace to its own tablespace.
SET GLOBAL innodb_file_per_table=1;
ALTER TABLE table_name ENGINE=InnoDB;
-- Move table from its own tablespace to system tablespace.
SET GLOBAL innodb_file_per_table=0;
ALTER TABLE table_name ENGINE=InnoDB;
```

Note

InnoDB always needs the system tablespace because it puts its internal data dictionary and undo logs there. The .ibd files are not sufficient for InnoDB to operate.

When a table is moved out of the system tablespace into its own .ibd file, the data files that make up the system tablespace remain the same size. The space formerly occupied by the table can be reused for new InnoDB data, but is not reclaimed for use by the operating system. When moving large InnoDB tables out of the system tablespace, where disk space is limited, you might prefer to turn on innodb_file_per_table and then recreate the entire instance using the mysqldump command.

14.2.5.4 Specifying the Location of a Tablespace

To create a new InnoDB file-per-table tablespace in a specific location outside the MySQL data directory, use the DATA DIRECTORY = *absolute_path_to_directory* clause of the CREATE TABLE statement.

Plan the location in advance, because you cannot use the DATA DIRECTORY clause with the ALTER TABLE statement. The directory you specify could be on another storage device with particular performance or capacity characteristics, such as a fast SSD or a high-capacity HDD.

Within the destination directory, MySQL creates a subdirectory corresponding to the database name, and within that a .ibd file for the new table. In the database directory underneath the MySQL DATADIR directory, MySQL creates a *table_name*.isl file containing the path name for the table. The .isl file is treated by MySQL like a symbolic link. (Using actual symbolic links has never been supported for InnoDB tables.)

The following example shows how you might run a small development or test instance of MySQL on a laptop with a primary hard drive that is 95% full, and place a new table named EXTERNAL on a different storage device with more free space. The shell commands show the different paths to the LOCAL table in its default location under the DATADIR directory, and the EXTERNAL table in the location you specified:

```
mysql> \! df -k .
Filesystem   1024-blocks      Used Available Capacity  iused   ifree %iused  Mounted on
/dev/disk0s2   244277768 231603532  12418236    95% 57964881 3104559   95%   /

mysql> use test;
Database changed
mysql> show variables like 'innodb_file_per_table';
+-----------------------+-------+
| Variable_name         | Value |
+-----------------------+-------+
| innodb_file_per_table | ON    |
+-----------------------+-------+
1 row in set (0.00 sec)

mysql> \! pwd
/usr/local/mysql
mysql> create table local (x int unsigned not null primary key);
Query OK, 0 rows affected (0.03 sec)

mysql> \! ls -l data/test/local.ibd
-rw-rw----  1 cirrus  staff  98304 Nov 13 15:24 data/test/local.ibd

mysql> create table external (x int unsigned not null primary key) data directory = '/volumes/external1/da
Query OK, 0 rows affected (0.03 sec)

mysql> \! ls -l /volumes/external1/data/test/external.ibd
-rwxrwxrwx  1 cirrus  staff  98304 Nov 13 15:34 /volumes/external1/data/test/external.ibd

mysql> select count(*) from local;
+----------+
| count(*) |
+----------+
|        0 |
+----------+
1 row in set (0.01 sec)

mysql> select count(*) from external;
+----------+
| count(*) |
+----------+
|        0 |
+----------+
1 row in set (0.01 sec)
```

Notes:

- MySQL initially holds the .ibd file open, preventing you from dismounting the device, but might eventually close the table if the server is busy. Be careful not to accidentally dismount the external device while MySQL is running, or to start MySQL while the device is disconnected. Attempting to access a table when the associated .ibd file is missing causes a serious error that requires a server restart.

  The server restart might fail if the .ibd file is still not at the expected path. In this case, manually remove the *table_name*.isl file in the database directory, and after restarting do a DROP TABLE to delete the .frm file and remove the information about the table from the data dictionary.

- Do not put MySQL tables on an NFS-mounted volume. NFS uses a message-passing protocol to write to files, which could cause data inconsistency if network messages are lost or received out of order.

- If you use an LVM snapshot, file copy, or other file-based mechanism to back up the .ibd file, always use the FLUSH TABLES ... FOR EXPORT statement first to make sure all changes that were buffered in memory are flushed to disk before the backup occurs.

- The DATA DIRECTORY clause is a supported alternative to using symbolic links, which has always been problematic and was never supported for individual InnoDB tables.

14.2.5.5 Copying Tablespaces to Another Server (Transportable Tablespaces)

This section describes how to copy file-per-table tablespaces (.idb files) from one database server to another using the Transportable Tablespace feature. Prior to MySQL 5.7.4, only non-partitioned InnoDB tables are supported. As of MySQL 5.7.4, partitioned InnoDB tables and individual InnoDB table partitions and subpartitions are also supported.

For information about other InnoDB table copying methods, see Section 14.2.6.2, "Moving or Copying InnoDB Tables to Another Machine".

There are many reasons why you might copy an InnoDB file-per-table tablespace to a different database server:

- To run reports without putting extra load on a production server.
- To set up identical data for a table on a new slave server.
- To restore a backed-up version of a table or partition after a problem or mistake.
- As a faster way of moving data around than importing the results of a mysqldump command. The data is available immediately, rather than having to be re-inserted and the indexes rebuilt.
- To move a file-per-table tablespace to a server with storage medium that better suits system requirements. For example, you may want to have busy tables on an SSD device, or large tables on a high-capacity HDD device.

Tablespace Copying Limitations and Usage Notes (Transportable Tablespaces)

- The tablespace copy procedure is only possible when innodb_file_per_table is set to ON, which is the default setting as of MySQL 5.6.6. Tables residing in the shared system tablespace cannot be quiesced.
- When a table is quiesced, only read-only transactions are allowed on the affected table.
- When importing a tablespace, the page size must match the page size of the importing instance.
- Prior to MySQL 5.7.4, DISCARD TABLESPACE is not supported for partitioned tables meaning that transportable tablespaces is also unsupported. If you run ALTER TABLE ... DISCARD TABLESPACE on a partitioned table, the following error is returned: ERROR 1031 (HY000): Table storage engine for 'part' doesn't have this option. As of MySQL 5.7.4, ALTER TABLE ... DISCARD TABLESPACE is supported for partitioned InnoDB tables, and ALTER TABLE ... DISCARD PARTITION ... TABLESPACE is supported for InnoDB table partitions.
- DISCARD TABLESPACE is not supported for tablespaces with a parent-child (primary key-foreign key) relationship when foreign_key_checks is set to 1. Before discarding a tablespace for parent-child tables, set foreign_key_checks=0. Partitioned InnoDB tables do not support foreign keys.
- ALTER TABLE ... IMPORT TABLESPACE does not enforce foreign key constraints on imported data. If there are foreign key constraints between tables, all tables should be exported at the same (logical) point in time. Partitioned InnoDB tables do not support foreign keys.
- ALTER TABLE ... IMPORT TABLESPACE and ALTER TABLE ... IMPORT PARTITION ... TABLESPACE do not require a .cfg metadata file to import a tablespace. However, metadata checks are not performed when importing without a .cfg file, and a warning similar to the following will be issued:

```
Message: InnoDB: IO Read error: (2, No such file or directory) Error opening '.\
test\t.cfg', will attempt to import without schema verification
1 row in set (0.00 sec)
```

The ability to import without a .cfg file may be more convenient when no schema mismatches are expected. Additionally, the ability to import without a .cfg file could be useful in crash recovery scenarios in which metadata cannot be collected from an .ibd file.

- Due to a .cfg metadata file limitation, schema mismatches are not reported for partition type or partition definition differences when importing tablespace files for partitioned tables. Column differences are reported.
- When running ALTER TABLE ... DISCARD PARTITION ... TABLESPACE and ALTER TABLE ... IMPORT PARTITION ... TABLESPACE on subpartitioned tables, both partition and subpartition table names are allowed. When a partition name is specified, subpartitions of that partition are included in the operation.
- In MySQL 5.6 or later, importing a tablespace file from another server works if both servers have GA (General Availability) status and their versions are within the same series. Otherwise, the file must have been created on the server into which it is imported.
- In replication scenarios, innodb_file_per_table must be set to ON on both the master and slave.
- On Windows, InnoDB stores database, tablespace, and table names internally in lowercase. To avoid import problems on case-sensitive operating systems such as Linux and UNIX, create all databases, tablespaces, and tables using lowercase names. A convenient way to accomplish this is to add the following line to the [mysqld] section of your my.cnf or my.ini file before creating databases, tablespaces, or tables:

  ```
  [mysqld]
  lower_case_table_names=1
  ```

Transportable Tablespace Examples

Example 1: Copying a Regular InnoDB Table From One Server To Another

This procedure demonstrates how to copy a regular InnoDB table from a running MySQL server instance to another running instance. The same procedure with minor adjustments can be used to perform a full table restore on the same instance.

1. On the source server, create a table if one does not exist:

   ```
   mysql> use test;
   mysql> CREATE TABLE t(c1 INT) engine=InnoDB;
   ```

2. On the destination server, create a table if one does not exist:

   ```
   mysql> use test;
   mysql> CREATE TABLE t(c1 INT) engine=InnoDB;
   ```

3. On the destination server, discard the existing tablespace. (Before a tablespace can be imported, InnoDB must discard the tablespace that is attached to the receiving table.)

   ```
   mysql> ALTER TABLE t DISCARD TABLESPACE;
   ```

4. On the source server, run FLUSH TABLES ... FOR EXPORT to quiesce the table and create the .cfg metadata file:

   ```
   mysql> use test;
   ```

```
mysql> FLUSH TABLES t FOR EXPORT;
```

The metadata (`.cfg`) is created in the `InnoDB` data directory.

> **Note**
>
> `FLUSH TABLES ... FOR EXPORT` is available as of MySQL 5.6.6. The statement ensures that changes to the named table have been flushed to disk so that a binary table copy can be made while the server is running. When `FLUSH TABLES ... FOR EXPORT` is run, `InnoDB` produces a `.cfg` file in the same database directory as the table. The `.cfg` file contains metadata used for schema verification when importing the tablespace file.

5. Copy the `.ibd` file and `.cfg` metadata file from the source server to the destination server. For example:

```
shell> scp /path/to/datadir/test/t.{ibd,cfg} destination-server:/path/to/datadir/test
```

> **Note**
>
> The `.ibd` file and `.cfg` file must be copied before releasing the shared locks, as described in the next step.

6. On the source server, use `UNLOCK TABLES` to release the locks acquired by `FLUSH TABLES ... FOR EXPORT`:

```
mysql> use test;
mysql> UNLOCK TABLES;
```

7. On the destination server, import the tablespace:

```
mysql> use test;
mysql> ALTER TABLE t IMPORT TABLESPACE;
```

> **Note**
>
> The `ALTER TABLE ... IMPORT TABLESPACE` feature does not enforce foreign key constraints on imported data. If there are foreign key constraints between tables, all tables should be exported at the same (logical) point in time. In this case you would stop updating the tables, commit all transactions, acquire shared locks on the tables, and then perform the export operation.

Example 2: Copying an `InnoDB` Partitioned Table From One Server To Another

This procedure demonstrates how to copy a partitioned `InnoDB` table from a running MySQL server instance to another running instance. The same procedure with minor adjustments can be used to perform a full restore of a partitioned `InnoDB` table on the same instance.

1. On the source server, create a partitioned table if one does not exist. In the following example, a table with three partitions (p0, p1, p2) is created:

```
mysql> use test;
mysql> CREATE TABLE t1 (i int) ENGINE = InnoDB PARTITION BY KEY (i) PARTITIONS 3;
```

In the *`/datadir/test`* directory, you will see a separate tablespace (`.ibd`) file for each of the three partitions.

```
mysql> \! ls /path/to/datadir/test/
db.opt  t1.frm  t1.par  t1#P#p0.ibd  t1#P#p1.ibd  t1#P#p2.ibd
```

2. On the destination server, create the same partitioned table:

   ```
   mysql> use test;
   mysql> CREATE TABLE t1 (i int) ENGINE = InnoDB PARTITION BY KEY (i) PARTITIONS 3;
   ```

   In the `/datadir/test` directory, you will see a separate tablespace (`.ibd`) file for each of the three partitions.

   ```
   mysql> \! ls /path/to/datadir/test/
   db.opt  t1.frm  t1.par  t1#P#p0.ibd  t1#P#p1.ibd  t1#P#p2.ibd
   ```

3. On the destination server, discard the tablespace for the partitioned table. (Before the tablespace can be imported on the destination server, the tablespace that is attached to the receiving table must be discarded.)

   ```
   mysql> ALTER TABLE t1 DISCARD TABLESPACE;
   ```

   The three `.ibd` files that make up the tablespace for the partitioned table are discarded from the `/datadir/test` directory, leaving the following files:

   ```
   mysql> \! ls /path/to/datadir/test/
   db.opt  t1.frm  t1.par
   ```

4. On the source server, run `FLUSH TABLES ... FOR EXPORT` to quiesce the partitioned table and create the `.cfg` metadata files:

   ```
   mysql> use test;
   mysql> FLUSH TABLES t1 FOR EXPORT;
   ```

   Metadata (`.cfg`) files, one for each tablespace (`.ibd`) file, are created in the `/datadir/test` directory on the source server:

   ```
   mysql> \! ls /path/to/datadir/test/
   db.opt  t1.par       t1#P#p0.ibd  t1#P#p1.ibd  t1#P#p2.ibd
   t1.frm  t1#P#p0.cfg  t1#P#p1.cfg  t1#P#p2.cfg
   ```

   Note

   `FLUSH TABLES ... FOR EXPORT` statement ensures that changes to the named table have been flushed to disk so that binary table copy can be made while the server is running. When `FLUSH TABLES ... FOR EXPORT` is run, `InnoDB` produces a `.cfg` metadata file for the table's tablespace files in the same database directory as the table. The `.cfg` files contain metadata used for schema verification when importing tablespace files. `FLUSH TABLES ... FOR EXPORT` can only be run on the table, not on individual table partitions.

5. Copy the `.ibd` and `.cfg` files from the source server database directory to the destination server database directory. For example:

```
shell> scp /path/to/datadir/test/t1*.{ibd,cfg} destination-server:/path/to/datadir/test
```

> **Note**
>
> The .ibd and .cfg files must be copied before releasing the shared locks, as described in the next step.

6. On the source server, use UNLOCK TABLES to release the locks acquired by FLUSH TABLES ... FOR EXPORT:

   ```
   mysql> use test;
   mysql> UNLOCK TABLES;
   ```

7. On the destination server, import the tablespace for the partitioned table:

   ```
   mysql> use test;
   mysql> ALTER TABLE t1 IMPORT TABLESPACE;
   ```

Example 3: Copying InnoDB Table Partitions From One Server To Another

This procedure demonstrates how to copy InnoDB table partitions from a running MySQL server instance to another running instance. The same procedure with minor adjustments can be used to perform a restore of InnoDB table partitions on the same instance. In the following example, a partitioned table with four partitions (p0, p1, p2, p3) is created on the source server. Two of the partitions (p2 and p3) are copied to the destination server.

1. On the source server, create a partitioned table if one does not exist. In the following example, a table with four partitions (p0, p1, p2, p3) is created:

   ```
   mysql> use test;
   mysql> CREATE TABLE t1 (i int) ENGINE = InnoDB PARTITION BY KEY (i) PARTITIONS 4;
   ```

   In the /datadir/test directory, you will see a separate tablespace (.ibd) file for each of the four partitions.

   ```
   mysql> \! ls /path/to/datadir/test/
   db.opt  t1.frm  t1.par  t1#P#p0.ibd  t1#P#p1.ibd  t1#P#p2.ibd t1#P#p3.ibd
   ```

2. On the destination server, create the same partitioned table:

   ```
   mysql> use test;
   mysql> CREATE TABLE t1 (i int) ENGINE = InnoDB PARTITION BY KEY (i) PARTITIONS 4;
   ```

   In the /datadir/test directory, you will see a separate tablespace (.ibd) file for each of the four partitions.

   ```
   mysql> \! ls /path/to/datadir/test/
   db.opt  t1.frm  t1.par  t1#P#p0.ibd  t1#P#p1.ibd  t1#P#p2.ibd t1#P#p3.ibd
   ```

3. On the destination server, discard the tablespace partitions that you plan to import from the source server. (Before tablespace partitions can be imported on the destination server, the corresponding partitions that are attached to the receiving table must be discarded.)

   ```
   mysql> ALTER TABLE t1 DISCARD PARTITION p2, p3 TABLESPACE;
   ```

The .ibd files for the two discarded partitions are removed from the */datadir/*test directory on the destination server, leaving the following files:

```
mysql> \! ls /path/to/datadir/test/
db.opt  t1.frm  t1.par t1#P#p0.ibd  t1#P#p1.ibd
```

Note

When ALTER TABLE ... DISCARD PARTITION ... TABLESPACE is run on subpartitioned tables, both partition and subpartition table names are allowed. When a partition name is specified, subpartitions of that partition are included in the operation.

4. On the source server, run FLUSH TABLES ... FOR EXPORT to quiesce the partitioned table and create the .cfg metadata files.

   ```
   mysql> use test;
   mysql> FLUSH TABLES t1 FOR EXPORT;
   ```

   The metadata files (.cfg files) are created in the */datadir/*test directory on the source server. There is a .cfg file for each tablespace (.ibd) file.

   ```
   mysql> \! ls /path/to/datadir/test/
   db.opt  t1.par       t1#P#p0.ibd  t1#P#p1.ibd  t1#P#p2.ibd t1#P#p3.ibd
   t1.frm  t1#P#p0.cfg  t1#P#p1.cfg  t1#P#p2.cfg t1#P#p3.cfg
   ```

   Note

   FLUSH TABLES ... FOR EXPORT statement ensures that changes to the named table have been flushed to disk so that binary table copy can be made while the server is running. When FLUSH TABLES ... FOR EXPORT is run, InnoDB produces a .cfg metadata file for the table's tablespace files in the same database directory as the table. The .cfg files contain metadata used for schema verification when importing tablespace files. FLUSH TABLES ... FOR EXPORT can only be run on the table, not on individual table partitions.

5. Copy the .ibd and .cfg files from the source server database directory to the destination server database directory. In this example, only the .ibd and .cfg files for partition 2 (p2) and partition 3 (p3) are copied to the data directory on the destination server. Partition 0 (p0) and partition 1 (p1) remain on the source server.

   ```
   shell> scp t1#P#p2.ibd  t1#P#p2.cfg t1#P#p3.ibd t1#P#p3.cfg destination-server:/path/to/datadir/test
   ```

   Note

   The .ibd files and .cfg files must be copied before releasing the shared locks, as described in the next step.

6. On the source server, use UNLOCK TABLES to release the locks acquired by FLUSH TABLES ... FOR EXPORT:

   ```
   mysql> use test;
   mysql> UNLOCK TABLES;
   ```

7. On the destination server, import the tablespace partitions (p2 and p3):

```
mysql> use test;
mysql> ALTER TABLE t1 IMPORT PARTITION p2, p3 TABLESPACE;
```

> **Note**
>
> When ALTER TABLE ... IMPORT PARTITION ... TABLESPACE is run on subpartitioned tables, both partition and subpartition table names are allowed. When a partition name is specified, subpartitions of that partition are included in the operation.

Tablespace Copying Internals (Transportable Tablespaces)

The following information describes internals and error log messaging for the transportable tablespaces copy procedure for a regular InnoDB table.

When ALTER TABLE ... DISCARD TABLESPACE is run on the destination instance:

- The table is locked in X mode.
- The tablespace is detached from the table.

When FLUSH TABLES ... FOR EXPORT is run on the source instance:

- The table being flushed for export is locked in shared mode.
- The purge coordinator thread is stopped.
- Dirty pages are synchronized to disk.
- Table metadata is written to the binary .cfg file.

Expected error log messages for this operation:

```
2013-09-24T13:10:19.903526Z 2 [Note] InnoDB: Sync to disk of '"test"."t"' started.
2013-09-24T13:10:19.903586Z 2 [Note] InnoDB: Stopping purge
2013-09-24T13:10:19.903725Z 2 [Note] InnoDB: Writing table metadata to './test/t.cfg'
2013-09-24T13:10:19.904014Z 2 [Note] InnoDB: Table '"test"."t"' flushed to disk
```

When UNLOCK TABLES is run on the source instance:

- The binary .cfg file is deleted.
- The shared lock on the table or tables being imported is released and the purge coordinator thread is restarted.

Expected error log messages for this operation:

```
2013-09-24T13:10:21.181104Z 2 [Note] InnoDB: Deleting the meta-data file './test/t.cfg'
2013-09-24T13:10:21.181180Z 2 [Note] InnoDB: Resuming purge
```

When ALTER TABLE ... IMPORT TABLESPACE is run on the destination instance, the import algorithm performs the following operations for each tablespace being imported:

- Each tablespace page is checked for corruption.

- The space ID and log sequence numbers (LSNs) on each page are updated
- Flags are validated and LSN updated for the header page.
- Btree pages are updated.
- The page state is set to dirty so that it will be written to disk.

Expected error log messages for this operation:

```
2013-07-18 15:15:01 34960 [Note] InnoDB: Importing tablespace for table 'test/t' that was exported from hos
2013-07-18 15:15:01 34960 [Note] InnoDB: Phase I - Update all pages
2013-07-18 15:15:01 34960 [Note] InnoDB: Sync to disk
2013-07-18 15:15:01 34960 [Note] InnoDB: Sync to disk - done!
2013-07-18 15:15:01 34960 [Note] InnoDB: Phase III - Flush changes to disk
2013-07-18 15:15:01 34960 [Note] InnoDB: Phase IV - Flush complete
```

Note

You may also receive a warning that a tablespace is discarded (if you discarded the tablespace for the destination table) and a message stating that statistics could not be calculated due to a missing .ibd file:

```
2013-07-18 15:14:38 34960 [Warning] InnoDB: Table "test"."t" tablespace is set as discard
2013-07-18 15:14:38 7f34d9a37700 InnoDB: cannot calculate statistics for table "test"."t"
http://dev.mysql.com/doc/refman/5.7/en/innodb-troubleshooting.html
```

14.2.5.6 Moving the Undo Log out of the System Tablespace

Although tablespace management typically involves files holding tables and indexes, you can also divide the undo log into separate undo tablespace files. This layout is different from the default configuration where the undo log is part of the system tablespace. See Separate Tablespaces for InnoDB Undo Logs for details.

14.2.5.7 Changing the Number or Size of InnoDB Log Files and Resizing the InnoDB Tablespace

This section describes how to change the number or size of InnoDB redo log files and how to increase or decrease InnoDB system tablespace size.

Changing the Number or Size of InnoDB Log Files

To change the number or size of InnoDB redo log files, perform the following steps:

1. Stop the MySQL server and make sure that it shuts down without errors.
2. Edit my.cnf to change the log file configuration. To change the log file size, configure innodb_log_file_size. To increase the number of log files, configure innodb_log_files_in_group.
3. Start the MySQL server again.

If InnoDB detects that the innodb_log_file_size differs from the redo log file size, it will write a log checkpoint, close and remove the old log files, create new log files at the requested size, and open the new log files.

Increasing the Size of the InnoDB Tablespace

The easiest way to increase the size of the InnoDB system tablespace is to configure it from the beginning to be auto-extending. Specify the autoextend attribute for the last data file in the tablespace definition. Then InnoDB increases the size of that file automatically in 8MB increments when it runs out of space. The increment size can be changed by setting the value of the innodb_autoextend_increment system variable, which is measured in megabytes.

You can expand the system tablespace by a defined amount by adding another data file:

1. Shut down the MySQL server.

2. If the previous last data file is defined with the keyword autoextend, change its definition to use a fixed size, based on how large it has actually grown. Check the size of the data file, round it down to the closest multiple of 1024 × 1024 bytes (= 1MB), and specify this rounded size explicitly in innodb_data_file_path.

3. Add a new data file to the end of innodb_data_file_path, optionally making that file auto-extending. Only the last data file in the innodb_data_file_path can be specified as auto-extending.

4. Start the MySQL server again.

For example, this tablespace has just one auto-extending data file ibdata1:

```
innodb_data_home_dir =
innodb_data_file_path = /ibdata/ibdata1:10M:autoextend
```

Suppose that this data file, over time, has grown to 988MB. Here is the configuration line after modifying the original data file to use a fixed size and adding a new auto-extending data file:

```
innodb_data_home_dir =
innodb_data_file_path = /ibdata/ibdata1:988M;/disk2/ibdata2:50M:autoextend
```

When you add a new data file to the system tablespace configuration, make sure that the filename does not refer to an existing file. InnoDB creates and initializes the file when you restart the server.

Decreasing the Size of the InnoDB Tablespace

Currently, you cannot remove a data file from the system tablespace. To decrease the system tablespace size, use this procedure:

1. Use mysqldump to dump all your InnoDB tables, including InnoDB tables located in the MySQL database. As of 5.6, there are five InnoDB tables included in the MySQL database:

   ```
   mysql> select table_name from information_schema.tables where table_schema='mysql' and engine='InnoDB';
   +----------------------+
   | table_name           |
   +----------------------+
   | innodb_index_stats   |
   | innodb_table_stats   |
   | slave_master_info    |
   | slave_relay_log_info |
   | slave_worker_info    |
   +----------------------+
   5 rows in set (0.00 sec)
   ```

2. Stop the server.
3. Remove all the existing tablespace files (*.ibd), including the ibdata and ib_log files. Do not forget to remove *.ibd files for tables located in the MySQL database.
4. Remove any .frm files for InnoDB tables.
5. Configure a new tablespace.
6. Restart the server.
7. Import the dump files.

> **Note**
>
> If your databases only use the InnoDB engine, it may be simpler to dump **all** databases, stop the server, remove all databases and InnoDB log files, restart the server, and import the dump files.

14.2.5.8 Using Raw Disk Partitions for the Shared Tablespace

You can use raw disk partitions as data files in the InnoDB system tablespace. This technique enables nonbuffered I/O on Windows and on some Linux and Unix systems without file system overhead. Perform tests with and without raw partitions to verify whether this change actually improves performance on your system.

When you use a raw disk partition, ensure that the user ID that runs the MySQL server has read and write privileges for that partition. For example, if you run the server as the mysql user, the partition must be readable and writeable by mysql. If you run the server with the --memlock option, the server must be run as root, so the partition must be readable and writeable by root.

The procedures described below involve option file modification. For additional information, see Section 4.2.3.3, “Using Option Files”.

Allocating a Raw Disk Partition on Linux and Unix Systems

1. When you create a new data file, specify the keyword newraw immediately after the data file size for the innodb_data_file_path option. The partition must be at least as large as the size that you specify. Note that 1MB in InnoDB is 1024 × 1024 bytes, whereas 1MB in disk specifications usually means 1,000,000 bytes.

```
[mysqld]
innodb_data_home_dir=
innodb_data_file_path=/dev/hdd1:3Gnewraw;/dev/hdd2:2Gnewraw
```

2. Restart the server. InnoDB notices the newraw keyword and initializes the new partition. However, do not create or change any InnoDB tables yet. Otherwise, when you next restart the server, InnoDB reinitializes the partition and your changes are lost. (As a safety measure InnoDB prevents users from modifying data when any partition with newraw is specified.)
3. After InnoDB has initialized the new partition, stop the server, change newraw in the data file specification to raw:

```
[mysqld]
innodb_data_home_dir=
innodb_data_file_path=/dev/hdd1:3Graw;/dev/hdd2:2Graw
```

4. Restart the server. InnoDB now permits changes to be made.

Allocating a Raw Disk Partition on Windows

On Windows systems, the same steps and accompanying guidelines described for Linux and Unix systems apply except that the innodb_data_file_path setting differs slightly on Windows.

1. When you create a new data file, specify the keyword newraw immediately after the data file size for the innodb_data_file_path option:

```
[mysqld]
innodb_data_home_dir=
innodb_data_file_path=//./D::10Gnewraw
```

   The //./ corresponds to the Windows syntax of \\.\ for accessing physical drives. In the example above, D: is the drive letter of the partition.

2. Restart the server. InnoDB notices the newraw keyword and initializes the new partition.

3. After InnoDB has initialized the new partition, stop the server, change newraw in the data file specification to raw:

```
[mysqld]
innodb_data_home_dir=
innodb_data_file_path=//./D::10Graw
```

4. Restart the server. InnoDB now permits changes to be made.

14.2.6 InnoDB Table Management

14.2.6.1 Creating InnoDB Tables

To create an InnoDB table, use the CREATE TABLE statement without any special clauses. Formerly, you needed the ENGINE=InnoDB clause, but not anymore now that InnoDB is the default storage engine. (You might still use that clause if you plan to use mysqldump or replication to replay the CREATE TABLE statement on a server running MySQL 5.1 or earlier, where the default storage engine is MyISAM.)

```
-- Default storage engine = InnoDB.
CREATE TABLE t1 (a INT, b CHAR (20), PRIMARY KEY (a));
-- Backwards-compatible with older MySQL.
CREATE TABLE t2 (a INT, b CHAR (20), PRIMARY KEY (a)) ENGINE=InnoDB;
```

Depending on the innodb_file_per_table setting, InnoDB creates each table and associated primary key index either in the system tablespace, or in a separate tablespace (represented by a .ibd file) for each table. MySQL creates t1.frm and t2.frm files in the test directory under the MySQL database directory. Internally, InnoDB adds an entry for the table to its own data dictionary. The entry includes the database name. For example, if test is the database in which the t1 table is created, the entry is for 'test/t1'. This means you can create a table of the same name t1 in some other database, and the table names do not collide inside InnoDB.

To see the properties of these tables, issue a SHOW TABLE STATUS statement:

```
SHOW TABLE STATUS FROM test LIKE 't%' \G;
```

In the status output, you see the row format property of these first tables is Compact. Although that setting is fine for basic experimentation, to take advantage of the most powerful InnoDB performance features, you will quickly graduate to using other row formats such as Dynamic and Compressed. Using those values requires a little bit of setup first:

```
set global innodb_file_per_table=1;
set global innodb_file_format=barracuda;
CREATE TABLE t3 (a INT, b CHAR (20), PRIMARY KEY (a)) row_format=dynamic;
CREATE TABLE t4 (a INT, b CHAR (20), PRIMARY KEY (a)) row_format=compressed;
```

Always set up a primary key for each InnoDB table, specifying the column or columns that:

- Are referenced by the most important queries.
- Are never left blank.
- Never have duplicate values.
- Rarely if ever change value once inserted.

For example, in a table containing information about people, you would not create a primary key on (firstname, lastname) because more than one person can have the same name, some people have blank last names, and sometimes people change their names. With so many constraints, often there is not an obvious set of columns to use as a primary key, so you create a new column with a numeric ID to serve as all or part of the primary key. You can declare an auto-increment column so that ascending values are filled in automatically as rows are inserted:

```
-- The value of ID can act like a pointer between related items in different tables.
CREATE TABLE t5 (id INT AUTO_INCREMENT, b CHAR (20), PRIMARY KEY (id));
-- The primary key can consist of more than one column. Any autoinc column must come first.
CREATE TABLE t6 (id INT AUTO_INCREMENT, a INT, b CHAR (20), PRIMARY KEY (id,a));
```

Although the table *works* correctly without you defining a primary key, the primary key is involved with many aspects of performance and is a crucial design aspect for any large or frequently used table. Make a habit of always specifying one in the CREATE TABLE statement. (If you create the table, load data, and then do ALTER TABLE to add a primary key later, that operation is much slower than defining the primary key when creating the table.)

14.2.6.2 Moving or Copying InnoDB Tables to Another Machine

This section describes techniques for moving or copying some or all InnoDB tables to a different server. For example, you might move an entire MySQL instance to a larger, faster server; you might clone an entire MySQL instance to a new replication slave server; you might copy individual tables to another server to develop and test an application, or to a data warehouse server to produce reports.

Techniques for moving or copying InnoDB tables include:

- Transportable Tablespaces
- MySQL Enterprise Backup
- Copying Data Files (Cold Backup Method)
- Export and Import (mysqldump)

Using Lowercase Names for Cross-Platform Moving or Copying

On Windows, InnoDB always stores database and table names internally in lowercase. To move databases in a binary format from Unix to Windows or from Windows to Unix, create all databases and tables using lowercase names. A convenient way to accomplish this is to add the following line to the [mysqld] section of your my.cnf or my.ini file before creating any databases or tables:

```
[mysqld]
```

```
lower_case_table_names=1
```

Transportable Tablespaces

Introduced in MySQL 5.6.6, the transportable tablespaces feature uses FLUSH TABLES ... FOR EXPORT to ready InnoDB tables for copying from one server instance to another. To use this feature, InnoDB tables must be created with innodb_file_per_table set to ON so that each InnoDB table has its own tablespace. For usage information, see Section 14.2.5.5, “Copying Tablespaces to Another Server (Transportable Tablespaces)”.

MySQL Enterprise Backup

The MySQL Enterprise Backup product lets you back up a running MySQL database, including InnoDB and MyISAM tables, with minimal disruption to operations while producing a consistent snapshot of the database. When MySQL Enterprise Backup is copying InnoDB tables, reads and writes to both InnoDB and MyISAM tables can continue. During the copying of MyISAM and other non-InnoDB tables, reads (but not writes) to those tables are permitted. In addition, MySQL Enterprise Backup can create compressed backup files, and back up subsets of InnoDB tables. In conjunction with the MySQL binary log, you can perform point-in-time recovery. MySQL Enterprise Backup is included as part of the MySQL Enterprise subscription.

For more details about MySQL Enterprise Backup, see Section 23.2, “MySQL Enterprise Backup”.

Copying Data Files (Cold Backup Method)

You can move an InnoDB database simply by copying all the relevant files listed under "Cold Backups" in Section 14.2.14, “InnoDB Backup and Recovery”.

Like MyISAM data files, InnoDB data and log files are binary-compatible on all platforms having the same floating-point number format. If the floating-point formats differ but you have not used FLOAT or DOUBLE data types in your tables, then the procedure is the same: simply copy the relevant files.

Portability Considerations for .ibd Files

When you move or copy .ibd files, the database directory name must be the same on the source and destination systems. The table definition stored in the InnoDB shared tablespace includes the database name. The transaction IDs and log sequence numbers stored in the tablespace files also differ between databases.

To move an .ibd file and the associated table from one database to another, use a RENAME TABLE statement:

```
RENAME TABLE db1.tbl_name TO db2.tbl_name;
```

If you have a “clean” backup of an .ibd file, you can restore it to the MySQL installation from which it originated as follows:

1. The table must not have been dropped or truncated since you copied the .ibd file, because doing so changes the table ID stored inside the tablespace.
2. Issue this ALTER TABLE statement to delete the current .ibd file:

   ```
   ALTER TABLE tbl_name DISCARD TABLESPACE;
   ```

3. Copy the backup .ibd file to the proper database directory.
4. Issue this ALTER TABLE statement to tell InnoDB to use the new .ibd file for the table:

```
ALTER TABLE tbl_name IMPORT TABLESPACE;
```

> **Note**
>
> The ALTER TABLE ... IMPORT TABLESPACE feature does not enforce foreign key constraints on imported data.

In this context, a “clean” .ibd file backup is one for which the following requirements are satisfied:

- There are no uncommitted modifications by transactions in the .ibd file.
- There are no unmerged insert buffer entries in the .ibd file.
- Purge has removed all delete-marked index records from the .ibd file.
- mysqld has flushed all modified pages of the .ibd file from the buffer pool to the file.

You can make a clean backup .ibd file using the following method:

1. Stop all activity from the mysqld server and commit all transactions.
2. Wait until SHOW ENGINE INNODB STATUS shows that there are no active transactions in the database, and the main thread status of InnoDB is Waiting for server activity. Then you can make a copy of the .ibd file.

Another method for making a clean copy of an .ibd file is to use the MySQL Enterprise Backup product:

1. Use MySQL Enterprise Backup to back up the InnoDB installation.
2. Start a second mysqld server on the backup and let it clean up the .ibd files in the backup.

Export and Import (mysqldump)

You can use mysqldump to dump your tables on one machine and then import the dump files on the other machine. Using this method, it does not matter whether the formats differ or if your tables contain floating-point data.

One way to increase the performance of this method is to switch off autocommit mode when importing data, assuming that the tablespace has enough space for the big rollback segment that the import transactions generate. Do the commit only after importing a whole table or a segment of a table.

14.2.6.3 Grouping DML Operations with Transactions

By default, connection to the MySQL server begins with autocommit mode enabled, which automatically commits every SQL statement as you execute it. This mode of operation might be unfamiliar if you have experience with other database systems, where it is standard practice to issue a sequence of DML statements and commit them or roll them back all together.

To use multiple-statement transactions, switch autocommit off with the SQL statement SET autocommit = 0 and end each transaction with COMMIT or ROLLBACK as appropriate. To leave autocommit on, begin each transaction with START TRANSACTION and end it with COMMIT or ROLLBACK. The following example shows two transactions. The first is committed; the second is rolled back.

```
shell> mysql test

mysql> CREATE TABLE customer (a INT, b CHAR (20), INDEX (a));
Query OK, 0 rows affected (0.00 sec)
mysql> -- Do a transaction with autocommit turned on.
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)
mysql> INSERT INTO customer VALUES (10, 'Heikki');
Query OK, 1 row affected (0.00 sec)
mysql> COMMIT;
Query OK, 0 rows affected (0.00 sec)
mysql> -- Do another transaction with autocommit turned off.
mysql> SET autocommit=0;
Query OK, 0 rows affected (0.00 sec)
mysql> INSERT INTO customer VALUES (15, 'John');
Query OK, 1 row affected (0.00 sec)
mysql> INSERT INTO customer VALUES (20, 'Paul');
Query OK, 1 row affected (0.00 sec)
mysql> DELETE FROM customer WHERE b = 'Heikki';
Query OK, 1 row affected (0.00 sec)
mysql> -- Now we undo those last 2 inserts and the delete.
mysql> ROLLBACK;
Query OK, 0 rows affected (0.00 sec)
mysql> SELECT * FROM customer;
+------+--------+
| a    | b      |
+------+--------+
|   10 | Heikki |
+------+--------+
1 row in set (0.00 sec)
mysql>
```

Transactions in Client-Side Languages

In APIs such as PHP, Perl DBI, JDBC, ODBC, or the standard C call interface of MySQL, you can send transaction control statements such as COMMIT to the MySQL server as strings just like any other SQL statements such as SELECT or INSERT. Some APIs also offer separate special transaction commit and rollback functions or methods.

14.2.6.4 Converting Tables from MyISAM to InnoDB

If you have existing tables, and applications that use them, that you want to convert to InnoDB for better reliability and scalability, use the following guidelines and tips. This section assumes most such tables were originally MyISAM, which was formerly the default.

Reduce Memory Usage for MyISAM, Increase Memory Usage for InnoDB

As you transition away from MyISAM tables, lower the value of the key_buffer_size configuration option to free memory no longer needed for caching results. Increase the value of the innodb_buffer_pool_size configuration option, which performs a similar role of allocating cache memory for InnoDB tables. The InnoDB buffer pool caches both table data and index data, so it does double duty in speeding up lookups for queries and keeping query results in memory for reuse.

- Allocate as much memory to this option as you can afford, often up to 80% of physical memory on the server.

- If the operating system runs short of memory for other processes and begins to swap, reduce the innodb_buffer_pool_size value. Swapping is such an expensive operation that it drastically reduces the benefit of the cache memory.

- If the innodb_buffer_pool_size value is several gigabytes or higher, consider increasing the values of innodb_buffer_pool_instances. Doing so helps on busy servers where many connections are reading data into the cache at the same time.

- On a busy server, run benchmarks with the Query Cache turned off. The InnoDB buffer pool provides similar benefits, so the Query Cache might be tying up memory unnecessarily.

Watch Out for Too-Long Or Too-Short Transactions

Because MyISAM tables do not support transactions, you might not have paid much attention to the autocommit configuration option and the COMMIT and ROLLBACK statements. These keywords are important to allow multiple sessions to read and write InnoDB tables concurrently, providing substantial scalability benefits in write-heavy workloads.

While a transaction is open, the system keeps a snapshot of the data as seen at the beginning of the transaction, which can cause substantial overhead if the system inserts, updates, and deletes millions of rows while a stray transaction keeps running. Thus, take care to avoid transactions that run for too long:

- If you are using a mysql session for interactive experiments, always COMMIT (to finalize the changes) or ROLLBACK (to undo the changes) when finished. Close down interactive sessions rather than leaving them open for long periods, to avoid keeping transactions open for long periods by accident.

- Make sure that any error handlers in your application also ROLLBACK incomplete changes or COMMIT completed changes.

- ROLLBACK is a relatively expensive operation, because INSERT, UPDATE, and DELETE operations are written to InnoDB tables prior to the COMMIT, with the expectation that most changes will be committed successfully and rollbacks will be rare. When experimenting with large volumes of data, avoid making changes to large numbers of rows and then rolling back those changes.

- When loading large volumes of data with a sequence of INSERT statements, periodically COMMIT the results to avoid having transactions that last for hours. In typical load operations for data warehousing, if something goes wrong, you TRUNCATE TABLE and start over from the beginning rather than doing a ROLLBACK.

The preceding tips save memory and disk space that can be wasted during too-long transactions. When transactions are shorter than they should be, the problem is excessive I/O. With each COMMIT, MySQL makes sure each change is safely recorded to disk, which involves some I/O.

- For most operations on InnoDB tables, you should use the setting autocommit=0. From an efficiency perspective, this avoids unnecessary I/O when you issue large numbers of consecutive INSERT, UPDATE, or DELETE statements. From a safety perspective, this allows you to issue a ROLLBACK statement to recover lost or garbled data if you make a mistake on the mysql command line, or in an exception handler in your application.

- The time when autocommit=1 is suitable for InnoDB tables is when running a sequence of queries for generating reports or analyzing statistics. In this situation, there is no I/O penalty related to COMMIT or ROLLBACK, and InnoDB can automatically optimize the read-only workload.

- If you make a series of related changes, finalize all those changes at once with a single COMMIT at the end. For example, if you insert related pieces of information into several tables, do a single COMMIT after making all the changes. Or if you run many consecutive INSERT statements, do a single COMMIT after all the data is loaded; if you are doing millions of INSERT statements, perhaps split up the huge transaction by issuing a COMMIT every ten thousand or hundred thousand records, so the transaction does not grow too large.

- Remember that even a SELECT statement opens a transaction, so after running some report or debugging queries in an interactive mysql session, either issue a COMMIT or close the mysql session.

Don't Worry Too Much About Deadlocks

You might see warning messages referring to “deadlocks” in the MySQL error log, or the output of SHOW ENGINE INNODB STATUS. Despite the scary-sounding name, a deadlock is not a serious issue for

InnoDB tables, and often does not require any corrective action. When two transactions start modifying multiple tables, accessing the tables in a different order, they can reach a state where each transaction is waiting for the other and neither can proceed. MySQL immediately detects this condition and cancels (rolls back) the “smaller” transaction, allowing the other to proceed.

Your applications do need error-handling logic to restart a transaction that is forcibly cancelled like this. When you re-issue the same SQL statements as before, the original timing issue no longer applies: either the other transaction has already finished and yours can proceed, or the other transaction is still in progress and your transaction waits until it finishes.

If deadlock warnings occur constantly, you might review the application code to reorder the SQL operations in a consistent way, or to shorten the transactions. You can test with the innodb_print_all_deadlocks option enabled to see all deadlock warnings in the MySQL error log, rather than only the last warning in the SHOW ENGINE INNODB STATUS output.

Plan the Storage Layout

To get the best performance from InnoDB tables, you can adjust a number of parameters related to storage layout.

When you convert MyISAM tables that are large, frequently accessed, and hold vital data, investigate and consider the innodb_file_per_table, innodb_file_format, and innodb_page_size configuration options, and the ROW_FORMAT and KEY_BLOCK_SIZE clauses of the CREATE TABLE statement.

During your initial experiments, the most important setting is innodb_file_per_table. Enabling this option before creating new InnoDB tables ensures that the InnoDB system tablespace files do not allocate disk space permanently for all the InnoDB data. With innodb_file_per_table enabled, DROP TABLE and TRUNCATE TABLE free disk space as you would expect.

Converting an Existing Table

To convert a non-InnoDB table to use InnoDB use ALTER TABLE:

```
ALTER TABLE table_name ENGINE=InnoDB;
```

⚠ **Important**

Do not convert MySQL system tables in the mysql database (such as user or host) to the InnoDB type. This is an unsupported operation. The system tables must always be of the MyISAM type.

Cloning the Structure of a Table

You might make an InnoDB table that is a clone of a MyISAM table, rather than doing the ALTER TABLE conversion, to test the old and new table side-by-side before switching.

Create an empty InnoDB table with identical column and index definitions. Use show create table *table_name*\G to see the full CREATE TABLE statement to use. Change the ENGINE clause to ENGINE=INNODB.

Transferring Existing Data

To transfer a large volume of data into an empty InnoDB table created as shown in the previous section, insert the rows with INSERT INTO *innodb_table* SELECT * FROM *myisam_table* ORDER BY *primary_key_columns*.

You can also create the indexes for the InnoDB table after inserting the data. Historically, creating new secondary indexes was a slow operation for InnoDB, but now you can create the indexes after the data is loaded with relatively little overhead from the index creation step.

If you have UNIQUE constraints on secondary keys, you can speed up a table import by turning off the uniqueness checks temporarily during the import operation:

```
SET unique_checks=0;
... import operation ...
SET unique_checks=1;
```

For big tables, this saves disk I/O because InnoDB can use its insert buffer to write secondary index records as a batch. Be certain that the data contains no duplicate keys. unique_checks permits but does not require storage engines to ignore duplicate keys.

To get better control over the insertion process, you might insert big tables in pieces:

```
INSERT INTO newtable SELECT * FROM oldtable
   WHERE yourkey > something AND yourkey <= somethingelse;
```

After all records have been inserted, you can rename the tables.

During the conversion of big tables, increase the size of the InnoDB buffer pool to reduce disk I/O, to a maximum of 80% of physical memory. You can also increase the sizes of the InnoDB log files.

Storage Requirements

By this point, as already mentioned, you should already have the innodb_file_per_table option enabled, so that if you temporarily make several copies of your data in InnoDB tables, you can recover all that disk space by dropping unneeded tables afterward.

Whether you convert the MyISAM table directly or create a cloned InnoDB table, make sure that you have sufficient disk space to hold both the old and new tables during the process. InnoDB tables require more disk space than MyISAM tables. If an ALTER TABLE operation runs out of space, it starts a rollback, and that can take hours if it is disk-bound. For inserts, InnoDB uses the insert buffer to merge secondary index records to indexes in batches. That saves a lot of disk I/O. For rollback, no such mechanism is used, and the rollback can take 30 times longer than the insertion.

In the case of a runaway rollback, if you do not have valuable data in your database, it may be advisable to kill the database process rather than wait for millions of disk I/O operations to complete. For the complete procedure, see Section 14.2.17.2, “Starting InnoDB on a Corrupted Database”.

Carefully Choose a PRIMARY KEY for Each Table

The PRIMARY KEY clause is a critical factor affecting the performance of MySQL queries and the space usage for tables and indexes. Perhaps you have phoned a financial institution where you are asked for an account number. If you do not have the number, you are asked for a dozen different pieces of information to “uniquely identify” yourself. The primary key is like that unique account number that lets you get straight down to business when querying or modifying the information in a table. Every row in the table must have a primary key value, and no two rows can have the same primary key value.

Here are some guidelines for the primary key, followed by more detailed explanations.

- Declare a PRIMARY KEY for each table. Typically, it is the most important column that you refer to in WHERE clauses when looking up a single row.
- Declare the PRIMARY KEY clause in the original CREATE TABLE statement, rather than adding it later through an ALTER TABLE statement.

- Choose the column and its data type carefully. Prefer numeric columns over character or string ones.
- Consider using an auto-increment column if there is not another stable, unique, non-null, numeric column to use.
- An auto-increment column is also a good choice if there is any doubt whether the value of the primary key column could ever change. Changing the value of a primary key column is an expensive operation, possibly involving rearranging data within the table and within each secondary index.

Consider adding a primary key to any table that does not already have one. Use the smallest practical numeric type based on the maximum projected size of the table. This can make each row slightly more compact, which can yield substantial space savings for large tables. The space savings are multiplied if the table has any secondary indexes, because the primary key value is repeated in each secondary index entry. In addition to reducing data size on disk, a small primary key also lets more data fit into the buffer pool, speeding up all kinds of operations and improving concurrency.

If the table already has a primary key on some longer column, such as a VARCHAR, consider adding a new unsigned AUTO_INCREMENT column and switching the primary key to that, even if that column is not referenced in queries. This design change can produce substantial space savings in the secondary indexes. You can designate the former primary key columns as UNIQUE NOT NULL to enforce the same constraints as the PRIMARY KEY clause, that is, to prevent duplicate or null values across all those columns.

If you spread related information across multiple tables, typically each table uses the same column for its primary key. For example, a personnel database might have several tables, each with a primary key of employee number. A sales database might have some tables with a primary key of customer number, and other tables with a primary key of order number. Because lookups using the primary key are very fast, you can construct efficient join queries for such tables.

If you leave the PRIMARY KEY clause out entirely, MySQL creates an invisible one for you. It is a 6-byte value that might be longer than you need, thus wasting space. Because it is hidden, you cannot refer to it in queries.

Application Performance Considerations

The extra reliability and scalability features of InnoDB do require more disk storage than equivalent MyISAM tables. You might change the column and index definitions slightly, for better space utilization, reduced I/O and memory consumption when processing result sets, and better query optimization plans making efficient use of index lookups.

If you do set up a numeric ID column for the primary key, use that value to cross-reference with related values in any other tables, particularly for join queries. For example, rather than accepting a country name as input and doing queries searching for the same name, do one lookup to determine the country ID, then do other queries (or a single join query) to look up relevant information across several tables. Rather than storing a customer or catalog item number as a string of digits, potentially using up several bytes, convert it to a numeric ID for storing and querying. A 4-byte unsigned INT column can index over 4 billion items (with the US meaning of billion: 1000 million). For the ranges of the different integer types, see Section 11.2.1, “Integer Types (Exact Value) - INTEGER, INT, SMALLINT, TINYINT, MEDIUMINT, BIGINT”.

Understand Files Associated with InnoDB Tables

InnoDB files require more care and planning than MyISAM files do:

- You must not delete the ibdata files that represent the InnoDB system tablespace.
- Copying InnoDB tables from one server to another requires issuing the FLUSH TABLES ... FOR EXPORT statement first, and copying the *table_name*.cfg file along with the *table_name*.ibd file.

14.2.6.5 AUTO_INCREMENT Handling in InnoDB

InnoDB provides an optimization that significantly improves scalability and performance of SQL statements that insert rows into tables with AUTO_INCREMENT columns. To use the AUTO_INCREMENT mechanism with an InnoDB table, an AUTO_INCREMENT column *ai_col* must be defined as part of an index such that it is possible to perform the equivalent of an indexed SELECT MAX(*ai_col*) lookup on the table to obtain the maximum column value. Typically, this is achieved by making the column the first column of some table index.

This section provides background information on the original ("traditional") implementation of auto-increment locking in InnoDB, explains the configurable locking mechanism, documents the parameter for configuring the mechanism, and describes its behavior and interaction with replication.

Traditional InnoDB Auto-Increment Locking

The original implementation of auto-increment handling in InnoDB uses the following strategy to prevent problems when using the binary log for statement-based replication or for certain recovery scenarios.

If you specify an AUTO_INCREMENT column for an InnoDB table, the table handle in the InnoDB data dictionary contains a special counter called the auto-increment counter that is used in assigning new values for the column. This counter is stored only in main memory, not on disk.

InnoDB uses the following algorithm to initialize the auto-increment counter for a table t that contains an AUTO_INCREMENT column named ai_col: After a server startup, for the first insert into a table t, InnoDB executes the equivalent of this statement:

```
SELECT MAX(ai_col) FROM t FOR UPDATE;
```

InnoDB increments the value retrieved by the statement and assigns it to the column and to the auto-increment counter for the table. By default, the value is incremented by one. This default can be overridden by the auto_increment_increment configuration setting.

If the table is empty, InnoDB uses the value 1. This default can be overridden by the auto_increment_offset configuration setting.

If a SHOW TABLE STATUS statement examines the table t before the auto-increment counter is initialized, InnoDB initializes but does not increment the value and stores it for use by later inserts. This initialization uses a normal exclusive-locking read on the table and the lock lasts to the end of the transaction.

InnoDB follows the same procedure for initializing the auto-increment counter for a freshly created table.

After the auto-increment counter has been initialized, if you do not explicitly specify a value for an AUTO_INCREMENT column, InnoDB increments the counter and assigns the new value to the column. If you insert a row that explicitly specifies the column value, and the value is bigger than the current counter value, the counter is set to the specified column value.

If a user specifies NULL or 0 for the AUTO_INCREMENT column in an INSERT, InnoDB treats the row as if the value was not specified and generates a new value for it.

The behavior of the auto-increment mechanism is not defined if you assign a negative value to the column, or if the value becomes bigger than the maximum integer that can be stored in the specified integer type.

When accessing the auto-increment counter, InnoDB uses a special table-level AUTO-INC lock that it keeps to the end of the current SQL statement, not to the end of the transaction. The special lock release strategy was introduced to improve concurrency for inserts into a table containing an AUTO_INCREMENT column. Nevertheless, two transactions cannot have the AUTO-INC lock on the same table simultaneously, which can have a performance impact if the AUTO-INC lock is held for a long time. That might be the case for a statement such as INSERT INTO t1 ... SELECT ... FROM t2 that inserts all rows from one table into another.

InnoDB uses the in-memory auto-increment counter as long as the server runs. When the server is stopped and restarted, InnoDB reinitializes the counter for each table for the first INSERT to the table, as described earlier.

A server restart also cancels the effect of the AUTO_INCREMENT = *N* table option in CREATE TABLE and ALTER TABLE statements, which you can use with InnoDB tables to set the initial counter value or alter the current counter value.

You may see gaps in the sequence of values assigned to the AUTO_INCREMENT column if you roll back transactions that have generated numbers using the counter.

Configurable InnoDB Auto-Increment Locking

As described in the previous section, InnoDB uses a special lock called the table-level AUTO-INC lock for inserts into tables with AUTO_INCREMENT columns. This lock is normally held to the end of the statement (not to the end of the transaction), to ensure that auto-increment numbers are assigned in a predictable and repeatable order for a given sequence of INSERT statements.

In the case of statement-based replication, this means that when an SQL statement is replicated on a slave server, the same values are used for the auto-increment column as on the master server. The result of execution of multiple INSERT statements is deterministic, and the slave reproduces the same data as on the master. If auto-increment values generated by multiple INSERT statements were interleaved, the result of two concurrent INSERT statements would be nondeterministic, and could not reliably be propagated to a slave server using statement-based replication.

To make this clear, consider an example that uses this table:

```
CREATE TABLE t1 (
  c1 INT(11) NOT NULL AUTO_INCREMENT,
  c2 VARCHAR(10) DEFAULT NULL,
  PRIMARY KEY (c1)
) ENGINE=InnoDB;
```

Suppose that there are two transactions running, each inserting rows into a table with an AUTO_INCREMENT column. One transaction is using an INSERT ... SELECT statement that inserts 1000 rows, and another is using a simple INSERT statement that inserts one row:

```
Tx1: INSERT INTO t1 (c2) SELECT 1000 rows from another table ...
Tx2: INSERT INTO t1 (c2) VALUES ('xxx');
```

InnoDB cannot tell in advance how many rows will be retrieved from the SELECT in the INSERT statement in Tx1, and it assigns the auto-increment values one at a time as the statement proceeds. With a table-level lock, held to the end of the statement, only one INSERT statement referring to table t1 can execute at a time, and the generation of auto-increment numbers by different statements is not interleaved. The auto-increment value generated by the Tx1 INSERT ... SELECT statement will be consecutive, and the (single) auto-increment value used by the INSERT statement in Tx2 will either be smaller or larger than all those used for Tx1, depending on which statement executes first.

As long as the SQL statements execute in the same order when replayed from the binary log (when using statement-based replication, or in recovery scenarios), the results will be the same as they were when Tx1 and Tx2 first ran. Thus, table-level locks held until the end of a statement make INSERT statements using auto-increment safe for use with statement-based replication. However, those locks limit concurrency and scalability when multiple transactions are executing insert statements at the same time.

In the preceding example, if there were no table-level lock, the value of the auto-increment column used for the INSERT in Tx2 depends on precisely when the statement executes. If the INSERT of Tx2 executes while the INSERT of Tx1 is running (rather than before it starts or after it completes), the specific auto-increment values assigned by the two INSERT statements are nondeterministic, and may vary from run to run.

InnoDB can avoid using the table-level AUTO-INC lock for a class of INSERT statements where the number of rows is known in advance, and still preserve deterministic execution and safety for statement-based replication. Further, if you are not using the binary log to replay SQL statements as part of recovery or replication, you can entirely eliminate use of the table-level AUTO-INC lock for even greater concurrency and performance, at the cost of permitting gaps in auto-increment numbers assigned by a statement and potentially having the numbers assigned by concurrently executing statements interleaved.

For INSERT statements where the number of rows to be inserted is known at the beginning of processing the statement, InnoDB quickly allocates the required number of auto-increment values without taking any lock, but only if there is no concurrent session already holding the table-level AUTO-INC lock (because that other statement will be allocating auto-increment values one-by-one as it proceeds). More precisely, such an INSERT statement obtains auto-increment values under the control of a mutex (a light-weight lock) that is *not* held until the statement completes, but only for the duration of the allocation process.

This new locking scheme enables much greater scalability, but it does introduce some subtle differences in how auto-increment values are assigned compared to the original mechanism. To describe the way auto-increment works in InnoDB, the following discussion defines some terms, and explains how InnoDB behaves using different settings of the innodb_autoinc_lock_mode configuration parameter, which you can set at server startup. Additional considerations are described following the explanation of auto-increment locking behavior.

First, some definitions:

- “INSERT-like” statements

  All statements that generate new rows in a table, including INSERT, INSERT ... SELECT, REPLACE, REPLACE ... SELECT, and LOAD DATA.

- “Simple inserts”

  Statements for which the number of rows to be inserted can be determined in advance (when the statement is initially processed). This includes single-row and multiple-row INSERT and REPLACE statements that do not have a nested subquery, but not INSERT ... ON DUPLICATE KEY UPDATE.

- “Bulk inserts”

  Statements for which the number of rows to be inserted (and the number of required auto-increment values) is not known in advance. This includes INSERT ... SELECT, REPLACE ... SELECT, and LOAD DATA statements, but not plain INSERT. InnoDB will assign new values for the AUTO_INCREMENT column one at a time as each row is processed.

- “Mixed-mode inserts”

  These are “simple insert” statements that specify the auto-increment value for some (but not all) of the new rows. An example follows, where c1 is an AUTO_INCREMENT column of table t1:

  ```
  INSERT INTO t1 (c1,c2) VALUES (1,'a'), (NULL,'b'), (5,'c'), (NULL,'d');
  ```

  Another type of “mixed-mode insert” is INSERT ... ON DUPLICATE KEY UPDATE, which in the worst case is in effect an INSERT followed by a UPDATE, where the allocated value for the AUTO_INCREMENT column may or may not be used during the update phase.

There are three possible settings for the innodb_autoinc_lock_mode parameter:

- innodb_autoinc_lock_mode = 0 ("traditional" lock mode)

  This lock mode provides the same behavior as before innodb_autoinc_lock_mode existed. For all "INSERT-like" statements, a special table-level AUTO-INC lock is obtained and held to the end of the statement. This assures that the auto-increment values assigned by any given statement are consecutive.

  This lock mode is provided for:

  - Backward compatibility.
  - Performance testing.
  - Working around issues with "mixed-mode inserts", due to the possible differences in semantics described later.

- innodb_autoinc_lock_mode = 1 ("consecutive" lock mode)

  This is the default lock mode. In this mode, "bulk inserts" use the special AUTO-INC table-level lock and hold it until the end of the statement. This applies to all INSERT ... SELECT, REPLACE ... SELECT, and LOAD DATA statements. Only one statement holding the AUTO-INC lock can execute at a time.

  With this lock mode, "simple inserts" (only) use a new locking model where a light-weight mutex is used during the allocation of auto-increment values, and no table-level AUTO-INC lock is used, unless an AUTO-INC lock is held by another transaction. If another transaction does hold an AUTO-INC lock, a "simple insert" waits for the AUTO-INC lock, as if it too were a "bulk insert".

  This lock mode ensures that, in the presence of INSERT statements where the number of rows is not known in advance (and where auto-increment numbers are assigned as the statement progresses), all auto-increment values assigned by any "INSERT-like" statement are consecutive, and operations are safe for statement-based replication.

  Simply put, the important impact of this lock mode is significantly better scalability. This mode is safe for use with statement-based replication. Further, as with "traditional" lock mode, auto-increment numbers assigned by any given statement are *consecutive*. In this mode, there is *no change* in semantics compared to "traditional" mode for any statement that uses auto-increment, with one important exception.

  The exception is for "mixed-mode inserts", where the user provides explicit values for an AUTO_INCREMENT column for some, but not all, rows in a multiple-row "simple insert". For such inserts, InnoDB will allocate more auto-increment values than the number of rows to be inserted. However, all values automatically assigned are consecutively generated (and thus higher than) the auto-increment value generated by the most recently executed previous statement. "Excess" numbers are lost.

- innodb_autoinc_lock_mode = 2 ("interleaved" lock mode)

  In this lock mode, no "INSERT-like" statements use the table-level AUTO-INC lock, and multiple statements can execute at the same time. This is the fastest and most scalable lock mode, but it is *not safe* when using statement-based replication or recovery scenarios when SQL statements are replayed from the binary log.

  In this lock mode, auto-increment values are guaranteed to be unique and monotonically increasing across all concurrently executing "INSERT-like" statements. However, because multiple statements can be generating numbers at the same time (that is, allocation of numbers is *interleaved* across statements), the values generated for the rows inserted by any given statement may not be consecutive.

If the only statements executing are "simple inserts" where the number of rows to be inserted is known ahead of time, there will be no gaps in the numbers generated for a single statement, except for "mixed-mode inserts". However, when "bulk inserts" are executed, there may be gaps in the auto-increment values assigned by any given statement.

The auto-increment locking modes provided by innodb_autoinc_lock_mode have several usage implications:

- Using auto-increment with replication

  If you are using statement-based replication, set innodb_autoinc_lock_mode to 0 or 1 and use the same value on the master and its slaves. Auto-increment values are not ensured to be the same on the slaves as on the master if you use innodb_autoinc_lock_mode = 2 ("interleaved") or configurations where the master and slaves do not use the same lock mode.

  If you are using row-based or mixed-format replication, all of the auto-increment lock modes are safe, since row-based replication is not sensitive to the order of execution of the SQL statements (and the mixed format uses row-based replication for any statements that are unsafe for statement-based replication).

- "Lost" auto-increment values and sequence gaps

  In all lock modes (0, 1, and 2), if a transaction that generated auto-increment values rolls back, those auto-increment values are "lost". Once a value is generated for an auto-increment column, it cannot be rolled back, whether or not the "INSERT-like" statement is completed, and whether or not the containing transaction is rolled back. Such lost values are not reused. Thus, there may be gaps in the values stored in an AUTO_INCREMENT column of a table.

- Gaps in auto-increment values for "bulk inserts"

  With innodb_autoinc_lock_mode set to 0 ("traditional") or 1 ("consecutive"), the auto-increment values generated by any given statement will be consecutive, without gaps, because the table-level AUTO-INC lock is held until the end of the statement, and only one such statement can execute at a time.

  With innodb_autoinc_lock_mode set to 2 ("interleaved"), there may be gaps in the auto-increment values generated by "bulk inserts," but only if there are concurrently executing "INSERT-like" statements.

  For lock modes 1 or 2, gaps may occur between successive statements because for bulk inserts the exact number of auto-increment values required by each statement may not be known and overestimation is possible.

- Auto-increment values assigned by "mixed-mode inserts"

  Consider a "mixed-mode insert," where a "simple insert" specifies the auto-increment value for some (but not all) resulting rows. Such a statement will behave differently in lock modes 0, 1, and 2. For example, assume c1 is an AUTO_INCREMENT column of table t1, and that the most recent automatically generated sequence number is 100. Consider the following "mixed-mode insert" statement:

  ```
  INSERT INTO t1 (c1,c2) VALUES (1,'a'), (NULL,'b'), (5,'c'), (NULL,'d');
  ```

  With innodb_autoinc_lock_mode set to 0 ("traditional"), the four new rows will be:

  ```
  +-----+------+
  | c1  | c2   |
  ```

```
+-----+------+
|   1 | a    |
| 101 | b    |
|   5 | c    |
| 102 | d    |
+-----+------+
```

The next available auto-increment value will be 103 because the auto-increment values are allocated one at a time, not all at once at the beginning of statement execution. This result is true whether or not there are concurrently executing “`INSERT`-like” statements (of any type).

With `innodb_autoinc_lock_mode` set to 1 (“consecutive”), the four new rows will also be:

```
+-----+------+
| c1  | c2   |
+-----+------+
|   1 | a    |
| 101 | b    |
|   5 | c    |
| 102 | d    |
+-----+------+
```

However, in this case, the next available auto-increment value will be 105, not 103 because four auto-increment values are allocated at the time the statement is processed, but only two are used. This result is true whether or not there are concurrently executing “`INSERT`-like” statements (of any type).

With `innodb_autoinc_lock_mode` set to mode 2 (“interleaved”), the four new rows will be:

```
+-----+------+
| c1  | c2   |
+-----+------+
|   1 | a    |
|   x | b    |
|   5 | c    |
|   y | d    |
+-----+------+
```

The values of *x* and *y* will be unique and larger than any previously generated rows. However, the specific values of *x* and *y* will depend on the number of auto-increment values generated by concurrently executing statements.

Finally, consider the following statement, issued when the most-recently generated sequence number was the value 4:

```
INSERT INTO t1 (c1,c2) VALUES (1,'a'), (NULL,'b'), (5,'c'), (NULL,'d');
```

With any `innodb_autoinc_lock_mode` setting, this statement will generate a duplicate-key error 23000 (`Can't write; duplicate key in table`) because 5 will be allocated for the row `(NULL, 'b')` and insertion of the row `(5, 'c')` will fail.

14.2.6.6 `InnoDB` and `FOREIGN KEY` Constraints

This section describes differences in the InnoDB storage engine's handling of foreign keys as compared with that of the MySQL Server.

Foreign Key Definitions

Foreign key definitions for `InnoDB` tables are subject to the following conditions:

- `InnoDB` permits a foreign key to reference any index column or group of columns. However, in the referenced table, there must be an index where the referenced columns are listed as the *first* columns in the same order.

- `InnoDB` does not currently support foreign keys for tables with user-defined partitioning. This means that no user-partitioned `InnoDB` table may contain foreign key references or columns referenced by foreign keys.

- `InnoDB` allows a foreign key constraint to reference a non-unique key. *This is an `InnoDB` extension to standard SQL.*

Referential Actions

Referential actions for foreign keys of `InnoDB` tables are subject to the following conditions:

- While `SET DEFAULT` is allowed by the MySQL Server, it is rejected as invalid by `InnoDB`. `CREATE TABLE` and `ALTER TABLE` statements using this clause are not allowed for InnoDB tables.

- If there are several rows in the parent table that have the same referenced key value, `InnoDB` acts in foreign key checks as if the other parent rows with the same key value do not exist. For example, if you have defined a `RESTRICT` type constraint, and there is a child row with several parent rows, `InnoDB` does not permit the deletion of any of those parent rows.

- `InnoDB` performs cascading operations through a depth-first algorithm, based on records in the indexes corresponding to the foreign key constraints.

- If `ON UPDATE CASCADE` or `ON UPDATE SET NULL` recurses to update the *same table* it has previously updated during the cascade, it acts like `RESTRICT`. This means that you cannot use self-referential `ON UPDATE CASCADE` or `ON UPDATE SET NULL` operations. This is to prevent infinite loops resulting from cascaded updates. A self-referential `ON DELETE SET NULL`, on the other hand, is possible, as is a self-referential `ON DELETE CASCADE`. Cascading operations may not be nested more than 15 levels deep.

- Like MySQL in general, in an SQL statement that inserts, deletes, or updates many rows, `InnoDB` checks `UNIQUE` and `FOREIGN KEY` constraints row-by-row. When performing foreign key checks, `InnoDB` sets shared row-level locks on child or parent records it has to look at. `InnoDB` checks foreign key constraints immediately; the check is not deferred to transaction commit. According to the SQL standard, the default behavior should be deferred checking. That is, constraints are only checked after the *entire SQL statement* has been processed. Until `InnoDB` implements deferred constraint checking, some things will be impossible, such as deleting a record that refers to itself using a foreign key.

Foreign Key Usage and Error Information

You can obtain general information about foreign keys and their usage from querying the `INFORMATION_SCHEMA.KEY_COLUMN_USAGE` table, and more information more specific to `InnoDB` tables can be found in the `INNODB_SYS_FOREIGN` and `INNODB_SYS_FOREIGN_COLS` tables, also in the `INFORMATION_SCHEMA` database. See also Section 13.1.14.2, "Using `FOREIGN KEY` Constraints".

In addition to `SHOW ERRORS`, in the event of a foreign key error involving `InnoDB` tables (usually Error 150 in the MySQL Server), you can obtain a detailed explanation of the most recent `InnoDB` foreign key error by checking the output of `SHOW ENGINE INNODB STATUS`.

14.2.6.7 Limits on `InnoDB` Tables

Warning

Do *not* convert MySQL system tables in the `mysql` database from `MyISAM` to `InnoDB` tables! This is an unsupported operation. If you do this, MySQL does not restart until you restore the old system tables from a backup or re-generate them with the mysql_install_db script.

Warning

It is not a good idea to configure InnoDB to use data files or log files on NFS volumes. Otherwise, the files might be locked by other processes and become unavailable for use by MySQL.

Maximums and Minimums

- A table can contain a maximum of 1017 columns (raised in MySQL 5.6.9 from the earlier limit of 1000).
- A table can contain a maximum of 64 secondary indexes.
- By default, an index key for a single-column index can be up to 767 bytes. The same length limit applies to any index key prefix. See Section 13.1.11, "CREATE INDEX Syntax". For example, you might hit this limit with a column prefix index of more than 255 characters on a TEXT or VARCHAR column, assuming a UTF-8 character set and the maximum of 3 bytes for each character. When the innodb_large_prefix configuration option is enabled, this length limit is raised to 3072 bytes, for InnoDB tables that use the DYNAMIC and COMPRESSED row formats.

  When you attempt to specify an index prefix length longer than allowed, the length is silently reduced to the maximum length for a nonunique index. For a unique index, exceeding the index prefix limit produces an error. To avoid such errors for replication configurations, avoid setting the innodb_large_prefix option on the master if it cannot also be set on the slaves, and the slaves have unique indexes that could be affected by this limit.

  This configuration option changes the error handling for some combinations of row format and prefix length longer than the maximum allowed. See innodb_large_prefix for details.
- The InnoDB internal maximum key length is 3500 bytes, but MySQL itself restricts this to 3072 bytes. This limit applies to the length of the combined index key in a multi-column index.
- If you reduce the InnoDB page size to 8KB or 4KB by specifying the innodb_page_size option when creating the MySQL instance, the maximum length of the index key is lowered proportionally, based on the limit of 3072 bytes for a 16KB page size. That is, the maximum index key length is 1536 bytes when the page size is 8KB, and 768 bytes when the page size is 4KB.
- The maximum row length, except for variable-length columns (VARBINARY, VARCHAR, BLOB and TEXT), is slightly less than half of a database page. That is, the maximum row length is about 8000 bytes for the default page size of 16KB; if you reduce the page size by specifying the innodb_page_size option when creating the MySQL instance, the maximum row length is 4000 bytes for 8KB pages and 2000 bytes for 4KB pages. LONGBLOB and LONGTEXT columns must be less than 4GB, and the total row length, including BLOB and TEXT columns, must be less than 4GB.

  If a row is less than half a page long, all of it is stored locally within the page. If it exceeds half a page, variable-length columns are chosen for external off-page storage until the row fits within half a page, as described in Section 14.2.10.2, "File Space Management".
- Although InnoDB supports row sizes larger than 65,535 bytes internally, MySQL itself imposes a row-size limit of 65,535 for the combined size of all columns:

```
mysql> CREATE TABLE t (a VARCHAR(8000), b VARCHAR(10000),
    -> c VARCHAR(10000), d VARCHAR(10000), e VARCHAR(10000),
    -> f VARCHAR(10000), g VARCHAR(10000)) ENGINE=InnoDB;
ERROR 1118 (42000): Row size too large. The maximum row size for the
```

```
used table type, not counting BLOBs, is 65535. You have to change some
columns to TEXT or BLOBs
```

See Section E.10.4, “Limits on Table Column Count and Row Size”.

- On some older operating systems, files must be less than 2GB. This is not a limitation of InnoDB itself, but if you require a large tablespace, you will need to configure it using several smaller data files rather than one or a file large data files.
- The combined size of the InnoDB log files can be up to 512GB.
- The minimum tablespace size is slightly larger than 10MB. The maximum tablespace size is four billion database pages (64TB). This is also the maximum size for a table.
- The default database page size in InnoDB is 16KB, or you can lower the page size to 8KB or 4KB by specifying the innodb_page_size option when creating the MySQL instance.

> **Note**
>
> Increasing the page size is not a supported operation: there is no guarantee that InnoDB will function normally with a page size greater than 16KB. Problems compiling or running InnoDB may occur. In particular, ROW_FORMAT=COMPRESSED in the Barracuda file format assumes that the page size is at most 16KB and uses 14-bit pointers.
>
> A MySQL instance using a particular InnoDB page size cannot use data files or log files from an instance that uses a different page size. This limitation could affect restore or downgrade operations using data from MySQL 5.6, which does support page sizes other than 16KB.

Index Types

- InnoDB tables support FULLTEXT indexes. See FULLTEXT Indexes for details.
- InnoDB tables support spatial data types, but not indexes on them.

Restrictions on InnoDB Tables

- ANALYZE TABLE determines index cardinality (as displayed in the Cardinality column of SHOW INDEX output) by doing random dives to each of the index trees and updating index cardinality estimates accordingly. Because these are only estimates, repeated runs of ANALYZE TABLE could produce different numbers. This makes ANALYZE TABLE fast on InnoDB tables but not 100% accurate because it does not take all rows into account.

  You can make the statistics collected by ANALYZE TABLE more precise and more stable by turning on the innodb_stats_persistent configuration option, as explained in Persistent Optimizer Statistics for InnoDB Tables. When that setting is enabled, it is important to run ANALYZE TABLE after major changes to indexed column data, because the statistics are not recalculated periodically (such as after a server restart) as they traditionally have been.

  You can change the number of random dives by modifying the innodb_stats_persistent_sample_pages system variable (if the persistent statistics setting is turned on), or the innodb_stats_transient_sample_pages system variable (if the persistent statistics setting is turned off).

  MySQL uses index cardinality estimates only in join optimization. If some join is not optimized in the right way, you can try using ANALYZE TABLE. In the few cases that ANALYZE TABLE does not produce values good enough for your particular tables, you can use FORCE INDEX with your queries to force the use of a particular index, or set the max_seeks_for_key system variable to ensure that MySQL prefers index lookups over table scans. See Section 5.1.4, “Server System Variables”, and Section C.5.6, “Optimizer-Related Issues”.

- If statements or transactions are running on a table and ANALYZE TABLE is run on the same table followed by a second ANALYZE TABLE operation, the second ANALYZE TABLE operation is blocked until the statements or transactions are completed. This behaviour occurs because ANALYZE TABLE marks the currently loaded table definition as obsolete when ANALYZE TABLE is finished running. New statements or transactions (including a second ANALYZE TABLE statement) must load the new table definition into the table cache, which cannot occur until currently running statements or transactions are completed and the old table definition is purged. Loading multiple concurrent table definitions is not supported.

- SHOW TABLE STATUS does not give accurate statistics on InnoDB tables, except for the physical size reserved by the table. The row count is only a rough estimate used in SQL optimization.

- InnoDB does not keep an internal count of rows in a table because concurrent transactions might “see” different numbers of rows at the same time. To process a SELECT COUNT(*) FROM t statement, InnoDB scans an index of the table, which takes some time if the index is not entirely in the buffer pool. If your table does not change often, using the MySQL query cache is a good solution. To get a fast count, you have to use a counter table you create yourself and let your application update it according to the inserts and deletes it does. If an approximate row count is sufficient, SHOW TABLE STATUS can be used. See Section 14.2.12.1, “InnoDB Performance Tuning Tips”.

- On Windows, InnoDB always stores database and table names internally in lowercase. To move databases in a binary format from Unix to Windows or from Windows to Unix, create all databases and tables using lowercase names.

- An AUTO_INCREMENT column *ai_col* must be defined as part of an index such that it is possible to perform the equivalent of an indexed SELECT MAX(*ai_col*) lookup on the table to obtain the maximum column value. Typically, this is achieved by making the column the first column of some table index.

- While initializing a previously specified AUTO_INCREMENT column on a table, InnoDB sets an exclusive lock on the end of the index associated with the AUTO_INCREMENT column. While accessing the auto-increment counter, InnoDB uses a specific AUTO-INC table lock mode where the lock lasts only to the end of the current SQL statement, not to the end of the entire transaction. Other clients cannot insert into the table while the AUTO-INC table lock is held. See Section 14.2.6.5, “AUTO_INCREMENT Handling in InnoDB”.

- When you restart the MySQL server, InnoDB may reuse an old value that was generated for an AUTO_INCREMENT column but never stored (that is, a value that was generated during an old transaction that was rolled back).

- When an AUTO_INCREMENT integer column runs out of values, a subsequent INSERT operation returns a duplicate-key error. This is general MySQL behavior, similar to how MyISAM works.

- DELETE FROM *tbl_name* does not regenerate the table but instead deletes all rows, one by one.

- Currently, cascaded foreign key actions do not activate triggers.

- You cannot create a table with a column name that matches the name of an internal InnoDB column (including DB_ROW_ID, DB_TRX_ID, DB_ROLL_PTR, and DB_MIX_ID). The server reports error 1005 and refers to error –1 in the error message. This restriction applies only to use of the names in uppercase.

Locking and Transactions

- LOCK TABLES acquires two locks on each table if innodb_table_locks=1 (the default). In addition to a table lock on the MySQL layer, it also acquires an InnoDB table lock. Versions of MySQL before 4.1.2 did not acquire InnoDB table locks; the old behavior can be selected by setting innodb_table_locks=0. If no InnoDB table lock is acquired, LOCK TABLES completes even if some records of the tables are being locked by other transactions.

  In MySQL 5.7, innodb_table_locks=0 has no effect for tables locked explicitly with LOCK TABLES ... WRITE. It does have an effect for tables locked for read or write by LOCK TABLES ... WRITE implicitly (for example, through triggers) or by LOCK TABLES ... READ.

- All InnoDB locks held by a transaction are released when the transaction is committed or aborted. Thus, it does not make much sense to invoke LOCK TABLES on InnoDB tables in autocommit=1 mode because the acquired InnoDB table locks would be released immediately.

- You cannot lock additional tables in the middle of a transaction because LOCK TABLES performs an implicit COMMIT and UNLOCK TABLES.

- The limit on data-modifying transactions is now 96 * 1023 concurrent transactions that generate undo records. As of MySQL 5.7.2, 32 of 128 rollback segments are assigned to non-redo logs for transactions that modify temporary tables and related objects. This reduces the maximum number of concurrent data-modifying transactions from 128K to 96K. The 96K limit assumes that transactions do not modify temporary tables. If all data-modifying transactions also modify temporary tables, the limit is 32K concurrent transactions.

14.2.7 InnoDB Compressed Tables

By using the SQL syntax and MySQL configuration options for compression, you can create tables where the data is stored in compressed form. Compression can help to improve both raw performance and scalability. The compression means less data is transferred between disk and memory, and takes up less space on disk and in memory. The benefits are amplified for tables with secondary indexes, because index data is compressed also. Compression can be especially important for SSD storage devices, because they tend to have lower capacity than HDD devices.

14.2.7.1 Overview of Table Compression

Because processors and cache memories have increased in speed more than disk storage devices, many workloads are disk-bound. Data compression enables smaller database size, reduced I/O, and improved throughput, at the small cost of increased CPU utilization. Compression is especially valuable for read-intensive applications, on systems with enough RAM to keep frequently used data in memory.

An InnoDB table created with ROW_FORMAT=COMPRESSED can use a smaller page size on disk than the usual 16KB default. Smaller pages require less I/O to read from and write to disk, which is especially valuable for SSD devices.

The page size is specified through the KEY_BLOCK_SIZE parameter. The different page size means the table must be in its own .ibd file rather than in the system tablespace, which requires enabling the innodb_file_per_table option. The level of compression is the same regardless of the KEY_BLOCK_SIZE value. As you specify smaller values for KEY_BLOCK_SIZE, you get the I/O benefits of increasingly smaller pages. But if you specify a value that is too small, there is additional overhead to reorganize the pages when data values cannot be compressed enough to fit multiple rows in each page. There is a hard limit on how small KEY_BLOCK_SIZE can be for a table, based on the lengths of the key columns for each of its indexes. Specify a value that is too small, and the CREATE TABLE or ALTER TABLE statement fails.

In the buffer pool, the compressed data is held in small pages, with a page size based on the KEY_BLOCK_SIZE value. For extracting or updating the column values, MySQL also creates a 16KB page in the buffer pool with the uncompressed data. Within the buffer pool, any updates to the uncompressed page are also re-written back to the equivalent compressed page. You might need to size your buffer pool to accommodate the additional data of both compressed and uncompressed pages, although the uncompressed pages are evicted from the buffer pool when space is needed, and then uncompressed again on the next access.

14.2.7.2 Enabling Compression for a Table

Before creating a compressed table, make sure the innodb_file_per_table configuration option is enabled, and innodb_file_format is set to Barracuda. You can set these parameters in the MySQL configuration file my.cnf or my.ini, or with the SET statement without shutting down the MySQL server.

To enable compression for a table, you use the clauses ROW_FORMAT=COMPRESSED, KEY_BLOCK_SIZE, or both in a CREATE TABLE or ALTER TABLE statement.

To create a compressed table, you might use statements like these:

```
SET GLOBAL innodb_file_per_table=1;
SET GLOBAL innodb_file_format=Barracuda;
CREATE TABLE t1
 (c1 INT PRIMARY KEY)
 ROW_FORMAT=COMPRESSED
 KEY_BLOCK_SIZE=8;
```

- If you specify ROW_FORMAT=COMPRESSED, you can omit KEY_BLOCK_SIZE; the default compressed page size of 8KB is used.

- If you specify KEY_BLOCK_SIZE, you can omit ROW_FORMAT=COMPRESSED; compression is enabled automatically.

- To determine the best value for KEY_BLOCK_SIZE, typically you create several copies of the same table with different values for this clause, then measure the size of the resulting .ibd files and see how well each performs with a realistic workload.

- For additional performance-related configuration options, see Section 14.2.7.3, “Tuning Compression for InnoDB Tables”.

The default uncompressed size of InnoDB data pages is 16KB. Depending on the combination of option values, MySQL uses a page size of 1KB, 2KB, 4KB, 8KB, or 16KB for the .ibd file of the table. The actual compression algorithm is not affected by the KEY_BLOCK_SIZE value; the value determines how large each compressed chunk is, which in turn affects how many rows can be packed into each compressed page.

Setting KEY_BLOCK_SIZE=16 typically does not result in much compression, since the normal InnoDB page size is 16KB. This setting may still be useful for tables with many long BLOB, VARCHAR or TEXT columns, because such values often do compress well, and might therefore require fewer overflow pages as described in Section 14.2.7.5, “How Compression Works for InnoDB Tables”.

All indexes of a table (including the clustered index) are compressed using the same page size, as specified in the CREATE TABLE or ALTER TABLE statement. Table attributes such as ROW_FORMAT and KEY_BLOCK_SIZE are not part of the CREATE INDEX syntax, and are ignored if they are specified (although you see them in the output of the SHOW CREATE TABLE statement).

Restrictions on Compressed Tables

Because MySQL versions prior to 5.1 cannot process compressed tables, using compression requires specifying the configuration parameter innodb_file_format=Barracuda, to avoid accidentally introducing compatibility issues.

Table compression is also not available for the InnoDB system tablespace. The system tablespace (space 0, the ibdata* files) can contain user data, but it also contains internal system information, and therefore is never compressed. Thus, compression applies only to tables (and indexes) stored in their own tablespaces, that is, created with the innodb_file_per_table option enabled.

Compression applies to an entire table and all its associated indexes, not to individual rows, despite the clause name ROW_FORMAT.

14.2.7.3 Tuning Compression for InnoDB Tables

Most often, the internal optimizations described in InnoDB Data Storage and Compression ensure that the system runs well with compressed data. However, because the efficiency of compression depends on the nature of your data, you can make decisions that affect the performance of compressed tables:

- Which tables to compress.
- What compressed page size to use.
- Whether to adjust the size of the buffer pool based on run-time performance characteristics, such as the amount of time the system spends compressing and uncompressing data. Whether the workload is more like a data warehouse (primarily queries) or an OLTP system (mix of queries and DML).
- If the system performs DML operations on compressed tables, and the way the data is distributed leads to expensive compression failures at runtime, you might adjust additional advanced configuration options.

Use the guidelines in this section to help make those architectural and configuration choices. When you are ready to conduct long-term testing and put compressed tables into production, see Section 14.2.7.4, “Monitoring Compression at Runtime” for ways to verify the effectiveness of those choices under real-world conditions.

When to Use Compression

In general, compression works best on tables that include a reasonable number of character string columns and where the data is read far more often than it is written. Because there are no guaranteed ways to predict whether or not compression benefits a particular situation, always test with a specific workload and data set running on a representative configuration. Consider the following factors when deciding which tables to compress.

Data Characteristics and Compression

A key determinant of the efficiency of compression in reducing the size of data files is the nature of the data itself. Recall that compression works by identifying repeated strings of bytes in a block of data. Completely randomized data is the worst case. Typical data often has repeated values, and so compresses effectively. Character strings often compress well, whether defined in CHAR, VARCHAR, TEXT or BLOB columns. On the other hand, tables containing mostly binary data (integers or floating point numbers) or data that is previously compressed (for example JPEG or PNG images) may not generally compress well, significantly or at all.

You choose whether to turn on compression for each InnoDB table. A table and all of its indexes use the same (compressed) page size. It might be that the primary key (clustered) index, which contains the data for all columns of a table, compresses more effectively than the secondary indexes. For those cases where there are long rows, the use of compression might result in long column values being stored “off-page”, as discussed in Section 14.2.9.3, “DYNAMIC and COMPRESSED Row Formats”. Those overflow pages may compress well. Given these considerations, for many applications, some tables compress more effectively than others, and you might find that your workload performs best only with a subset of tables compressed.

To determine whether or not to compress a particular table, conduct experiments. You can get a rough estimate of how efficiently your data can be compressed by using a utility that implements LZ77 compression (such as gzip or WinZip) on a copy of the .ibd file for an uncompressed table. You can expect less compression from a MySQL compressed table than from file-based compression tools, because MySQL compresses data in chunks based on the page size, 16KB by default. In addition to user data, the page format includes some internal system data that is not compressed. File-based compression utilities can examine much larger chunks of data, and so might find more repeated strings in a huge file than MySQL can find in an individual page.

Another way to test compression on a specific table is to copy some data from your uncompressed table to a similar, compressed table (having all the same indexes) and look at the size of the resulting .ibd file. For example:

```
use test;
set global innodb_file_per_table=1;
set global innodb_file_format=Barracuda;
set global autocommit=0;

-- Create an uncompressed table with a million or two rows.
create table big_table as select * from information_schema.columns;
insert into big_table select * from big_table;
insert into big_table select * from big_table;
insert into big_table select * from big_table;
insert into big_table select * from big_table;
insert into big_table select * from big_table;
insert into big_table select * from big_table;
insert into big_table select * from big_table;
insert into big_table select * from big_table;
insert into big_table select * from big_table;
insert into big_table select * from big_table;
commit;
alter table big_table add id int unsigned not null primary key auto_increment;

show create table big_table\G

select count(id) from big_table;

-- Check how much space is needed for the uncompressed table.
\! ls -l data/test/big_table.ibd

create table key_block_size_4 like big_table;
alter table key_block_size_4 key_block_size=4 row_format=compressed;

insert into key_block_size_4 select * from big_table;
commit;

-- Check how much space is needed for a compressed table
-- with particular compression settings.
\! ls -l data/test/key_block_size_4.ibd
```

This experiment produced the following numbers, which of course could vary considerably depending on your table structure and data:

```
-rw-rw----  1 cirrus  staff  310378496 Jan  9 13:44 data/test/big_table.ibd
-rw-rw----  1 cirrus  staff  83886080 Jan  9 15:10 data/test/key_block_size_4.ibd
```

To see whether compression is efficient for your particular workload:

- For simple tests, use a MySQL instance with no other compressed tables and run queries against the INFORMATION_SCHEMA.INNODB_CMP table.
- For more elaborate tests involving workloads with multiple compressed tables, run queries against the INFORMATION_SCHEMA.INNODB_CMP_PER_INDEX table. Because the statistics in the INNODB_CMP_PER_INDEX table are expensive to collect, you must enable the configuration option innodb_cmp_per_index_enabled before querying that table, and you might restrict such testing to a development server or a non-critical slave server.
- Run some typical SQL statements against the compressed table you are testing.
- Examine the ratio of successful compression operations to overall compression operations by querying the INFORMATION_SCHEMA.INNODB_CMP or INFORMATION_SCHEMA.INNODB_CMP_PER_INDEX table, and comparing COMPRESS_OPS to COMPRESS_OPS_OK.
- If a high percentage of compression operations complete successfully, the table might be a good candidate for compression.
- If you get a high proportion of compression failures, you can adjust innodb_compression_level, innodb_compression_failure_threshold_pct, and innodb_compression_pad_pct_max options as described in Compression Enhancements for OLTP Workloads, and try further tests.

Database Compression versus Application Compression

Decide whether to compress data in your application or in the table; do not use both types of compression for the same data. When you compress the data in the application and store the results in a compressed table, extra space savings are extremely unlikely, and the double compression just wastes CPU cycles.

Compressing in the Database

When enabled, MySQL table compression is automatic and applies to all columns and index values. The columns can still be tested with operators such as LIKE, and sort operations can still use indexes even when the index values are compressed. Because indexes are often a significant fraction of the total size of a database, compression could result in significant savings in storage, I/O or processor time. The compression and decompression operations happen on the database server, which likely is a powerful system that is sized to handle the expected load.

Compressing in the Application

If you compress data such as text in your application, before it is inserted into the database, You might save overhead for data that does not compress well by compressing some columns and not others. This approach uses CPU cycles for compression and uncompression on the client machine rather than the database server, which might be appropriate for a distributed application with many clients, or where the client machine has spare CPU cycles.

Hybrid Approach

Of course, it is possible to combine these approaches. For some applications, it may be appropriate to use some compressed tables and some uncompressed tables. It may be best to externally compress some data (and store it in uncompressed tables) and allow MySQL to compress (some of) the other tables in the application. As always, up-front design and real-life testing are valuable in reaching the right decision.

Workload Characteristics and Compression

In addition to choosing which tables to compress (and the page size), the workload is another key determinant of performance. If the application is dominated by reads, rather than updates, fewer pages need to be reorganized and recompressed after the index page runs out of room for the per-page “modification log” that MySQL maintains for compressed data. If the updates predominantly change non-indexed columns or those containing BLOBs or large strings that happen to be stored “off-page”, the overhead of compression may be acceptable. If the only changes to a table are INSERTs that use a monotonically increasing primary key, and there are few secondary indexes, there is little need to reorganize and recompress index pages. Since MySQL can “delete-mark” and delete rows on compressed pages “in place” by modifying uncompressed data, DELETE operations on a table are relatively efficient.

For some environments, the time it takes to load data can be as important as run-time retrieval. Especially in data warehouse environments, many tables may be read-only or read-mostly. In those cases, it might or might not be acceptable to pay the price of compression in terms of increased load time, unless the resulting savings in fewer disk reads or in storage cost is significant.

Fundamentally, compression works best when the CPU time is available for compressing and uncompressing data. Thus, if your workload is I/O bound, rather than CPU-bound, you might find that compression can improve overall performance. When you test your application performance with different compression configurations, test on a platform similar to the planned configuration of the production system.

Configuration Characteristics and Compression

Reading and writing database pages from and to disk is the slowest aspect of system performance. Compression attempts to reduce I/O by using CPU time to compress and uncompress data, and is most effective when I/O is a relatively scarce resource compared to processor cycles.

This is often especially the case when running in a multi-user environment with fast, multi-core CPUs. When a page of a compressed table is in memory, MySQL often uses additional memory, typically 16KB, in the buffer pool for an uncompressed copy of the page. The adaptive LRU algorithm attempts to balance the use of memory between compressed and uncompressed pages to take into account whether the workload is running in an I/O-bound or CPU-bound manner. Still, a configuration with more memory dedicated to the buffer pool tends to run better when using compressed tables than a configuration where memory is highly constrained.

Choosing the Compressed Page Size

The optimal setting of the compressed page size depends on the type and distribution of data that the table and its indexes contain. The compressed page size should always be bigger than the maximum record size, or operations may fail as noted in Compression of B-Tree Pages.

Setting the compressed page size too large wastes some space, but the pages do not have to be compressed as often. If the compressed page size is set too small, inserts or updates may require time-consuming recompression, and the B-tree nodes may have to be split more frequently, leading to bigger data files and less efficient indexing.

Typically, you set the compressed page size to 8K or 4K bytes. Given that the maximum row size for an InnoDB table is around 8K, KEY_BLOCK_SIZE=8 is usually a safe choice.

14.2.7.4 Monitoring Compression at Runtime

Overall application performance, CPU and I/O utilization and the size of disk files are good indicators of how effective compression is for your application. This section builds on the performance tuning advice from Section 14.2.7.3, “Tuning Compression for InnoDB Tables”, and shows how to find problems that might not turn up during initial testing.

To dig deeper into performance considerations for compressed tables, you can monitor compression performance at runtime using the Information Schema tables described in Example 14.11, “Using the Compression Information Schema Tables”. These tables reflect the internal use of memory and the rates of compression used overall.

The INNODB_CMP table reports information about compression activity for each compressed page size (KEY_BLOCK_SIZE) in use. The information in these tables is system-wide: it summarizes the compression statistics across all compressed tables in your database. You can use this data to help decide whether or not to compress a table by examining these tables when no other compressed tables are being accessed. It involves relatively low overhead on the server, so you might query it periodically on a production server to check the overall efficiency of the compression feature.

The INNODB_CMP_PER_INDEX table reports information about compression activity for individual tables and indexes. This information is more targeted and more useful for evaluating compression efficiency and diagnosing performance issues one table or index at a time. (Because that each InnoDB table is represented as a clustered index, MySQL does not make a big distinction between tables and indexes in this context.) The INNODB_CMP_PER_INDEX table does involve substantial overhead, so it is more suitable for development servers, where you can compare the effects of different workloads, data, and compression settings in isolation. To guard against imposing this monitoring overhead by accident, you must enable the innodb_cmp_per_index_enabled configuration option before you can query the INNODB_CMP_PER_INDEX table.

The key statistics to consider are the number of, and amount of time spent performing, compression and uncompression operations. Since MySQL splits B-tree nodes when they are too full to contain the compressed data following a modification, compare the number of “successful” compression operations with the number of such operations overall. Based on the information in the INNODB_CMP and INNODB_CMP_PER_INDEX tables and overall application performance and hardware resource utilization, you might make changes in your hardware configuration, adjust the size of the buffer pool, choose a different page size, or select a different set of tables to compress.

If the amount of CPU time required for compressing and uncompressing is high, changing to faster or multi-core CPUs can help improve performance with the same data, application workload and set of compressed tables. Increasing the size of the buffer pool might also help performance, so that more uncompressed pages can stay in memory, reducing the need to uncompress pages that exist in memory only in compressed form.

A large number of compression operations overall (compared to the number of INSERT, UPDATE and DELETE operations in your application and the size of the database) could indicate that some of your compressed tables are being updated too heavily for effective compression. If so, choose a larger page size, or be more selective about which tables you compress.

If the number of “successful” compression operations (COMPRESS_OPS_OK) is a high percentage of the total number of compression operations (COMPRESS_OPS), then the system is likely performing well. If the ratio is low, then MySQL is reorganizing, recompressing, and splitting B-tree nodes more often than is desirable. In this case, avoid compressing some tables, or increase KEY_BLOCK_SIZE for some of the compressed tables. You might turn off compression for tables that cause the number of “compression failures” in your application to be more than 1% or 2% of the total. (Such a failure ratio might be acceptable during a temporary operation such as a data load).

14.2.7.5 How Compression Works for InnoDB Tables

This section describes some internal implementation details about compression for InnoDB tables. The information presented here may be helpful in tuning for performance, but is not necessary to know for basic use of compression.

Compression Algorithms

Some operating systems implement compression at the file system level. Files are typically divided into fixed-size blocks that are compressed into variable-size blocks, which easily leads into fragmentation. Every time something inside a block is modified, the whole block is recompressed before it is written to disk. These properties make this compression technique unsuitable for use in an update-intensive database system.

MySQL implements compression with the help of the well-known zlib library, which implements the LZ77 compression algorithm. This compression algorithm is mature, robust, and efficient in both CPU utilization and in reduction of data size. The algorithm is “lossless”, so that the original uncompressed data can always be reconstructed from the compressed form. LZ77 compression works by finding sequences of data that are repeated within the data to be compressed. The patterns of values in your data determine how well it compresses, but typical user data often compresses by 50% or more.

Unlike compression performed by an application, or compression features of some other database management systems, InnoDB compression applies both to user data and to indexes. In many cases, indexes can constitute 40-50% or more of the total database size, so this difference is significant. When compression is working well for a data set, the size of the InnoDB data files (the `.idb` files) is 25% to 50% of the uncompressed size or possibly smaller. Depending on the workload, this smaller database can in turn lead to a reduction in I/O, and an increase in throughput, at a modest cost in terms of increased CPU utilization. You can adjust the balance between compression level and CPU overhead by modifying the `innodb_compression_level` configuration option.

InnoDB Data Storage and Compression

All user data in InnoDB tables is stored in pages comprising a B-tree index (the clustered index). In some other database systems, this type of index is called an “index-organized table”. Each row in the index node contains the values of the (user-specified or system-generated) primary key and all the other columns of the table.

Secondary indexes in InnoDB tables are also B-trees, containing pairs of values: the index key and a pointer to a row in the clustered index. The pointer is in fact the value of the primary key of the table, which is used to access the clustered index if columns other than the index key and primary key are required. Secondary index records must always fit on a single B-tree page.

The compression of B-tree nodes (of both clustered and secondary indexes) is handled differently from compression of overflow pages used to store long `VARCHAR`, `BLOB`, or `TEXT` columns, as explained in the following sections.

Compression of B-Tree Pages

Because they are frequently updated, B-tree pages require special treatment. It is important to minimize the number of times B-tree nodes are split, as well as to minimize the need to uncompress and recompress their content.

One technique MySQL uses is to maintain some system information in the B-tree node in uncompressed form, thus facilitating certain in-place updates. For example, this allows rows to be delete-marked and deleted without any compression operation.

In addition, MySQL attempts to avoid unnecessary uncompression and recompression of index pages when they are changed. Within each B-tree page, the system keeps an uncompressed “modification log” to record changes made to the page. Updates and inserts of small records may be written to this modification log without requiring the entire page to be completely reconstructed.

When the space for the modification log runs out, InnoDB uncompresses the page, applies the changes and recompresses the page. If recompression fails (a situation known as a compression failure), the B-tree nodes are split and the process is repeated until the update or insert succeeds.

To avoid frequent compression failures in write-intensive workloads, such as for OLTP applications, MySQL sometimes reserves some empty space (padding) in the page, so that the modification log fills up sooner and the page is recompressed while there is still enough room to avoid splitting it. The amount of padding space left in each page varies as the system keeps track of the frequency of page splits. On a busy server doing frequent writes to compressed tables, you can adjust the innodb_compression_failure_threshold_pct, and innodb_compression_pad_pct_max configuration options to fine-tune this mechanism.

Generally, MySQL requires that each B-tree page in an InnoDB table can accommodate at least two records. For compressed tables, this requirement has been relaxed. Leaf pages of B-tree nodes (whether of the primary key or secondary indexes) only need to accommodate one record, but that record must fit, in uncompressed form, in the per-page modification log. If innodb_strict_mode is ON, MySQL checks the maximum row size during CREATE TABLE or CREATE INDEX. If the row does not fit, the following error message is issued: ERROR HY000: Too big row.

If you create a table when innodb_strict_mode is OFF, and a subsequent INSERT or UPDATE statement attempts to create an index entry that does not fit in the size of the compressed page, the operation fails with ERROR 42000: Row size too large. (This error message does not name the index for which the record is too large, or mention the length of the index record or the maximum record size on that particular index page.) To solve this problem, rebuild the table with ALTER TABLE and select a larger compressed page size (KEY_BLOCK_SIZE), shorten any column prefix indexes, or disable compression entirely with ROW_FORMAT=DYNAMIC or ROW_FORMAT=COMPACT.

Compressing BLOB, VARCHAR, and TEXT Columns

In an InnoDB table, BLOB, VARCHAR, and TEXT columns that are not part of the primary key may be stored on separately allocated overflow pages. We refer to these columns as off-page columns. Their values are stored on singly-linked lists of overflow pages.

For tables created in ROW_FORMAT=DYNAMIC or ROW_FORMAT=COMPRESSED, the values of BLOB, TEXT, or VARCHAR columns may be stored fully off-page, depending on their length and the length of the entire row. For columns that are stored off-page, the clustered index record only contains 20-byte pointers to the overflow pages, one per column. Whether any columns are stored off-page depends on the page size and the total size of the row. When the row is too long to fit entirely within the page of the clustered index, MySQL chooses the longest columns for off-page storage until the row fits on the clustered index page. As noted above, if a row does not fit by itself on a compressed page, an error occurs.

Tables created in older versions of MySQL use the Antelope file format, which supports only ROW_FORMAT=REDUNDANT and ROW_FORMAT=COMPACT. In these formats, MySQL stores the first 768 bytes of BLOB, VARCHAR, and TEXT columns in the clustered index record along with the primary key. The 768-byte prefix is followed by a 20-byte pointer to the overflow pages that contain the rest of the column value.

When a table is in COMPRESSED format, all data written to overflow pages is compressed “as is”; that is, MySQL applies the zlib compression algorithm to the entire data item. Other than the data, compressed overflow pages contain an uncompressed header and trailer comprising a page checksum and a link to the next overflow page, among other things. Therefore, very significant storage savings can be obtained for longer BLOB, TEXT, or VARCHAR columns if the data is highly compressible, as is often the case with text data. Image data, such as JPEG, is typically already compressed and so does not benefit much from being stored in a compressed table; the double compression can waste CPU cycles for little or no space savings.

The overflow pages are of the same size as other pages. A row containing ten columns stored off-page occupies ten overflow pages, even if the total length of the columns is only 8K bytes. In an uncompressed table, ten uncompressed overflow pages occupy 160K bytes. In a compressed table with an 8K page size, they occupy only 80K bytes. Thus, it is often more efficient to use compressed table format for tables with long column values.

Using a 16K compressed page size can reduce storage and I/O costs for BLOB, VARCHAR, or TEXT columns, because such data often compress well, and might therefore require fewer overflow pages, even though the B-tree nodes themselves take as many pages as in the uncompressed form.

Compression and the InnoDB Buffer Pool

In a compressed InnoDB table, every compressed page (whether 1K, 2K, 4K or 8K) corresponds to an uncompressed page of 16K bytes (or a smaller size if innodb_page_size is set). To access the data in a page, MySQL reads the compressed page from disk if it is not already in the buffer pool, then uncompresses the page to its original form. This section describes how InnoDB manages the buffer pool with respect to pages of compressed tables.

To minimize I/O and to reduce the need to uncompress a page, at times the buffer pool contains both the compressed and uncompressed form of a database page. To make room for other required database pages, MySQL can evict from the buffer pool an uncompressed page, while leaving the compressed page in memory. Or, if a page has not been accessed in a while, the compressed form of the page might be written to disk, to free space for other data. Thus, at any given time, the buffer pool might contain both the compressed and uncompressed forms of the page, or only the compressed form of the page, or neither.

MySQL keeps track of which pages to keep in memory and which to evict using a least-recently-used (LRU) list, so that hot (frequently accessed) data tends to stay in memory. When compressed tables are accessed, MySQL uses an adaptive LRU algorithm to achieve an appropriate balance of compressed and uncompressed pages in memory. This adaptive algorithm is sensitive to whether the system is running in an I/O-bound or CPU-bound manner. The goal is to avoid spending too much processing time uncompressing pages when the CPU is busy, and to avoid doing excess I/O when the CPU has spare cycles that can be used for uncompressing compressed pages (that may already be in memory). When the system is I/O-bound, the algorithm prefers to evict the uncompressed copy of a page rather than both copies, to make more room for other disk pages to become memory resident. When the system is CPU-bound, MySQL prefers to evict both the compressed and uncompressed page, so that more memory can be used for “hot” pages and reducing the need to uncompress data in memory only in compressed form.

Compression and the InnoDB Redo Log Files

Before a compressed page is written to a data file, MySQL writes a copy of the page to the redo log (if it has been recompressed since the last time it was written to the database). This is done to ensure that redo logs are usable for crash recovery, even in the unlikely case that the zlib library is upgraded and that change introduces a compatibility problem with the compressed data. Therefore, some increase in the size of log files, or a need for more frequent checkpoints, can be expected when using compression. The amount of increase in the log file size or checkpoint frequency depends on the number of times compressed pages are modified in a way that requires reorganization and recompression.

Note that compressed tables use a different file format for the redo log and the per-table tablespaces than in MySQL 5.1 and earlier. The MySQL Enterprise Backup product supports this latest Barracuda file format for compressed InnoDB tables. The older InnoDB Hot Backup product can only back up tables using the file format Antelope, and thus does not support compressed InnoDB tables.

14.2.7.6 SQL Compression Syntax Warnings and Errors

Specifying ROW_FORMAT=COMPRESSED or KEY_BLOCK_SIZE in CREATE TABLE or ALTER TABLE statements produces the following warnings if the Barracuda file format is not enabled. You can view them with the SHOW WARNINGS statement.

Level	Code	Message
Warning	1478	InnoDB: KEY_BLOCK_SIZE requires innodb_file_per_table.
Warning	1478	InnoDB: KEY_BLOCK_SIZE requires innodb_file_format=1

Level	Code	Message
Warning	1478	InnoDB: ignoring KEY_BLOCK_SIZE=*4*.
Warning	1478	InnoDB: ROW_FORMAT=COMPRESSED requires innodb_file_per_table.
Warning	1478	InnoDB: assuming ROW_FORMAT=COMPACT.

Notes:

- By default, these messages are only warnings, not errors, and the table is created without compression, as if the options were not specified.
- When innodb_strict_mode is enabled, MySQL generates an error, not a warning, for these cases. The table is not created if the current configuration does not permit using compressed tables.

The "non-strict" behavior lets you import a mysqldump file into a database that does not support compressed tables, even if the source database contained compressed tables. In that case, MySQL creates the table in ROW_FORMAT=COMPACT instead of preventing the operation.

To import the dump file into a new database, and have the tables re-created as they exist in the original database, ensure the server has the proper settings for the configuration parameters innodb_file_format and innodb_file_per_table.

The attribute KEY_BLOCK_SIZE is permitted only when ROW_FORMAT is specified as COMPRESSED or is omitted. Specifying a KEY_BLOCK_SIZE with any other ROW_FORMAT generates a warning that you can view with SHOW WARNINGS. However, the table is non-compressed; the specified KEY_BLOCK_SIZE is ignored).

Level	Code	Message
Warning	1478	InnoDB: ignoring KEY_BLOCK_SIZE=*n* unless ROW_FORMAT=COMPRESSED.

If you are running with innodb_strict_mode enabled, the combination of a KEY_BLOCK_SIZE with any ROW_FORMAT other than COMPRESSED generates an error, not a warning, and the table is not created.

Table 14.2, "Meaning of CREATE TABLE and ALTER TABLE options" summarizes how the various options on CREATE TABLE and ALTER TABLE are handled.

Table 14.2 Meaning of CREATE TABLE and ALTER TABLE options

Option	Usage	Description
ROW_FORMAT= REDUNDANT	Storage format used prior to MySQL 5.0.3	Less efficient than ROW_FORMAT=COMPACT; for backward compatibility
ROW_FORMAT= COMPACT	Default storage format since MySQL 5.0.3	Stores a prefix of 768 bytes of long column values in the clustered index page, with the remaining bytes stored in an overflow page
ROW_FORMAT= DYNAMIC	Available only with innodb_file _format=Barracuda	Store values within the clustered index page if they fit; if not, stores only a 20-byte pointer to an overflow page (no prefix)
ROW_FORMAT= COMPRESSED	Available only with innodb_file _format=Barracuda	Compresses the table and indexes using zlib to default compressed page size of 8K bytes; implies ROW_FORMAT=DYNAMIC
KEY_BLOCK_ SIZE=*n*	Available only with innodb_file _format=Barracuda	Specifies compressed page size of 1, 2, 4, 8 or 16 kilobytes; implies ROW_FORMAT=DYNAMIC and ROW_FORMAT=COMPRESSED

Table 14.3, “CREATE/ALTER TABLE Warnings and Errors when InnoDB Strict Mode is OFF” summarizes error conditions that occur with certain combinations of configuration parameters and options on the CREATE TABLE or ALTER TABLE statements, and how the options appear in the output of SHOW TABLE STATUS.

When innodb_strict_mode is OFF, MySQL creates or alters the table, but ignores certain settings as shown below. You can see the warning messages in the MySQL error log. When innodb_strict_mode is ON, these specified combinations of options generate errors, and the table is not created or altered. To see the full description of the error condition, issue the SHOW ERRORS statement: example:

```
mysql> CREATE TABLE x (id INT PRIMARY KEY, c INT)

-> ENGINE=INNODB KEY_BLOCK_SIZE=33333;

ERROR 1005 (HY000): Can't create table 'test.x' (errno: 1478)

mysql> SHOW ERRORS;
+-------+------+-------------------------------------------+
| Level | Code | Message                                   |
+-------+------+-------------------------------------------+
| Error | 1478 | InnoDB: invalid KEY_BLOCK_SIZE=33333.     |
| Error | 1005 | Can't create table 'test.x' (errno: 1478) |
+-------+------+-------------------------------------------+

2 rows in set (0.00 sec)
```

Table 14.3 CREATE/ALTER TABLE Warnings and Errors when InnoDB Strict Mode is OFF

Syntax	Warning or Error Condition	Resulting ROW_FORMAT, as shown in SHOW TABLE STATUS
ROW_FORMAT=REDUNDANT	None	REDUNDANT
ROW_FORMAT=COMPACT	None	COMPACT
ROW_FORMAT=COMPRESSED or ROW_FORMAT=DYNAMIC or KEY_BLOCK_SIZE is specified	Ignored unless both innodb_file_format=Barracuda and innodb_file_per_table are enabled	COMPACT
Invalid KEY_BLOCK_SIZE is specified (not 1, 2, 4, 8 or 16)	KEY_BLOCK_SIZE is ignored	the requested one, or COMPACT by default
ROW_FORMAT=COMPRESSED and valid KEY_BLOCK_SIZE are specified	None; KEY_BLOCK_SIZE specified is used, not the 8K default	COMPRESSED
KEY_BLOCK_SIZE is specified with REDUNDANT, COMPACT or DYNAMIC row format	KEY_BLOCK_SIZE is ignored	REDUNDANT, COMPACT or DYNAMIC
ROW_FORMAT is not one of REDUNDANT, COMPACT, DYNAMIC or COMPRESSED	Ignored if recognized by the MySQL parser. Otherwise, an error is issued.	COMPACT or N/A

When innodb_strict_mode is ON, MySQL rejects invalid ROW_FORMAT or KEY_BLOCK_SIZE parameters. For compatibility with earlier versions of MySQL, strict mode is not enabled by default; instead, MySQL issues warnings (not errors) for ignored invalid parameters.

Note that it is not possible to see the chosen KEY_BLOCK_SIZE using SHOW TABLE STATUS. The statement SHOW CREATE TABLE displays the KEY_BLOCK_SIZE (even if it was ignored when creating the table). The real compressed page size of the table cannot be displayed by MySQL.

14.2.8 InnoDB File-Format Management

As InnoDB evolves, new on-disk data structures are sometimes required to support new features. Features such as compressed tables (see Section 14.2.7, "InnoDB Compressed Tables"), and long variable-length columns stored off-page (see Section 14.2.9, "InnoDB Row Storage and Row Formats") require data file formats that are not compatible with prior versions of InnoDB. These features both require use of the new Barracuda file format.

Note

All other new features are compatible with the original Antelope file format and do not require the Barracuda file format.

This section discusses enabling file formats for new InnoDB tables, verifying compatibility of different file formats between MySQL releases, identifying the file format in use, downgrading the file format, and file format names that may be used in the future.

Named File Formats. InnoDB uses named file formats to help manage compatibility in upgrade and downgrade situations, or heterogeneous systems running different levels of MySQL. Currently, Antelope and Barracuda file formats are supported. Barracuda is the newest file format. It supports important InnoDB features such as compressed tables and the DYNAMIC row format for more efficient BLOB storage. The original InnoDB file format, which previously did not have a name, is now known as Antelope. Future versions of InnoDB may introduce a series of file formats, identified with the names of animals, in ascending alphabetic order.

14.2.8.1 Enabling File Formats

The innodb_file_format configuration parameter defines the file format to use for new InnoDB tables. This parameter is only applicable for tables that have their own tablespace, and therefore requires that innodb_file_per_table be enabled.

The innodb_file_format parameter currently supports Antelope and Barracuda file formats. To create new tables that take advantage of features supported by the Barracuda file format, such as table compression or the new DYNAMIC row format, set innodb_file_format to BARRACUDA.

To preclude the use of new features supported by the Barracuda file format that would make your database inaccessible to the built-in InnoDB in MySQL 5.1 and prior releases, omit innodb_file_format or set it to Antelope.

You can set the value of innodb_file_format on the command line when you start mysqld, or in the option file my.cnf (Unix operating systems) or my.ini (Windows). You can also change it dynamically with the SET GLOBAL statement.

```
mysql> SET GLOBAL innodb_file_format=BARRACUDA;
Query OK, 0 rows affected (0.00 sec)
```

Although Oracle recommends using the Barracuda format for new tables where practical, in MySQL 5.5 the default file format is still Antelope, for maximum compatibility with replication configurations containing different MySQL releases.

14.2.8.2 Verifying File Format Compatibility

InnoDB 1.1 incorporates several checks to guard against the possible crashes and data corruptions that might occur if you run an older release of the MySQL server on InnoDB data files using a newer file format. These checks take place when the server is started, and when you first access a table. This section describes these checks, how you can control them, and error and warning conditions that might arise.

Backward Compatibility

Considerations of backward compatibility only apply when using a recent version of InnoDB (the InnoDB Plugin, or MySQL 5.5 and higher with InnoDB 1.1) alongside an older one (MySQL 5.1 or earlier, with the built-in InnoDB rather than the InnoDB Plugin). To minimize the chance of compatibility issues, you can standardize on the InnoDB Plugin for all your MySQL 5.1 and earlier database servers.

In general, a newer version of InnoDB may create a table or index that cannot safely be read or written with a prior version of InnoDB without risk of crashes, hangs, wrong results or corruptions. InnoDB 1.1 includes a mechanism to guard against these conditions, and to help preserve compatibility among database files and versions of InnoDB. This mechanism lets you take advantage of some new features of an InnoDB release (such as performance improvements and bug fixes), and still preserve the option of using your database with a prior version of InnoDB, by preventing accidental use of new features that create downward-incompatible disk files.

If a version of InnoDB supports a particular file format (whether or not that format is the default), you can query and update any table that requires that format or an earlier format. Only the creation of new tables using new features is limited based on the particular file format enabled. Conversely, if a tablespace contains a table or index that uses a file format that is not supported by the currently running software, it cannot be accessed at all, even for read access.

The only way to “downgrade” an InnoDB tablespace to an earlier file format is to copy the data to a new table, in a tablespace that uses the earlier format. This can be done with the `ALTER TABLE` statement, as described in Section 14.2.8.4, “Downgrading the File Format”.

The easiest way to determine the file format of an existing InnoDB tablespace is to examine the properties of the table it contains, using the `SHOW TABLE STATUS` command or querying the table `INFORMATION_SCHEMA.TABLES`. If the `Row_format` of the table is reported as `'Compressed'` or `'Dynamic'`, the tablespace containing the table uses the Barracuda format. Otherwise, it uses the prior InnoDB file format, Antelope.

Internal Details

Every InnoDB per-table tablespace (represented by a `*.ibd` file) file is labeled with a file format identifier. The system tablespace (represented by the `ibdata` files) is tagged with the “highest” file format in use in a group of InnoDB database files, and this tag is checked when the files are opened.

Creating a compressed table, or a table with `ROW_FORMAT=DYNAMIC`, updates the file header for the corresponding `.ibd` file and the table type in the InnoDB data dictionary with the identifier for the Barracuda file format. From that point forward, the table cannot be used with a version of InnoDB that does not support this new file format. To protect against anomalous behavior, InnoDB version 5.0.21 and later performs a compatibility check when the table is opened. (In many cases, the `ALTER TABLE` statement recreates a table and thus changes its properties. The special case of adding or dropping indexes without rebuilding the table is described in `InnoDB` Fast Index Creation.)

Definition of ib-file set

To avoid confusion, for the purposes of this discussion we define the term “ib-file set” to mean the set of operating system files that InnoDB manages as a unit. The ib-file set includes the following files:

- The system tablespace (one or more `ibdata` files) that contain internal system information (including internal catalogs and undo information) and may include user data and indexes.

- Zero or more single-table tablespaces (also called “file per table” files, named *.ibd files).
- InnoDB log files; usually two, ib_logfile0 and ib_logfile1. Used for crash recovery and in backups.

An “ib-file set” does not include the corresponding .frm files that contain metadata about InnoDB tables. The .frm files are created and managed by MySQL, and can sometimes get out of sync with the internal metadata in InnoDB.

Multiple tables, even from more than one database, can be stored in a single “ib-file set”. (In MySQL, a “database” is a logical collection of tables, what other systems refer to as a “schema” or “catalog”.)

Compatibility Check When InnoDB Is Started

To prevent possible crashes or data corruptions when InnoDB opens an ib-file set, it checks that it can fully support the file formats in use within the ib-file set. If the system is restarted following a crash, or a “fast shutdown” (i.e., innodb_fast_shutdown is greater than zero), there may be on-disk data structures (such as redo or undo entries, or doublewrite pages) that are in a “too-new” format for the current software. During the recovery process, serious damage can be done to your data files if these data structures are accessed. The startup check of the file format occurs before any recovery process begins, thereby preventing consistency issues with the new tables or startup problems for the MySQL server.

Beginning with version InnoDB 1.0.1, the system tablespace records an identifier or tag for the “highest” file format used by any table in any of the tablespaces that is part of the ib-file set. Checks against this file format tag are controlled by the configuration parameter innodb_file_format_check, which is ON by default.

If the file format tag in the system tablespace is newer or higher than the highest version supported by the particular currently executing software and if innodb_file_format_check is ON, the following error is issued when the server is started:

```
InnoDB: Error: the system tablespace is in a
file format that this version doesn't support
```

You can also set innodb_file_format to a file format name. Doing so prevents InnoDB from starting if the current software does not support the file format specified. It also sets the “high water mark” to the value you specify. The ability to set innodb_file_format_check will be useful (with future releases of InnoDB) if you manually “downgrade” all of the tables in an ib-file set (as described in Downgrading the InnoDB Storage Engine). You can then rely on the file format check at startup if you subsequently use an older version of InnoDB to access the ib-file set.

In some limited circumstances, you might want to start the server and use an ib-file set that is in a “too new” format (one that is not supported by the software you are using). If you set the configuration parameter innodb_file_format_check to OFF, InnoDB opens the database, but issues this warning message in the error log:

```
InnoDB: Warning: the system tablespace is in a
file format that this version doesn't support
```

Note

This is a very dangerous setting, as it permits the recovery process to run, possibly corrupting your database if the previous shutdown was a crash or “fast shutdown”. You should only set innodb_file_format_check to OFF if you are sure that the previous shutdown was done with innodb_fast_shutdown=0, so that essentially

> no recovery process occurs. In a future release, this parameter setting may be renamed from OFF to UNSAFE. (However, until there are newer releases of InnoDB that support additional file formats, even disabling the startup checking is in fact “safe”.)

The parameter innodb_file_format_check affects only what happens when a database is opened, not subsequently. Conversely, the parameter innodb_file_format (which enables a specific format) only determines whether or not a new table can be created in the enabled format and has no effect on whether or not a database can be opened.

The file format tag is a “high water mark”, and as such it is increased after the server is started, if a table in a “higher” format is created or an existing table is accessed for read or write (assuming its format is supported). If you access an existing table in a format higher than the format the running software supports, the system tablespace tag is not updated, but table-level compatibility checking applies (and an error is issued), as described in Compatibility Check When a Table Is Opened. Any time the high water mark is updated, the value of innodb_file_format_check is updated as well, so the command SELECT @@innodb_file_format_check; displays the name of the newest file format known to be used by tables in the currently open ib-file set and supported by the currently executing software.

To best illustrate this behavior, consider the scenario described in Table 14.4, “InnoDB Data File Compatibility and Related InnoDB Parameters”. Imagine that some future version of InnoDB supports the Cheetah format and that an ib-file set has been used with that version.

Table 14.4 InnoDB Data File Compatibility and Related InnoDB Parameters

innodb file format check	innodb file format	Highest file format used in ib-file set	Highest file format supported by InnoDB	Result
OFF	Antelope or Barracuda	Barracuda	Barracuda	Database can be opened; tables can be created which require Antelope or Barracuda file format
OFF	Antelope or Barracuda	Cheetah	Barracuda	Database can be opened with a warning, since the database contains files in a “too new” format; tables can be created in Antelope or Barracuda file format; tables in Cheetah format cannot be accessed
OFF	Cheetah	Barracuda	Barracuda	Database cannot be opened; innodb_file_format cannot be set to Cheetah
ON	Antelope or Barracuda	Barracuda	Barracuda	Database can be opened; tables can be created in Antelope or Barracuda file format
ON	Antelope or Barracuda	Cheetah	Barracuda	Database cannot be opened, since the database contains files in a “too new” format (Cheetah)
ON	Cheetah	Barracuda	Barracuda	Database cannot be opened; innodb_file_format cannot be set to Cheetah

Compatibility Check When a Table Is Opened

When a table is first accessed, InnoDB (including some releases prior to InnoDB 1.0) checks that the file format of the tablespace in which the table is stored is fully supported. This check prevents crashes or corruptions that would otherwise occur when tables using a “too new” data structure are encountered.

All tables using any file format supported by a release can be read or written (assuming the user has sufficient privileges). The setting of the system configuration parameter innodb_file_format can prevent creating a new table that uses specific file formats, even if they are supported by a given release.

Such a setting might be used to preserve backward compatibility, but it does not prevent accessing any table that uses any supported format.

As noted in Named File Formats, versions of MySQL older than 5.0.21 cannot reliably use database files created by newer versions if a new file format was used when a table was created. To prevent various error conditions or corruptions, InnoDB checks file format compatibility when it opens a file (for example, upon first access to a table). If the currently running version of InnoDB does not support the file format identified by the table type in the InnoDB data dictionary, MySQL reports the following error:

```
ERROR 1146 (42S02): Table 'test.t1' doesn't exist
```

InnoDB also writes a message to the error log:

```
InnoDB: table test/t1: unknown table type 33
```

The table type should be equal to the tablespace flags, which contains the file format version as discussed in Section 14.2.8.3, “Identifying the File Format in Use”.

Versions of InnoDB prior to MySQL 4.1 did not include table format identifiers in the database files, and versions prior to MySQL 5.0.21 did not include a table format compatibility check. Therefore, there is no way to ensure proper operations if a table in a “too new” format is used with versions of InnoDB prior to 5.0.21.

The file format management capability in InnoDB 1.0 and higher (tablespace tagging and run-time checks) allows InnoDB to verify as soon as possible that the running version of software can properly process the tables existing in the database.

If you permit InnoDB to open a database containing files in a format it does not support (by setting the parameter `innodb_file_format_check` to `OFF`), the table-level checking described in this section still applies.

Users are *strongly* urged not to use database files that contain Barracuda file format tables with releases of InnoDB older than the MySQL 5.1 with the InnoDB Plugin. It is possible to “downgrade” such tables to the Antelope format with the procedure described in Section 14.2.8.4, “Downgrading the File Format”.

14.2.8.3 Identifying the File Format in Use

After you enable a given `innodb_file_format`, this change applies only to newly created tables rather than existing ones. If you do create a new table, the tablespace containing the table is tagged with the “earliest” or “simplest” file format that is required for the table's features. For example, if you enable file format Barracuda, and create a new table that is not compressed and does not use `ROW_FORMAT=DYNAMIC`, the new tablespace that contains the table is tagged as using file format Antelope.

It is easy to identify the file format used by a given tablespace or table. The table uses the Barracuda format if the `Row_format` reported by `SHOW CREATE TABLE` or `INFORMATION_SCHEMA.TABLES` is one of `'Compressed'` or `'Dynamic'`. (The `Row_format` is a separate column; ignore the contents of the `Create_options` column, which may contain the string `ROW_FORMAT`.) If the table in a tablespace uses neither of those features, the file uses the format supported by prior releases of InnoDB, now called file format Antelope. Then, the `Row_format` is one of `'Redundant'` or `'Compact'`.

Internal Details

InnoDB has two different file formats (Antelope and Barracuda) and four different row formats (Redundant, Compact, Dynamic, and Compressed). The Antelope file format contains Redundant and Compact row formats. A tablespace that uses the Barracuda file format uses either the Dynamic or Compressed row format.

File and row format information is written in the tablespace flags (a 32-bit number) in the `*.ibd` file in the 4 bytes starting at position 54 of the file, most significant byte first (the first byte of the file is byte zero). On some systems, you can display these bytes in hexadecimal with the command `od -t x1 -j 54 -N 4 tablename.ibd`. If all bytes are zero, the tablespace uses the Antelope file format, which is the format used by the standard InnoDB storage engine up to version 5.1. The system tablespace will always have zero in the tablespace flags.

The first 10 bits of the tablespace flags can be described this way:

- Bit 0: Zero for Antelope, and bits 1 to 5 will also be zero. One for Barracuda, and bits 1 to 5 may be set.
- Bits 1 to 4: A 4 bit number representing the compressed page size. 0 = not compressed, 1 = 1k, 2 = 2k, 3 = 4k, 4 = 8k.
- Bit 5: Same value as Bit 0, zero for Antelope, one for Barracuda. If bits 0 and 5 are set and bits 1 to 4 are not, the row format is Dynamic.
- Bits 6 to 9: A 4-bit number indicating the physical page size of the tablespace. 0=16k (original default), 3=4k, 4=8k, 5=16k. These are the only valid values for My SQL 5.6 and later.
- Bit 10: Tablespace location. 0 = default, 1 = used `DATA DIRECTORY` in `CREATE TABLE` to choose the tablespace location.

Note

Tablespace flags are similar to table flags found in the InnoDB dictionary table, “`SYS_TABLES`”. They differ in the meaning of bit 0 and bits 6 to 10. Table flags will set bit 0 to one if the row format of a particular table is “Compact”. Tablespace flags cannot do that since the system tablespace can contain both Redundant and Compact row formats. So, for tablespace flags, bit 0 and bit 5 are always the same value.

Table flags can be viewed by issuing the command:

```
SELECT * FROM INFORMATION_SCHEMA.INNODB_SYS_TABLES;
```

The first 7 bits of the table flags can be described this way:

- Bit 0: Zero for Redundant row format, and bits 1 to 5 will be zero. One for Compact row format, and bits 1 to 5 may be set.
- Bits 1 to 4: A 4 bit number representing the compressed page size. 0 = not compressed, 1 = 1k, 2 = 2k, 3 = 4k, 4 = 8k.
- Bit 5: Zero for Antelope file format, and one for Barracuda file format. If bit 5 is set and bits 1 to 4 are not, the row format is Dynamic. Also, if bit 5 is set, bit 0 must also be set.
- Bit 6: Tablespace location. 0 = default, 1 = `DATA DIRECTORY` was used in `CREATE TABLE` to choose a tablespace location.

If bits 7 to 31 are not zero, the table is corrupt or the `SYS_TABLES` record is corrupt, and the table cannot be used.

14.2.8.4 Downgrading the File Format

Each InnoDB tablespace file (with a name matching *.ibd) is tagged with the file format used to create its table and indexes. The way to downgrade the tablespace is to re-create the table and its indexes. The easiest way to recreate a table and its indexes is to use the command:

```
ALTER TABLE t ROW_FORMAT=COMPACT;
```

on each table that you want to downgrade. The COMPACT row format uses the file format Antelope. It was introduced in MySQL 5.0.3.

14.2.8.5 Future InnoDB File Formats

The file format used by the standard built-in InnoDB in MySQL 5.1 is the Antelope format. The file format introduced with InnoDB Plugin 1.0 is the Barracuda format. Although no new features have been announced that would require additional new file formats, the InnoDB file format mechanism allows for future enhancements.

For the sake of completeness, these are the file format names that might be used for future file formats: Antelope, Barracuda, Cheetah, Dragon, Elk, Fox, Gazelle, Hornet, Impala, Jaguar, Kangaroo, Leopard, Moose, Nautilus, Ocelot, Porpoise, Quail, Rabbit, Shark, Tiger, Urchin, Viper, Whale, Xenops, Yak and Zebra. These file formats correspond to the internal identifiers 0..25.

14.2.9 InnoDB Row Storage and Row Formats

This section discusses how certain InnoDB features, such as table compression and off-page storage of long columns, are controlled by the ROW_FORMAT clause of the CREATE TABLE statement. It discusses considerations for choosing the right row format and compatibility of row formats between MySQL releases.

14.2.9.1 Overview of InnoDB Row Storage

The storage for rows and associated columns affects performance for queries and DML operations. As more rows fit into a single disk page, queries and index lookups can work faster, less cache memory is required in the InnoDB buffer pool, and less I/O is required to write out updated values for the numeric and short string columns.

The data in each InnoDB table is divided into pages. The pages that make up each table are arranged in a tree data structure called a B-tree index. Table data and secondary indexes both use this type of structure. The B-tree index that represents an entire table is known as the clustered index, which is organized according to the primary key columns. The nodes of the index data structure contain the values of all the columns in that row (for the clustered index) or the index columns and the primary key columns (for secondary indexes).

Variable-length columns are an exception to this rule. Columns such as BLOB and VARCHAR that are too long to fit on a B-tree page are stored on separately allocated disk pages called overflow pages. We call such columns off-page columns. The values of these columns are stored in singly-linked lists of overflow pages, and each such column has its own list of one or more overflow pages. In some cases, all or a prefix of the long column value is stored in the B-tree, to avoid wasting storage and eliminating the need to read a separate page.

The next section describes the clauses you can use with the CREATE TABLE and ALTER TABLE statements to control how these variable-length columns are represented: ROW_FORMAT and KEY_BLOCK_SIZE. To use these clauses, you might also need to change the settings for the innodb_file_per_table and innodb_file_format configuration options.

14.2.9.2 Specifying the Row Format for a Table

You specify the row format for a table with the ROW_FORMAT clause of the CREATE TABLE and ALTER TABLE statements. For example:

```
CREATE TABLE t1 (f1 int unsigned) ROW_FORMAT=DYNAMIC ENGINE=INNODB;
```

InnoDB ROW_FORMAT options include COMPACT, REDUNDANT, DYNAMIC, and COMPRESSED. For InnoDB tables, rows are stored in COMPACT format (ROW_FORMAT=COMPACT) by default. Refer to the CREATE TABLE documentation for additional information about the ROW_FORMAT table option.

The physical row structure for InnoDB tables is dependant on the ROW_FORMAT that you specify. See Physical Row Structure for more information.

```
  CREATE TABLE t1 (f1 int unsigned) ROW_FORMAT=DYNAMIC ENGINE=INNODB;
```

InnoDB ROW_FORMAT options include COMPACT, REDUNDANT, DYNAMIC, and COMPRESSED. For InnoDB tables, rows are stored in COMPACT format (ROW_FORMAT=COMPACT) by default. Refer to the CREATE TABLE documentation for additional information about the ROW_FORMAT table option.

The physical row structure for InnoDB tables is dependant on the ROW_FORMAT that you specify. See Physical Row Structure for more information.

14.2.9.3 DYNAMIC and COMPRESSED Row Formats

This section discusses the DYNAMIC and COMPRESSED row formats for InnoDB tables. You can only create these kinds of tables when the innodb_file_format configuration option is set to Barracuda. (The Barracuda file format also allows the COMPACT and REDUNDANT row formats.)

When a table is created with ROW_FORMAT=DYNAMIC or ROW_FORMAT=COMPRESSED, long column values are stored fully off-page, and the clustered index record contains only a 20-byte pointer to the overflow page.

Whether any columns are stored off-page depends on the page size and the total size of the row. When the row is too long, InnoDB chooses the longest columns for off-page storage until the clustered index record fits on the B-tree page.

The DYNAMIC row format maintains the efficiency of storing the entire row in the index node if it fits (as do the COMPACT and REDUNDANT formats), but this new format avoids the problem of filling B-tree nodes with a large number of data bytes of long columns. The DYNAMIC format is based on the idea that if a portion of a long data value is stored off-page, it is usually most efficient to store all of the value off-page. With DYNAMIC format, shorter columns are likely to remain in the B-tree node, minimizing the number of overflow pages needed for any given row.

The COMPRESSED row format uses similar internal details for off-page storage as the DYNAMIC row format, with additional storage and performance considerations from the table and index data being compressed and using smaller page sizes. With the COMPRESSED row format, the option KEY_BLOCK_SIZE controls how much column data is stored in the clustered index, and how much is placed on overflow pages. For full details about the COMPRESSED row format, see Section 14.2.7, “InnoDB Compressed Tables”.

14.2.9.4 COMPACT and REDUNDANT Row Formats

Early versions of InnoDB used an unnamed file format (now called Antelope) for database files. With that file format, tables are defined with ROW_FORMAT=COMPACT or ROW_FORMAT=REDUNDANT. InnoDB stores up to the first 768 bytes of variable-length columns (such as BLOB and VARCHAR) in the index record within the B-tree node, with the remainder stored on the overflow pages.

To preserve compatibility with those prior versions, tables created with the newest InnoDB default to the COMPACT row format. See Section 14.2.9.3, “DYNAMIC and COMPRESSED Row Formats” for information about the newer DYNAMIC and COMPRESSED row formats.

With the Antelope file format, if the value of a column is 768 bytes or less, no overflow page is needed, and some savings in I/O may result, since the value is in the B-tree node. This works well for relatively short BLOBs, but may cause B-tree nodes to fill with data rather than key values, reducing their efficiency. Tables with many BLOB columns could cause B-tree nodes to become too full of data, and contain too few rows, making the entire index less efficient than if the rows were shorter or if the column values were stored off-page.

14.2.10 InnoDB Disk I/O and File Space Management

As a DBA, you must manage disk I/O to keep the I/O subsystem from becoming saturated, and manage disk space to avoid filling up storage devices. The ACID design model requires a certain amount of I/O that might seem redundant, but helps to ensure data reliability. Within these constraints, InnoDB tries to optimize the database work and the organization of disk files to minimize the amount of disk I/O. Sometimes, I/O is postponed until the database is not busy, or until everything needs to be brought to a consistent state, such as during a database restart after a fast shutdown.

This section discusses the main considerations for I/O and disk space with the default kind of MySQL tables (also known as InnoDB tables):

- Controlling the amount of background I/O used to improve query performance.
- Enabling or disabling features that provide extra durability at the expense of additional I/O.
- Organizing tables into many small files, a few larger files, or a combination of both.
- Balancing the size of redo log files against the I/O activity that occurs when the log files become full.
- How to reorganize a table for optimal query performance.

14.2.10.1 InnoDB Disk I/O

InnoDB uses asynchronous disk I/O where possible, by creating a number of threads to handle I/O operations, while permitting other database operations to proceed while the I/O is still in progress. On Linux and Windows platforms, InnoDB uses the available OS and library functions to perform “native” asynchronous I/O. On other platforms, InnoDB still uses I/O threads, but the threads may actually wait for I/O requests to complete; this technique is known as “simulated” asynchronous I/O.

Read-Ahead

If InnoDB can determine there is a high probability that data might be needed soon, it performs read-ahead operations to bring that data into the buffer pool so that it is available in memory. Making a few large read requests for contiguous data can be more efficient than making several small, spread-out requests. There are two read-ahead heuristics in InnoDB:

- In sequential read-ahead, if InnoDB notices that the access pattern to a segment in the tablespace is sequential, it posts in advance a batch of reads of database pages to the I/O system.

- In random read-ahead, if InnoDB notices that some area in a tablespace seems to be in the process of being fully read into the buffer pool, it posts the remaining reads to the I/O system.

Doublewrite Buffer

InnoDB uses a novel file flush technique involving a structure called the doublewrite buffer. It adds safety to recovery following an operating system crash or a power outage, and improves performance on most varieties of Unix by reducing the need for fsync() operations.

Before writing pages to a data file, InnoDB first writes them to a contiguous tablespace area called the doublewrite buffer. Only after the write and the flush to the doublewrite buffer has completed does InnoDB write the pages to their proper positions in the data file. If the operating system crashes in the middle of a page write (causing a torn page condition), InnoDB can later find a good copy of the page from the doublewrite buffer during recovery.

14.2.10.2 File Space Management

The data files that you define in the configuration file form the InnoDB system tablespace. The files are logically concatenated to form the tablespace. There is no striping in use. Currently, you cannot define where within the tablespace your tables are allocated. In a newly created tablespace, InnoDB allocates space starting from the first data file.

To avoid the issues that come with storing all tables and indexes inside the system tablespace, you can turn on the innodb_file_per_table configuration option, which stores each newly created table in a separate tablespace file (with extension .ibd). For tables stored this way, there is less fragmentation within the disk file, and when the table is truncated, the space is returned to the operating system rather than still being reserved by InnoDB within the system tablespace.

Pages, Extents, Segments, and Tablespaces

Each tablespace consists of database pages. Every tablespace in a MySQL instance has the same page size. By default, all tablespaces have a page size of 16KB; you can reduce the page size to 8KB or 4KB by specifying the innodb_page_size option when you create the MySQL instance.

The pages are grouped into extents of size 1MB (64 consecutive 16KB pages, or 128 8KB pages, or 256 4KB pages). The “files” inside a tablespace are called segments in InnoDB. (These segments are different from the rollback segment, which actually contains many tablespace segments.)

When a segment grows inside the tablespace, InnoDB allocates the first 32 pages to it one at a time. After that, InnoDB starts to allocate whole extents to the segment. InnoDB can add up to 4 extents at a time to a large segment to ensure good sequentiality of data.

Two segments are allocated for each index in InnoDB. One is for nonleaf nodes of the B-tree, the other is for the leaf nodes. Keeping the leaf nodes contiguous on disk enables better sequential I/O operations, because these leaf nodes contain the actual table data.

Some pages in the tablespace contain bitmaps of other pages, and therefore a few extents in an InnoDB tablespace cannot be allocated to segments as a whole, but only as individual pages.

When you ask for available free space in the tablespace by issuing a SHOW TABLE STATUS statement, InnoDB reports the extents that are definitely free in the tablespace. InnoDB always reserves some extents for cleanup and other internal purposes; these reserved extents are not included in the free space.

When you delete data from a table, InnoDB contracts the corresponding B-tree indexes. Whether the freed space becomes available for other users depends on whether the pattern of deletes frees individual pages or extents to the tablespace. Dropping a table or deleting all rows from it is guaranteed to release the space to other users, but remember that deleted rows are physically removed only by the purge operation, which happens automatically some time after they are no longer needed for transaction rollbacks or consistent reads. (See Section 14.2.2.12, "InnoDB Multi-Versioning".)

To see information about the tablespace, use the Tablespace Monitor. See Section 14.2.12.4, "InnoDB Monitors".

How Pages Relate to Table Rows

The maximum row length, except for variable-length columns (VARBINARY, VARCHAR, BLOB and TEXT), is slightly less than half of a database page. That is, the maximum row length is about 8000 bytes. LONGBLOB and LONGTEXT columns must be less than 4GB, and the total row length, including BLOB and TEXT columns, must be less than 4GB.

If a row is less than half a page long, all of it is stored locally within the page. If it exceeds half a page, variable-length columns are chosen for external off-page storage until the row fits within half a page. For a column chosen for off-page storage, InnoDB stores the first 768 bytes locally in the row, and the rest externally into overflow pages. Each such column has its own list of overflow pages. The 768-byte prefix is accompanied by a 20-byte value that stores the true length of the column and points into the overflow list where the rest of the value is stored.

14.2.10.3 InnoDB Checkpoints

Making your log files very large may reduce disk I/O during checkpointing. It often makes sense to set the total size of the log files as large as the buffer pool or even larger. Although in the past large log files could make crash recovery take excessive time, starting with MySQL 5.5, performance enhancements to crash recovery make it possible to use large log files with fast startup after a crash. (Strictly speaking, this performance improvement is available for MySQL 5.1 with the InnoDB Plugin 1.0.7 and higher. It is with MySQL 5.5 that this improvement is available in the default InnoDB storage engine.)

How Checkpoint Processing Works

InnoDB implements a checkpoint mechanism known as fuzzy checkpointing. InnoDB flushes modified database pages from the buffer pool in small batches. There is no need to flush the buffer pool in one single batch, which would disrupt processing of user SQL statements during the checkpointing process.

During crash recovery, InnoDB looks for a checkpoint label written to the log files. It knows that all modifications to the database before the label are present in the disk image of the database. Then InnoDB scans the log files forward from the checkpoint, applying the logged modifications to the database.

14.2.10.4 Defragmenting a Table

Random insertions into or deletions from a secondary index can cause the index to become fragmented. Fragmentation means that the physical ordering of the index pages on the disk is not close to the index ordering of the records on the pages, or that there are many unused pages in the 64-page blocks that were allocated to the index.

One symptom of fragmentation is that a table takes more space than it "should" take. How much that is exactly, is difficult to determine. All InnoDB data and indexes are stored in B-trees, and their fill factor may vary from 50% to 100%. Another symptom of fragmentation is that a table scan such as this takes more time than it "should" take:

```
SELECT COUNT(*) FROM t WHERE non_indexed_column <> 12345;
```

The preceding query requires MySQL to perform a full table scan, the slowest type of query for a large table.

To speed up index scans, you can periodically perform a “null” `ALTER TABLE` operation, which causes MySQL to rebuild the table:

```
ALTER TABLE tbl_name ENGINE=INNODB
```

You can also use `ALTER TABLE tbl_name FORCE` to perform a “null” alter operation that rebuilds the table.

As of MySQL 5.7.4, both `ALTER TABLE tbl_name ENGINE=INNODB` and `ALTER TABLE tbl_name FORCE` use online DDL (`ALGORITHM=COPY`). For more information, see Section 14.2.11.1, “Overview of Online DDL”.

Another way to perform a defragmentation operation is to use `mysqldump` to dump the table to a text file, drop the table, and reload it from the dump file.

Another way to perform a defragmentation operation is to use `mysqldump` to dump the table to a text file, drop the table, and reload it from the dump file.

If the insertions into an index are always ascending and records are deleted only from the end, the `InnoDB` filespace management algorithm guarantees that fragmentation in the index does not occur.

14.2.10.5 Reclaiming Disk Space with `TRUNCATE TABLE`

To reclaim operating system disk space when truncating an `InnoDB` table, the table must be stored in its own .ibd file. For a table to be stored in its own .ibd file, `innodb_file_per_table` must enabled when the table is created. Additionally, there cannot be a foreign key constraint between the table being truncated and other tables, otherwise the `TRUNCATE TABLE` operation fails. A foreign key constraint between two columns in the same table, however, is permitted.

When a table is truncated, it is dropped and re-created in a new `.ibd` file, and the freed space is returned to the operating system. This is in contrast to truncating `InnoDB` tables that are stored within the `InnoDB` system tablespace (tables created when `innodb_file_per_table=OFF`), where only `InnoDB` can use the freed space after the table is truncated.

The ability to truncate tables and return disk space to the operating system also means that physical backups can be smaller. Truncating tables that are stored in the system tablespace (tables created when `innodb_file_per_table=OFF`) leaves blocks of unused space in the system tablespace.

14.2.11 `InnoDB` and Online DDL

You can perform several kinds of online DDL operations on `InnoDB` tables: that is, allowing DML operations and queries on the table while the DDL is in progress, performing the operation “in-place” without rebuilding the entire table, or both. This enhancement has the following benefits:

- It improves responsiveness and availability in busy production environments, where making a table unavailable for minutes or hours whenever you modify its indexes or column definitions is not practical.

- It lets you adjust the balance between performance and concurrency during the DDL operation, by choosing whether to block access to the table entirely (`LOCK=EXCLUSIVE` clause), allow queries but not DML (`LOCK=SHARED` clause), or allow full query and DML access to the table (`LOCK=NONE` clause).

When you omit the LOCK clause or specify LOCK=DEFAULT, MySQL allows as much concurrency as possible depending on the type of operation.

- By doing the changes in-place where possible, rather than creating a new copy of the table, it avoids temporary increases in disk space usage and the I/O overhead of copying the table and reconstructing all the secondary indexes.

14.2.11.1 Overview of Online DDL

Historically, many DDL operations on InnoDB tables were expensive. Many ALTER TABLE operations worked by creating a new, empty table defined with the requested table options and indexes, then copying the existing rows to the new table one-by-one, updating the indexes as the rows were inserted. After all rows from the original table were copied, the old table was dropped and the copy was renamed with the name of the original table.

MySQL 5.5, and MySQL 5.1 with the InnoDB Plugin, optimized CREATE INDEX and DROP INDEX to avoid the table-copying behavior. That feature was known as Fast Index Creation. MySQL 5.6 enhances many other types of ALTER TABLE operations to avoid copying the table. Another enhancement allows SELECT queries and INSERT, UPDATE, and DELETE (DML) statements to proceed while the table is being altered. In MySQL 5.7, ALTER TABLE RENAME INDEX was also enhanced to avoid table copying. This combination of features is now known as online DDL.

This new mechanism also means that you can generally speed the overall process of creating and loading a table and associated indexes by creating the table without any secondary indexes, then adding the secondary indexes after the data is loaded.

Although no syntax changes are required in the CREATE INDEX or DROP INDEX commands, some factors affect the performance, space usage, and semantics of this operation (see Section 14.2.11.9, “Limitations of Online DDL”).

The online DDL enhancements in MySQL 5.6 improve many DDL operations that formerly required a table copy, blocked DML operations on the table, or both. Table 14.5, “Summary of Online Status for DDL Operations” shows the variations of the ALTER TABLE statement and shows how the online DDL feature applies to each one.

With the exception of ALTER TABLE partitioning clauses, online DDL operations for partitioned InnoDB tables follow the same rules that apply to regular InnoDB tables. For more information, see Section 14.2.11.8, “Online DDL for Partitioned InnoDB Tables”.

- The “In-Place?” column shows which operations allow the ALGORITHM=INPLACE clause; the preferred value is “Yes”.

- The “Copies Table?” column shows which operations are able to avoid the expensive table-copying operation; the preferred value is “No”. This column is mostly the reverse of the “In-Place?” column, except that a few operations allow ALGORITHM=INPLACE but still involve some amount of table copying.

- The “Allows Concurrent DML?” column shows which operations can be performed fully online; the preferred value is “Yes”. You can specify LOCK=NONE to assert that full concurrency is allowed during the DDL, but MySQL automatically allows this level of concurrency when possible. When concurrent DML is allowed, concurrent queries are also always allowed.

- The “Allows Concurrent Queries?” column shows which DDL operations allow queries on the table while the operation is in progress; the preferred value is “Yes”. Concurrent query is allowed during all online DDL operations. It is shown with “Yes” listed for all cells, for reference purposes. You can specify LOCK=SHARED to assert that concurrent queries are allowed during the DDL, but MySQL automatically allows this level of concurrency when possible.

- The “Notes” column explains any exceptions to the “yes/no” values of the other columns, such as when the answer depends on the setting of a configuration option or some other clause in the DDL statement. The values “Yes*” and “No*” indicate that an answer depends on these additional notes.

Table 14.5 Summary of Online Status for DDL Operations

Operation	In-Place?	Copies Table?	Allows Concurrent DML?	Allows Concurrent Query?	Notes
CREATE INDEX, ADD INDEX	Yes*	No*	Yes	Yes	Some restrictions for FULLTEXT index; see next row. Currently, the operation is not in-place (that is, it copies the table) if the same index being created was also dropped by an earlier clause in the same ALTER TABLE statement.
ADD FULLTEXT INDEX	Yes	No*	No	Yes	Creating the first FULLTEXT index for a table involves a table copy, unless there is a user-supplied FTS_DOC_ID column. Subsequent FULLTEXT indexes on the same table can be created in-place.
RENAME INDEX	Yes	No	No	No	
DROP INDEX	Yes	No	Yes	Yes	
OPTIMIZE TABLE	Yes	Yes	Yes	Yes	Uses ALGORITHM=INPLACE as of MySQL 5.7.4. ALGORITHM=COPY is used if old_alter_table=1 or mysqld --skip-new option is enabled. OPTIMIZE TABLE using online DDL (ALGORITHM=INPLACE) is not supported for tables with FULLTEXT indexes.
Set default value for a column	Yes	No	Yes	Yes	Modifies .frm file only, not the data file.
Change auto-increment value for a column	Yes	No	Yes	Yes	Modifies a value stored in memory, not the data file.
Add a foreign key constraint	Yes*	No*	Yes	Yes	To avoid copying the table, disable foreign_key_checks during constraint creation.
Drop a foreign key constraint	Yes	No	Yes	Yes	The foreign_key_checks option can be enabled or disabled.
Rename a column	Yes*	No*	Yes*	Yes	To allow concurrent DML, keep the same data type and only change the column name.
Add a column	Yes	Yes	Yes*	Yes	Concurrent DML is not allowed when adding an auto-increment column. Although ALGORITHM=INPLACE is allowed, the data is reorganized

Operation	In-Place?	Copies Table?	Allows Concurrent DML?	Allows Concurrent Query?	Notes
					substantially, so it is still an expensive operation.
Drop a column	Yes	Yes	Yes	Yes	Although ALGORITHM=INPLACE is allowed, the data is reorganized substantially, so it is still an expensive operation.
Reorder columns	Yes	Yes	Yes	Yes	Although ALGORITHM=INPLACE is allowed, the data is reorganized substantially, so it is still an expensive operation.
Change ROW_FORMAT property	Yes	Yes	Yes	Yes	Although ALGORITHM=INPLACE is allowed, the data is reorganized substantially, so it is still an expensive operation.
Change KEY_BLOCK_SIZE property	Yes	Yes	Yes	Yes	Although ALGORITHM=INPLACE is allowed, the data is reorganized substantially, so it is still an expensive operation.
Make column NULL	Yes	Yes	Yes	Yes	Although ALGORITHM=INPLACE is allowed, the data is reorganized substantially, so it is still an expensive operation.
Make column NOT NULL	Yes*	Yes	Yes	Yes	When SQL_MODE includes strict_all_tables or strict_all_tables, the operation fails if the column contains any nulls. Although ALGORITHM=INPLACE is allowed, the data is reorganized substantially, so it is still an expensive operation.
Change data type of column	No*	Yes*	No	Yes	Exception: VARCHAR size may be increased using online ALTER TABLE. See InnoDB Online DDL Column Properties for more information.
Add primary key	Yes*	Yes	Yes	Yes	Although ALGORITHM=INPLACE is allowed, the data is reorganized substantially, so it is still an expensive operation. ALGORITHM=INPLACE is not allowed under certain conditions if columns have to be converted to NOT NULL. See Example 14.9, “Creating and Dropping the Primary Key”.

Operation	In-Place?	Copies Table?	Allows Concurrent DML?	Allows Concurrent Query?	Notes
Drop primary key and add another	Yes	Yes	Yes	Yes	ALGORITHM=INPLACE is only allowed when you add a new primary key in the same ALTER TABLE; the data is reorganized substantially, so it is still an expensive operation.
Drop primary key	No	Yes	No	Yes	Restrictions apply when you drop a primary key primary key without adding a new one in the same ALTER TABLE statement.
Convert character set	No	Yes	No	Yes	Rebuilds the table if the new character encoding is different.
Specify character set	No	Yes	No	Yes	Rebuilds the table if the new character encoding is different.
Rebuild with FORCE option	Yes	Yes	Yes	Yes	Uses ALGORITHM=INPLACE as of MySQL 5.6.17. ALGORITHM=COPY is used if old_alter_table=1 or mysqld --skip-new option is enabled. Table rebuild using online DDL (ALGORITHM=INPLACE) is not supported for tables with FULLTEXT indexes.
Rebuild with “null” ALTER TABLE ... ENGINE=INNODB	Yes	Yes	Yes	Yes	Uses ALGORITHM=INPLACE as of MySQL 5.7.4. ALGORITHM=COPY is used if old_alter_table=1 or mysqld --skip-new option is enabled. Table rebuild using online DDL (ALGORITHM=INPLACE) is not supported for tables with FULLTEXT indexes.

The following sections shows the basic syntax, and usage notes related to online DDL, for each of the major operations that can be performed with concurrent DML, in-place, or both:

Secondary Indexes

- Create secondary indexes: `CREATE INDEX name ON table (col_list)` or `ALTER TABLE table ADD INDEX name (col_list)`. (Creating a a FULLTEXT index still requires locking the table.)
- Drop secondary indexes: `DROP INDEX name ON table;` or `ALTER TABLE table DROP INDEX name`

Creating and dropping secondary indexes on InnoDB tables skips the table-copying behavior, the same as in MySQL 5.5 and MySQL 5.1 with the InnoDB Plugin.

In MySQL 5.6 and higher, the table remains available for read and write operations while the index is being created or dropped. The CREATE INDEX or DROP INDEX statement only finishes after all transactions that are accessing the table are completed, so that the initial state of the index reflects the most recent contents of the table. Previously, modifying the table while an index was being created or dropped typically resulted in a deadlock that cancelled the INSERT, UPDATE, or DELETE statement on the table.

Column Properties

- Set a default value for a column: `ALTER TABLE tbl ALTER COLUMN col SET DEFAULT literal` or `ALTER TABLE tbl ALTER COLUMN col DROP DEFAULT`

  The default values for columns are stored in the .frm file for the table, not the `InnoDB` data dictionary.

- Changing the auto-increment value for a column: `ALTER TABLE table AUTO_INCREMENT=next_value;`

  Especially in a distributed system using replication or sharding, you sometimes reset the auto-increment counter for a table to a specific value. The next row inserted into the table uses the specified value for its auto-increment column. You might also use this technique in a data warehousing environment where you periodically empty all the tables and reload them, and you can restart the auto-increment sequence from 1.

- Renaming a column: `ALTER TABLE tbl CHANGE old_col_name new_col_name datatype`

  When you keep the same data type and `[NOT] NULL` attribute, only changing the column name, this operation can always be performed online.

  As part of this enhancement, you can now rename a column that is part of a foreign key constraint, which was not allowed before. The foreign key definition is automatically updated to use the new column name. Renaming a column participating in a foreign key only works with the in-place mode of `ALTER TABLE`. If you use the `ALGORITHM=COPY` clause, or some other condition causes the command to use `ALGORITHM=COPY` behind the scenes, the `ALTER TABLE` statement will fail.

- Extending `VARCHAR` size using an in-place `ALTER TABLE` statement, as in this example:

  ```
  ALTER TABLE t1 ALGORITHM=INPLACE, CHANGE COLUMN c1 c1 VARCHAR(255);
  ```

  The number of length bytes required by a `VARCHAR` column must remain the same. For `VARCHAR` values of 0 to 255, one length byte is required to encode the value. For `VARCHAR` values of 256 bytes or more, two length bytes are required. As a result, in-place `ALTER TABLE` only supports increasing `VARCHAR` size from 0 to 255 bytes or increasing `VARCHAR` size from a value equal to or greater than 256 bytes. In-place `ALTER TABLE` does not support increasing `VARCHAR` size from less than 256 bytes to a value equal to or greater than 256 bytes. In this case, the number of required length bytes would change from 1 to 2, which is only supported by a table copy (`ALGORITHM=COPY`). For example, attempting to change `VARCHAR` column size from 255 to 256 using in-place `ALTER TABLE` would return an error:

  ```
  ALTER TABLE t1 ALGORITHM=INPLACE, CHANGE COLUMN c1 c1 VARCHAR(256);
  ERROR 0A000: ALGORITHM=INPLACE is not supported. Reason: Cannot change
  column type INPLACE. Try ALGORITHM=COPY.
  ```

  Decreasing `VARCHAR` size using in-place `ALTER TABLE` is not supported. Decreasing `VARCHAR` size requires a table copy (`ALGORITHM=COPY`).

Foreign Keys

- Adding or dropping a foreign key constraint:

  ```
  ALTER TABLE tbl1 ADD CONSTRAINT fk_name FOREIGN KEY index (col1) REFERENCES tbl2(col2) referential_actio
  ALTER TABLE tbl DROP FOREIGN KEY fk_name;
  ```

  Dropping a foreign key can be performed online with the `foreign_key_checks` option enabled or disabled. Creating a foreign key online requires `foreign_key_checks` to be disabled.

If you do not know the names of the foreign key constraints on a particular table, issue the following statement and find the constraint name in the CONSTRAINT clause for each foreign key:

```
show create table table\G
```

Or, query the information_schema.table_constraints table and use the constraint_name and constraint_type columns to identify the foreign key names.

As a consequence of this enhancement, you can now also drop a foreign key and its associated index in a single statement, which previously required separate statements in a strict order:

```
ALTER TABLE table DROP FOREIGN KEY constraint, DROP INDEX index;
```

If foreign keys are already present in the table being altered (that is, it is a child table containing any FOREIGN KEY ... REFERENCE clauses), additional restrictions apply to online DDL operations, even those not directly involving the foreign key columns:

- Concurrent DML is disallowed during online DDL operations on such child tables. (This restriction is being evaluated as a bug and might be lifted.)

- An ALTER TABLE on the child table could also wait for another transaction to commit, if a change to the parent table caused associated changes in the child table through an ON UPDATE or ON DELETE clause using the CASCADE or SET NULL parameters.

In the same way, if a table is the parent table in a foreign key relationship, even though it does not contain any FOREIGN KEY clauses, it could wait for the ALTER TABLE to complete if an INSERT, UPDATE, or DELETE statement caused an ON UPDATE or ON DELETE action in the child table.

Notes on ALGORITHM=COPY

Any ALTER TABLE operation run with the ALGORITHM=COPY clause prevents concurrent DML operations. Concurrent queries are still allowed. That is, a table-copying operation always includes at least the concurrency restrictions of LOCK=SHARED (allow queries but not DML). You can further restrict concurrency for such operations by specifying LOCK=EXCLUSIVE (prevent DML and queries).

Concurrent DML but Table Copy Still Required

Some other ALTER TABLE operations allow concurrent DML but still require a table copy. However, the table copy for these operations is faster than it was in MySQL 5.5 and prior.

- Adding, dropping, or reordering columns.

- Adding or dropping a primary key.

- Changing the ROW_FORMAT or KEY_BLOCK_SIZE properties for a table.

- Changing the nullable status for a column.

- OPTIMIZE TABLE

- Rebuilding a table with the FORCE option

- Rebuilding a table using a “null” ALTER TABLE ... ENGINE=INNODB statement

Note

As your database schema evolves with new columns, data types, constraints, indexes, and so on, keep your CREATE TABLE statements up to date with the latest table definitions. Even with the performance improvements of online DDL, it is more efficient to create stable database structures at the beginning, rather than creating part of the schema and then issuing ALTER TABLE statements afterward.

The main exception to this guideline is for secondary indexes on tables with large numbers of rows. It is typically most efficient to create the table with all details specified except the secondary indexes, load the data, then create the secondary indexes. You can use the same technique with foreign keys (load the data first, then set up the foreign keys) if you know the initial data is clean and do not need consistency checks during the loading process.

Whatever sequence of CREATE TABLE, CREATE INDEX, ALTER TABLE, and similar statements went into putting a table together, you can capture the SQL needed to reconstruct the current form of the table by issuing the statement SHOW CREATE TABLE *table*\G (uppercase \G required for tidy formatting). This output shows clauses such as numeric precision, NOT NULL, and CHARACTER SET that are sometimes added behind the scenes, and you might otherwise leave out when cloning the table on a new system or setting up foreign key columns with identical type.

14.2.11.2 Performance and Concurrency Considerations for Online DDL

Online DDL improves several aspects of MySQL operation, such as performance, concurrency, availability, and scalability:

- Because queries and DML operations on the table can proceed while the DDL is in progress, applications that access the table are more responsive. Reduced locking and waiting for other resources all throughout the MySQL server leads to greater scalability, even for operations not involving the table being altered.
- For in-place operations, by avoiding the disk I/O and CPU cycles to rebuild the table, you minimize the overall load on the database and maintain good performance and high throughput during the DDL operation.
- For in-place operations, because less data is read into the buffer pool than if all the data was copied, you avoid purging frequently accessed data from memory, which formerly could cause a temporary performance dip after a DDL operation.

If an online operation requires temporary files, InnoDB creates them in the temporary file directory, not the directory containing the original table. If this directory is not large enough to hold such files, you may need to set the tmpdir system variable to a different directory. (See Section C.5.4.4, “Where MySQL Stores Temporary Files”.)

Locking Options for Online DDL

While an InnoDB table is being changed by a DDL operation, the table may or may not be locked, depending on the internal workings of that operation and the LOCK clause of the ALTER TABLE statement. By default, MySQL uses as little locking as possible during a DDL operation; you specify the clause either to make the locking more restrictive than it normally would be (thus limiting concurrent DML, or DML and queries), or to ensure that some expected degree of locking is allowed for an operation. If the LOCK clause specifies a level of locking that is not available for that specific kind of DDL operation, such as

LOCK=SHARED or LOCK=NONE while creating or dropping a primary key, the clause works like an assertion, causing the statement to fail with an error. The following list shows the different possibilities for the LOCK clause, from the most permissive to the most restrictive:

- For DDL operations with LOCK=NONE, both queries and concurrent DML are allowed. This clause makes the ALTER TABLE fail if the kind of DDL operation cannot be performed with the requested type of locking, so specify LOCK=NONE if keeping the table fully available is vital and it is OK to cancel the DDL if that is not possible. For example, you might use this clause in DDLs for tables involving customer signups or purchases, to avoid making those tables unavailable by mistakenly issuing an expensive ALTER TABLE statement.

- For DDL operations with LOCK=SHARED, any writes to the table (that is, DML operations) are blocked, but the data in the table can be read. This clause makes the ALTER TABLE fail if the kind of DDL operation cannot be performed with the requested type of locking, so specify LOCK=SHARED if keeping the table available for queries is vital and it is OK to cancel the DDL if that is not possible. For example, you might use this clause in DDLs for tables in a data warehouse, where it is OK to delay data load operations until the DDL is finished, but queries cannot be delayed for long periods.

- For DDL operations with LOCK=DEFAULT, or with the LOCK clause omitted, MySQL uses the lowest level of locking that is available for that kind of operation, allowing concurrent queries, DML, or both wherever possible. This is the setting to use when making pre-planned, pre-tested changes that you know will not cause any availability problems based on the workload for that table.

- For DDL operations with LOCK=EXCLUSIVE, both queries and DML operations are blocked. This clause makes the ALTER TABLE fail if the kind of DDL operation cannot be performed with the requested type of locking, so specify LOCK=EXCLUSIVE if the primary concern is finishing the DDL in the shortest time possible, and it is OK to make applications wait when they try to access the table. You might also use LOCK=EXCLUSIVE if the server is supposed to be idle, to avoid unexpected accesses to the table.

An online DDL statement for an InnoDB table always waits for currently executing transactions that are accessing the table to commit or roll back, because it requires exclusive access to the table for a brief period while the DDL statement is being prepared. Likewise, it requires exclusive access to the table for a brief time before finishing. Thus, an online DDL statement waits for any transactions that are started while the DDL is in progress, and query or modify the table, to commit or roll back before the DDL completes.

Because there is some processing work involved with recording the changes made by concurrent DML operations, then applying those changes at the end, an online DDL operation could take longer overall than the old-style mechanism that blocks table access from other sessions. The reduction in raw performance is balanced against better responsiveness for applications that use the table. When evaluating the ideal techniques for changing table structure, consider end-user perception of performance, based on factors such as load times for web pages.

A newly created InnoDB secondary index contains only the committed data in the table at the time the CREATE INDEX or ALTER TABLE statement finishes executing. It does not contain any uncommitted values, old versions of values, or values marked for deletion but not yet removed from the old index.

Performance of In-Place versus Table-Copying DDL Operations

The raw performance of an online DDL operation is largely determined by whether the operation is performed in-place, or requires copying and rebuilding the entire table. See Table 14.5, “Summary of Online Status for DDL Operations” to see what kinds of operations can be performed in-place, and any requirements for avoiding table-copy operations.

The performance speedup from in-place DDL applies to operations on secondary indexes, not to the primary key index. The rows of an InnoDB table are stored in a clustered index organized based on the primary key, forming what some database systems call an “index-organized table”. Because the table structure is so closely tied to the primary key, redefining the primary key still requires copying the data.

When an operation on the primary key uses ALGORITHM=INPLACE, even though the data is still copied, it is more efficient than using ALGORITHM=COPY because:

- No undo logging or associated redo logging is required for ALGORITHM=INPLACE. These operations add overhead to DDL statements that use ALGORITHM=COPY.
- The secondary index entries are pre-sorted, and so can be loaded in order.
- The change buffer is not used, because there are no random-access inserts into the secondary indexes.

To judge the relative performance of online DDL operations, you can run such operations on a big InnoDB table using current and earlier versions of MySQL. You can also run all the performance tests under the latest MySQL version, simulating the previous DDL behavior for the “before” results, by setting the old_alter_table system variable. Issue the statement set old_alter_table=1 in the session, and measure DDL performance to record the “before” figures. Then set old_alter_table=0 to re-enable the newer, faster behavior, and run the DDL operations again to record the “after” figures.

For a basic idea of whether a DDL operation does its changes in-place or performs a table copy, look at the “rows affected” value displayed after the command finishes. For example, here are lines you might see after doing different types of DDL operations:

- Changing the default value of a column (super-fast, does not affect the table data at all):

  ```
  Query OK, 0 rows affected (0.07 sec)
  ```

- Adding an index (takes time, but 0 rows affected shows that the table is not copied):

  ```
  Query OK, 0 rows affected (21.42 sec)
  ```

- Changing the data type of a column (takes substantial time and does require rebuilding all the rows of the table):

  ```
  Query OK, 1671168 rows affected (1 min 35.54 sec)
  ```

  > **Note**
  >
  > Changing the data type of a column requires rebuilding all the rows of the table with the exception of changing VARCHAR size, which may be performed using online ALTER TABLE. See InnoDB Online DDL Column Properties for more information.

For example, before running a DDL operation on a big table, you might check whether the operation will be fast or slow as follows:

1. Clone the table structure.
2. Populate the cloned table with a tiny amount of data.
3. Run the DDL operation on the cloned table.
4. Check whether the “rows affected” value is zero or not. A non-zero value means the operation will require rebuilding the entire table, which might require special planning. For example, you might do the DDL operation during a period of scheduled downtime, or on each replication slave server one at a time.

For a deeper understanding of the reduction in MySQL processing, examine the performance_schema and INFORMATION_SCHEMA tables related to InnoDB before and after DDL operations, to see the number of physical reads, writes, memory allocations, and so on.

14.2.11.3 SQL Syntax for Online DDL

Typically, you do not need to do anything special to enable online DDL when using the ALTER TABLE statement for InnoDB tables. See Table 14.5, “Summary of Online Status for DDL Operations” for the kinds of DDL operations that can be performed in-place, allowing concurrent DML, or both. Some variations require particular combinations of configuration settings or ALTER TABLE clauses.

You can control the various aspects of a particular online DDL operation by using the LOCK and ALGORITHM clauses of the ALTER TABLE statement. These clauses come at the end of the statement, separated from the table and column specifications by commas. The LOCK clause is useful for fine-tuning the degree of concurrent access to the table. The ALGORITHM clause is primarily intended for performance comparisons and as a fallback to the older table-copying behavior in case you encounter any issues with existing DDL code. For example:

- To avoid accidentally making the table unavailable for reads, writes, or both, you could specify a clause on the ALTER TABLE statement such as LOCK=NONE (allow both reads and writes) or LOCK=SHARED (allow reads). The operation halts immediately if the requested level of concurrency is not available.
- To compare performance, you could run one statement with ALGORITHM=INPLACE and another with ALGORITHM=COPY, as an alternative to setting the old_alter_table configuration option.
- To avoid the chance of tying up the server by running an ALTER TABLE that copied the table, you could include ALGORITHM=INPLACE so the statement halts immediately if it cannot use the in-place mechanism. See Table 14.5, “Summary of Online Status for DDL Operations” for a list of the DDL operations that can or cannot be performed in-place.

See Section 14.2.11.2, “Performance and Concurrency Considerations for Online DDL” for more details about the LOCK clause. For full examples of using online DDL, see Section 14.2.11.5, “Examples of Online DDL”.

14.2.11.4 Combining or Separating DDL Statements

Before the introduction of online DDL, it was common practice to combine many DDL operations into a single ALTER TABLE statement. Because each ALTER TABLE statement involved copying and rebuilding the table, it was more efficient to make several changes to the same table at once, since those changes could all be done with a single rebuild operation for the table. The downside was that SQL code involving DDL operations was harder to maintain and to reuse in different scripts. If the specific changes were different each time, you might have to construct a new complex ALTER TABLE for each slightly different scenario.

For DDL operations that can be done in-place, as shown in Table 14.5, “Summary of Online Status for DDL Operations”, now you can separate them into individual ALTER TABLE statements for easier scripting and maintenance, without sacrificing efficiency. For example, you might take a complicated statement such as:

```
alter table t1 add index i1(c1), add unique index i2(c2), change c4_old_name c4_new_name integer unsigned;
```

and break it down into simpler parts that can be tested and performed independently, such as:

```
alter table t1 add index i1(c1);
alter table t1 add unique index i2(c2);
alter table t1 change c4_old_name c4_new_name integer unsigned not null;
```

You might still use multi-part ALTER TABLE statements for:

- Operations that must be performed in a specific sequence, such as creating an index followed by a foreign key constraint that uses that index.
- Operations all using the same specific LOCK clause, that you want to either succeed or fail as a group.
- Operations that cannot be performed in-place, that is, that still copy and rebuild the table.
- Operations for which you specify ALGORITHM=COPY or old_alter_table=1, to force the table-copying behavior if needed for precise backward-compatibility in specialized scenarios.

14.2.11.5 Examples of Online DDL

Here are code examples showing some operations whose performance, concurrency, and scalability are improved by the latest online DDL enhancements.

- Example 14.1, "Schema Setup Code for Online DDL Experiments" sets up tables named BIG_TABLE and SMALL_TABLE used in the subsequent examples.
- Example 14.2, "Speed and Efficiency of CREATE INDEX and DROP INDEX" illustrates the performance aspects of creating and dropping indexes.
- Example 14.3, "Concurrent DML During CREATE INDEX and DROP INDEX" shows queries and DML statements running during a DROP INDEX operation.
- Example 14.4, "Renaming a Column" demonstrates the speed improvement for renaming a column, and shows the care needed to keep the data type precisely the same when doing the rename operation.
- Example 14.5, "Dropping Foreign Keys" demonstrates how foreign keys work with online DDL. Because two tables are involved in foreign key operations, there are extra locking considerations. Thus, tables with foreign keys sometimes have restrictions for online DDL operations.
- Example 14.6, "Changing Auto-Increment Value" demonstrates how auto-increment columns work with online DDL. Tables with auto-increment columns sometimes have restrictions for online DDL operations.
- Example 14.7, "Controlling Concurrency with the LOCK Clause" demonstrates the options to permit or restrict concurrent queries and DML operations while an online DDL operation is in progress. It shows the situations when the DDL statement might wait, or the concurrent transaction might wait, or the concurrent transaction might cancel a DML statement due to a deadlock error.
- Example 14.8, "Schema Setup Code for Online DDL Experiments" demonstrates creating and dropping multiple indexes in a single statement, which can be more efficient than using a separate statement for each index operation.
- Example 14.9, "Creating and Dropping the Primary Key" demonstrates how it is more efficient to define a primary key when creating the table, and relatively expensive to add one later.

Example 14.1 Schema Setup Code for Online DDL Experiments

Here is the code that sets up the initial tables used in these demonstrations:

```
/*
Setup code for the online DDL demonstration:
- Set up some config variables.
- Create 2 tables that are clones of one of the INFORMATION_SCHEMA tables
  that always has some data. The "small" table has a couple of thousand rows.
  For the "big" table, keep doubling the data until it reaches over a million rows.
- Set up a primary key for the sample tables, since we are demonstrating InnoDB aspects.
*/

set autocommit = 0;
set foreign_key_checks = 1;
set global innodb_file_per_table = 1;
set old_alter_table=0;
prompt mysql:

use test;

\! echo "Setting up 'small' table:"
drop table if exists small_table;
create table small_table as select * from information_schema.columns;
alter table small_table add id int unsigned not null primary key auto_increment;
select count(id) from small_table;

\! echo "Setting up 'big' table:"
drop table if exists big_table;
create table big_table as select * from information_schema.columns;
show create table big_table\G

insert into big_table select * from big_table;
insert into big_table select * from big_table;
insert into big_table select * from big_table;
insert into big_table select * from big_table;
insert into big_table select * from big_table;
insert into big_table select * from big_table;
insert into big_table select * from big_table;
insert into big_table select * from big_table;
insert into big_table select * from big_table;
insert into big_table select * from big_table;
commit;

alter table big_table add id int unsigned not null primary key auto_increment;
select count(id) from big_table;
```

Running this code gives this output, condensed for brevity and with the most important points bolded:

```

Setting up 'small' table:
Query OK, 0 rows affected (0.01 sec)

Query OK, 1678 rows affected (0.13 sec)
Records: 1678  Duplicates: 0  Warnings: 0

Query OK, 1678 rows affected (0.07 sec)
Records: 1678  Duplicates: 0  Warnings: 0

+-----------+
| count(id) |
+-----------+
|      1678 |
+-----------+
1 row in set (0.00 sec)

Setting up 'big' table:
Query OK, 0 rows affected (0.16 sec)

Query OK, 1678 rows affected (0.17 sec)
Records: 1678  Duplicates: 0  Warnings: 0

*************************** 1. row ***************************
       Table: big_table
Create Table: CREATE TABLE `big_table` (
  `TABLE_CATALOG` varchar(512) CHARACTER SET utf8 NOT NULL DEFAULT '',
  `TABLE_SCHEMA` varchar(64) CHARACTER SET utf8 NOT NULL DEFAULT '',
  `TABLE_NAME` varchar(64) CHARACTER SET utf8 NOT NULL DEFAULT '',
  `COLUMN_NAME` varchar(64) CHARACTER SET utf8 NOT NULL DEFAULT '',
  `ORDINAL_POSITION` bigint(21) unsigned NOT NULL DEFAULT '0',
  `COLUMN_DEFAULT` longtext CHARACTER SET utf8,
  `IS_NULLABLE` varchar(3) CHARACTER SET utf8 NOT NULL DEFAULT '',
  `DATA_TYPE` varchar(64) CHARACTER SET utf8 NOT NULL DEFAULT '',
  `CHARACTER_MAXIMUM_LENGTH` bigint(21) unsigned DEFAULT NULL,
  `CHARACTER_OCTET_LENGTH` bigint(21) unsigned DEFAULT NULL,
  `NUMERIC_PRECISION` bigint(21) unsigned DEFAULT NULL,
  `NUMERIC_SCALE` bigint(21) unsigned DEFAULT NULL,
  `DATETIME_PRECISION` bigint(21) unsigned DEFAULT NULL,
  `CHARACTER_SET_NAME` varchar(32) CHARACTER SET utf8 DEFAULT NULL,
  `COLLATION_NAME` varchar(32) CHARACTER SET utf8 DEFAULT NULL,
  `COLUMN_TYPE` longtext CHARACTER SET utf8 NOT NULL,
  `COLUMN_KEY` varchar(3) CHARACTER SET utf8 NOT NULL DEFAULT '',
  `EXTRA` varchar(30) CHARACTER SET utf8 NOT NULL DEFAULT '',
  `PRIVILEGES` varchar(80) CHARACTER SET utf8 NOT NULL DEFAULT '',
  `COLUMN_COMMENT` varchar(1024) CHARACTER SET utf8 NOT NULL DEFAULT ''
) ENGINE=InnoDB DEFAULT CHARSET=latin1
1 row in set (0.00 sec)

Query OK, 1678 rows affected (0.09 sec)
Records: 1678  Duplicates: 0  Warnings: 0

Query OK, 3356 rows affected (0.07 sec)
Records: 3356  Duplicates: 0  Warnings: 0

Query OK, 6712 rows affected (0.17 sec)
Records: 6712  Duplicates: 0  Warnings: 0

Query OK, 13424 rows affected (0.44 sec)
Records: 13424  Duplicates: 0  Warnings: 0

Query OK, 26848 rows affected (0.63 sec)
Records: 26848  Duplicates: 0  Warnings: 0

Query OK, 53696 rows affected (1.72 sec)
Records: 53696  Duplicates: 0  Warnings: 0

Query OK, 107392 rows affected (3.02 sec)
Records: 107392  Duplicates: 0  Warnings: 0

Query OK, 214784 rows affected (6.28 sec)
Records: 214784  Duplicates: 0  Warnings: 0

Query OK, 429568 rows affected (13.25 sec)
Records: 429568  Duplicates: 0  Warnings: 0

Query OK, 859136 rows affected (28.16 sec)
Records: 859136  Duplicates: 0  Warnings: 0

Query OK, 0 rows affected (0.03 sec)

Query OK, 1718272 rows affected (1 min 9.22 sec)
Records: 1718272  Duplicates: 0  Warnings: 0

+-----------+
| count(id) |
+-----------+
|   1718272 |
+-----------+
1 row in set (1.75 sec)
```

Example 14.2 Speed and Efficiency of CREATE INDEX and DROP INDEX

Here is a sequence of statements demonstrating the relative speed of CREATE INDEX and DROP INDEX statements. For a small table, the elapsed time is less than a second whether we use the fast or slow technique, so we look at the “rows affected” output to verify which operations can avoid the table rebuild. For a large table, the difference in efficiency is obvious because skipping the table rebuild saves substantial time.

```
\! clear

\! echo "=== Create and drop index (small table, new/fast technique) ==="
\! echo
\! echo "Data size (kilobytes) before index created: "
\! du -k data/test/small_table.ibd
create index i_dtyp_small on small_table (data_type), algorithm=inplace;
\! echo "Data size after index created: "
\! du -k data/test/small_table.ibd
drop index i_dtyp_small on small_table, algorithm=inplace;

-- Compare against the older slower DDL.

\! echo "=== Create and drop index (small table, old/slow technique) ==="
\! echo
\! echo "Data size (kilobytes) before index created: "
\! du -k data/test/small_table.ibd
create index i_dtyp_small on small_table (data_type), algorithm=copy;
\! echo "Data size after index created: "
\! du -k data/test/small_table.ibd
drop index i_dtyp_small on small_table, algorithm=copy;

-- In the above example, we examined the "rows affected" number,
-- ideally looking for a zero figure. Let's try again with a larger
-- sample size, where we'll see that the actual time taken can
-- vary significantly.

\! echo "=== Create and drop index (big table, new/fast technique) ==="
\! echo
\! echo "Data size (kilobytes) before index created: "
\! du -k data/test/big_table.ibd
create index i_dtyp_big on big_table (data_type), algorithm=inplace;
\! echo "Data size after index created: "
\! du -k data/test/big_table.ibd
drop index i_dtyp_big on big_table, algorithm=inplace;

\! echo "=== Create and drop index (big table, old/slow technique) ==="
\! echo
\! echo "Data size (kilobytes) before index created: "
\! du -k data/test/big_table.ibd
create index i_dtyp_big on big_table (data_type), algorithm=copy;
\! echo "Data size after index created: "
\! du -k data/test/big_table.ibd
drop index i_dtyp_big on big_table, algorithm=copy;
```

Running this code gives this output, condensed for brevity and with the most important points bolded:

```
Query OK, 0 rows affected (0.00 sec)

=== Create and drop index (small table, new/fast technique) ===

Data size (kilobytes) before index created:
384  data/test/small_table.ibd
Query OK, 0 rows affected (0.04 sec)
Records: 0  Duplicates: 0  Warnings: 0

Data size after index created:
432  data/test/small_table.ibd
Query OK, 0 rows affected (0.02 sec)
Records: 0  Duplicates: 0  Warnings: 0

Query OK, 0 rows affected (0.00 sec)

=== Create and drop index (small table, old/slow technique) ===
Data size (kilobytes) before index created:
432  data/test/small_table.ibd
Query OK, 1678 rows affected (0.12 sec)
Records: 1678  Duplicates: 0  Warnings: 0

Data size after index created:
448  data/test/small_table.ibd
Query OK, 1678 rows affected (0.10 sec)
Records: 1678  Duplicates: 0  Warnings: 0

Query OK, 0 rows affected (0.00 sec)

=== Create and drop index (big table, new/fast technique) ===

Data size (kilobytes) before index created:
315392  data/test/big_table.ibd
Query OK, 0 rows affected (33.32 sec)
Records: 0  Duplicates: 0  Warnings: 0

Data size after index created:
335872  data/test/big_table.ibd
Query OK, 0 rows affected (0.02 sec)
Records: 0  Duplicates: 0  Warnings: 0

Query OK, 0 rows affected (0.00 sec)

=== Create and drop index (big table, old/slow technique) ===

Data size (kilobytes) before index created:
335872  data/test/big_table.ibd
Query OK, 1718272 rows affected (1 min 5.01 sec)
Records: 1718272  Duplicates: 0  Warnings: 0

Data size after index created:
348160  data/test/big_table.ibd
Query OK, 1718272 rows affected (46.59 sec)
Records: 1718272  Duplicates: 0  Warnings: 0
```

Example 14.3 Concurrent DML During CREATE INDEX and DROP INDEX

Here are some snippets of code that I ran in separate `mysql` sessions connected to the same database, to illustrate DML statements (insert, update, or delete) running at the same time as `CREATE INDEX` and `DROP INDEX`.

```

/*
CREATE INDEX statement to run against a table while
insert/update/delete statements are modifying the
column being indexed.
*/

-- We'll run this script in one session, while simultaneously creating and dropping
-- an index on test/big_table.table_name in another session.

use test;
create index i_concurrent on big_table(table_name);
```

```

/*
DROP INDEX statement to run against a table while
insert/update/delete statements are modifying the
column being indexed.
*/

-- We'll run this script in one session, while simultaneously creating and dropping
-- an index on test/big_table.table_name in another session.
```

```
use test;
drop index i_concurrent on big_table;
```

```

/*
Some queries and insert/update/delete statements to run against a table
while an index is being created or dropped. Previously, these operations
would have stalled during the index create/drop period and possibly
timed out or deadlocked.
*/

-- We'll run this script in one session, while simultaneously creating and dropping
-- an index on test/big_table.table_name in another session.

-- In our test instance, that column has about 1.7M rows, with 136 different values.
-- Sample values: COLUMNS (20480), ENGINES (6144), EVENTS (24576), FILES (38912), TABLES (21504), VIEWS (10240

set autocommit = 0;
use test;

select distinct character_set_name from big_table where table_name = 'FILES';
delete from big_table where table_name = 'FILES';
select distinct character_set_name from big_table where table_name = 'FILES';

-- I'll issue the final rollback interactively, not via script,
-- the better to control the timing.
-- rollback;
```

Running this code gives this output, condensed for brevity and with the most important points bolded:

```

mysql: source concurrent_ddl_create.sql
Database changed
Query OK, 0 rows affected (1 min 25.15 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql: source concurrent_ddl_drop.sql
Database changed
Query OK, 0 rows affected (24.98 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql: source concurrent_dml.sql
Query OK, 0 rows affected (0.00 sec)

Database changed
+--------------------+
| character_set_name |
+--------------------+
| NULL               |
| utf8               |
+--------------------+
2 rows in set (0.32 sec)

Query OK, 38912 rows affected (1.84 sec)

Empty set (0.01 sec)

mysql: rollback;
Query OK, 0 rows affected (1.05 sec)
```

Example 14.4 Renaming a Column

Here is a demonstration of using `ALTER TABLE` to rename a column. We use the new, fast DDL mechanism to change the name, then the old, slow DDL mechanism (with `old_alter_table=1`) to restore the original column name.

Notes:

- Because the syntax for renaming a column also involves re-specifying the data type, be very careful to specify exactly the same data type to avoid a costly table rebuild. In this case, we checked the output of `show create table` *`table`*`\G` and copied any clauses such as `CHARACTER SET` and `NOT NULL` from the original column definition.
- Again, renaming a column for a small table is fast enough that we need to examine the “rows affected” number to verify that the new DDL mechanism is more efficient than the old one. With a big table, the difference in elapsed time makes the improvement obvious.

```
/*
Run through a sequence of 'rename column' statements.
Because this operation involves only metadata, not table data,
it is fast for big and small tables, with new or old DDL mechanisms.
*/

\! clear

\! echo "Rename column (fast technique, small table):"
alter table small_table change `IS_NULLABLE` `NULLABLE` varchar(3) character set utf8 not null, algorithm=i
\! echo "Rename back to original name (slow technique):"
alter table small_table change `NULLABLE` `IS_NULLABLE` varchar(3) character set utf8 not null, algorithm=c


\! echo "Rename column (fast technique, big table):"
alter table big_table change `IS_NULLABLE` `NULLABLE` varchar(3) character set utf8 not null, algorithm=inp
\! echo "Rename back to original name (slow technique):"
alter table big_table change `NULLABLE` `IS_NULLABLE` varchar(3) character set utf8 not null, algorithm=cop
```

Running this code gives this output, condensed for brevity and with the most important points bolded:

```
Rename column (fast technique, small table):
Query OK, 0 rows affected (0.05 sec)

Query OK, 0 rows affected (0.13 sec)
Records: 0  Duplicates: 0  Warnings: 0

Rename back to original name (slow technique):
Query OK, 0 rows affected (0.00 sec)

Query OK, 1678 rows affected (0.35 sec)
Records: 1678  Duplicates: 0  Warnings: 0

Rename column (fast technique, big table):
Query OK, 0 rows affected (0.00 sec)

Query OK, 0 rows affected (0.11 sec)
Records: 0  Duplicates: 0  Warnings: 0

Rename back to original name (slow technique):
Query OK, 0 rows affected (0.00 sec)

Query OK, 1718272 rows affected (1 min 0.00 sec)
Records: 1718272  Duplicates: 0  Warnings: 0

Query OK, 0 rows affected (0.00 sec)
```

Example 14.5 Dropping Foreign Keys

Here is a demonstration of foreign keys, including improvement to the speed of dropping a foreign key constraint.

```
/*
Demonstrate aspects of foreign keys that are or aren't affected by the DDL improvements.
```

```
- Create a new table with only a few values to serve as the parent table.
- Set up the 'small' and 'big' tables as child tables using a foreign key.
- Verify that the ON DELETE CASCADE clause makes changes ripple from parent to child tables.
- Drop the foreign key constraints, and optionally associated indexes. (This is the operation that is sped up.
*/

\! clear

-- Make sure foreign keys are being enforced, and allow
-- rollback after doing some DELETEs that affect both
-- parent and child tables.
set foreign_key_checks = 1;
set autocommit = 0;

-- Create a parent table, containing values that we know are already present
-- in the child tables.
drop table if exists schema_names;
create table schema_names (id int unsigned not null primary key auto_increment, schema_name varchar(64) charac

show create table schema_names\G
show create table small_table\G
show create table big_table\G

-- Creating the foreign key constraint still involves a table rebuild when foreign_key_checks=1,
-- as illustrated by the "rows affected" figure.
alter table small_table add constraint small_fk foreign key i_table_schema (table_schema) references schema_na
alter table big_table add constraint big_fk foreign key i_table_schema (table_schema) references schema_names(

show create table small_table\G
show create table big_table\G

select schema_name from schema_names order by schema_name;
select count(table_schema) howmany, table_schema from small_table group by table_schema;
select count(table_schema) howmany, table_schema from big_table group by table_schema;

-- big_table is the parent table.
-- schema_names is the parent table.
-- big_table is the child table.
-- (One row in the parent table can have many "children" in the child table.)
-- Changes to the parent table can ripple through to the child table.
-- For example, removing the value 'test' from schema_names.schema_name will
-- result in the removal of 20K or so rows from big_table.

delete from schema_names where schema_name = 'test';

select schema_name from schema_names order by schema_name;
select count(table_schema) howmany, table_schema from small_table group by table_schema;
select count(table_schema) howmany, table_schema from big_table group by table_schema;

-- Because we've turned off autocommit, we can still get back those deleted rows
-- if the DELETE was issued by mistake.
rollback;

select schema_name from schema_names order by schema_name;
select count(table_schema) howmany, table_schema from small_table group by table_schema;
select count(table_schema) howmany, table_schema from big_table group by table_schema;

-- All of the cross-checking between parent and child tables would be
-- deadly slow if there wasn't the requirement for the corresponding
-- columns to be indexed!

-- But we can get rid of the foreign key using a fast operation
-- that doesn't rebuild the table.
-- If we didn't specify a constraint name when setting up the foreign key, we would
-- have to find the auto-generated name such as 'big_table_ibfk_1' in the
-- output from 'show create table'.
-- For the small table, we'll drop the foreign key and the associated index.
-- Having an index on a small table is less critical.

\! echo "DROP FOREIGN KEY and INDEX from small_table:"
alter table small_table drop foreign key small_fk, drop index small_fk;

-- For the big table, we'll drop the foreign key and leave the associated index.
-- If we are still doing queries that reference the indexed column, the index is
-- very important to avoid a full table scan of the big table.
\! echo "DROP FOREIGN KEY from big_table:"
alter table big_table drop foreign key big_fk;


show create table small_table\G
show create table big_table\G
```

Running this code gives this output, condensed for brevity and with the most important points bolded:

```

Query OK, 0 rows affected (0.00 sec)

Query OK, 0 rows affected (0.00 sec)

Query OK, 0 rows affected (0.01 sec)

Query OK, 4 rows affected (0.03 sec)
Records: 4  Duplicates: 0  Warnings: 0

*************************** 1. row ***************************
       Table: schema_names
Create Table: CREATE TABLE `schema_names` (
  `id` int(10) unsigned NOT NULL AUTO_INCREMENT,
  `schema_name` varchar(64) CHARACTER SET utf8 NOT NULL,
  PRIMARY KEY (`id`),
  KEY `i_schema` (`schema_name`)
) ENGINE=InnoDB AUTO_INCREMENT=8 DEFAULT CHARSET=latin1
1 row in set (0.00 sec)

*************************** 1. row ***************************
       Table: small_table
Create Table: CREATE TABLE `small_table` (
  `TABLE_CATALOG` varchar(512) CHARACTER SET utf8 NOT NULL DEFAULT '',
  `TABLE_SCHEMA` varchar(64) CHARACTER SET utf8 NOT NULL DEFAULT '',
  `TABLE_NAME` varchar(64) CHARACTER SET utf8 NOT NULL DEFAULT '',
  `COLUMN_NAME` varchar(64) CHARACTER SET utf8 NOT NULL DEFAULT '',
  `ORDINAL_POSITION` bigint(21) unsigned NOT NULL DEFAULT '0',
  `COLUMN_DEFAULT` longtext CHARACTER SET utf8,
  `IS_NULLABLE` varchar(3) CHARACTER SET utf8 NOT NULL,
  `DATA_TYPE` varchar(64) CHARACTER SET utf8 NOT NULL DEFAULT '',
  `CHARACTER_MAXIMUM_LENGTH` bigint(21) unsigned DEFAULT NULL,
  `CHARACTER_OCTET_LENGTH` bigint(21) unsigned DEFAULT NULL,
  `NUMERIC_PRECISION` bigint(21) unsigned DEFAULT NULL,
  `NUMERIC_SCALE` bigint(21) unsigned DEFAULT NULL,
  `DATETIME_PRECISION` bigint(21) unsigned DEFAULT NULL,
  `CHARACTER_SET_NAME` varchar(32) CHARACTER SET utf8 DEFAULT NULL,
  `COLLATION_NAME` varchar(32) CHARACTER SET utf8 DEFAULT NULL,
  `COLUMN_TYPE` longtext CHARACTER SET utf8 NOT NULL,
  `COLUMN_KEY` varchar(3) CHARACTER SET utf8 NOT NULL DEFAULT '',
  `EXTRA` varchar(30) CHARACTER SET utf8 NOT NULL DEFAULT '',
  `PRIVILEGES` varchar(80) CHARACTER SET utf8 NOT NULL DEFAULT '',
  `COLUMN_COMMENT` varchar(1024) CHARACTER SET utf8 NOT NULL DEFAULT '',
  `id` int(10) unsigned NOT NULL AUTO_INCREMENT,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=1679 DEFAULT CHARSET=latin1
1 row in set (0.00 sec)

*************************** 1. row ***************************
```

```
       Table: big_table
Create Table: CREATE TABLE `big_table` (
  `TABLE_CATALOG` varchar(512) CHARACTER SET utf8 NOT NULL DEFAULT '',
  `TABLE_SCHEMA` varchar(64) CHARACTER SET utf8 NOT NULL DEFAULT '',
  `TABLE_NAME` varchar(64) CHARACTER SET utf8 NOT NULL DEFAULT '',
  `COLUMN_NAME` varchar(64) CHARACTER SET utf8 NOT NULL DEFAULT '',
  `ORDINAL_POSITION` bigint(21) unsigned NOT NULL DEFAULT '0',
  `COLUMN_DEFAULT` longtext CHARACTER SET utf8,
  `IS_NULLABLE` varchar(3) CHARACTER SET utf8 NOT NULL,
  `DATA_TYPE` varchar(64) CHARACTER SET utf8 NOT NULL DEFAULT '',
  `CHARACTER_MAXIMUM_LENGTH` bigint(21) unsigned DEFAULT NULL,
  `CHARACTER_OCTET_LENGTH` bigint(21) unsigned DEFAULT NULL,
  `NUMERIC_PRECISION` bigint(21) unsigned DEFAULT NULL,
  `NUMERIC_SCALE` bigint(21) unsigned DEFAULT NULL,
  `DATETIME_PRECISION` bigint(21) unsigned DEFAULT NULL,
  `CHARACTER_SET_NAME` varchar(32) CHARACTER SET utf8 DEFAULT NULL,
  `COLLATION_NAME` varchar(32) CHARACTER SET utf8 DEFAULT NULL,
  `COLUMN_TYPE` longtext CHARACTER SET utf8 NOT NULL,
  `COLUMN_KEY` varchar(3) CHARACTER SET utf8 NOT NULL DEFAULT '',
  `EXTRA` varchar(30) CHARACTER SET utf8 NOT NULL DEFAULT '',
  `PRIVILEGES` varchar(80) CHARACTER SET utf8 NOT NULL DEFAULT '',
  `COLUMN_COMMENT` varchar(1024) CHARACTER SET utf8 NOT NULL DEFAULT '',
  `id` int(10) unsigned NOT NULL AUTO_INCREMENT,
  PRIMARY KEY (`id`),
  KEY `big_fk` (`TABLE_SCHEMA`)
) ENGINE=InnoDB AUTO_INCREMENT=1718273 DEFAULT CHARSET=latin1
1 row in set (0.00 sec)

Query OK, 1678 rows affected (0.10 sec)
Records: 1678  Duplicates: 0  Warnings: 0

Query OK, 1718272 rows affected (1 min 14.54 sec)
Records: 1718272  Duplicates: 0  Warnings: 0

*************************** 1. row ***************************
       Table: small_table
Create Table: CREATE TABLE `small_table` (
  `TABLE_CATALOG` varchar(512) CHARACTER SET utf8 NOT NULL DEFAULT '',
  `TABLE_SCHEMA` varchar(64) CHARACTER SET utf8 NOT NULL DEFAULT '',
  `TABLE_NAME` varchar(64) CHARACTER SET utf8 NOT NULL DEFAULT '',
  `COLUMN_NAME` varchar(64) CHARACTER SET utf8 NOT NULL DEFAULT '',
  `ORDINAL_POSITION` bigint(21) unsigned NOT NULL DEFAULT '0',
  `COLUMN_DEFAULT` longtext CHARACTER SET utf8,
  `IS_NULLABLE` varchar(3) CHARACTER SET utf8 NOT NULL,
  `DATA_TYPE` varchar(64) CHARACTER SET utf8 NOT NULL DEFAULT '',
  `CHARACTER_MAXIMUM_LENGTH` bigint(21) unsigned DEFAULT NULL,
  `CHARACTER_OCTET_LENGTH` bigint(21) unsigned DEFAULT NULL,
  `NUMERIC_PRECISION` bigint(21) unsigned DEFAULT NULL,
  `NUMERIC_SCALE` bigint(21) unsigned DEFAULT NULL,
  `DATETIME_PRECISION` bigint(21) unsigned DEFAULT NULL,
  `CHARACTER_SET_NAME` varchar(32) CHARACTER SET utf8 DEFAULT NULL,
  `COLLATION_NAME` varchar(32) CHARACTER SET utf8 DEFAULT NULL,
  `COLUMN_TYPE` longtext CHARACTER SET utf8 NOT NULL,
  `COLUMN_KEY` varchar(3) CHARACTER SET utf8 NOT NULL DEFAULT '',
  `EXTRA` varchar(30) CHARACTER SET utf8 NOT NULL DEFAULT '',
  `PRIVILEGES` varchar(80) CHARACTER SET utf8 NOT NULL DEFAULT '',
  `COLUMN_COMMENT` varchar(1024) CHARACTER SET utf8 NOT NULL DEFAULT '',
  `id` int(10) unsigned NOT NULL AUTO_INCREMENT,
  PRIMARY KEY (`id`),
  KEY `small_fk` (`TABLE_SCHEMA`),
  CONSTRAINT `small_fk` FOREIGN KEY (`TABLE_SCHEMA`) REFERENCES `schema_names` (`schema_name`) ON DELETE CASCA
) ENGINE=InnoDB AUTO_INCREMENT=1679 DEFAULT CHARSET=latin1
1 row in set (0.12 sec)

*************************** 1. row ***************************
       Table: big_table
Create Table: CREATE TABLE `big_table` (
  `TABLE_CATALOG` varchar(512) CHARACTER SET utf8 NOT NULL DEFAULT '',
  `TABLE_SCHEMA` varchar(64) CHARACTER SET utf8 NOT NULL DEFAULT '',
  `TABLE_NAME` varchar(64) CHARACTER SET utf8 NOT NULL DEFAULT '',
  `COLUMN_NAME` varchar(64) CHARACTER SET utf8 NOT NULL DEFAULT '',
  `ORDINAL_POSITION` bigint(21) unsigned NOT NULL DEFAULT '0',
  `COLUMN_DEFAULT` longtext CHARACTER SET utf8,
  `IS_NULLABLE` varchar(3) CHARACTER SET utf8 NOT NULL,
  `DATA_TYPE` varchar(64) CHARACTER SET utf8 NOT NULL DEFAULT '',
  `CHARACTER_MAXIMUM_LENGTH` bigint(21) unsigned DEFAULT NULL,
  `CHARACTER_OCTET_LENGTH` bigint(21) unsigned DEFAULT NULL,
  `NUMERIC_PRECISION` bigint(21) unsigned DEFAULT NULL,
  `NUMERIC_SCALE` bigint(21) unsigned DEFAULT NULL,
  `DATETIME_PRECISION` bigint(21) unsigned DEFAULT NULL,
  `CHARACTER_SET_NAME` varchar(32) CHARACTER SET utf8 DEFAULT NULL,
  `COLLATION_NAME` varchar(32) CHARACTER SET utf8 DEFAULT NULL,
  `COLUMN_TYPE` longtext CHARACTER SET utf8 NOT NULL,
  `COLUMN_KEY` varchar(3) CHARACTER SET utf8 NOT NULL DEFAULT '',
  `EXTRA` varchar(30) CHARACTER SET utf8 NOT NULL DEFAULT '',
  `PRIVILEGES` varchar(80) CHARACTER SET utf8 NOT NULL DEFAULT '',
  `COLUMN_COMMENT` varchar(1024) CHARACTER SET utf8 NOT NULL DEFAULT '',
  `id` int(10) unsigned NOT NULL AUTO_INCREMENT,
  PRIMARY KEY (`id`),
  KEY `big_fk` (`TABLE_SCHEMA`),
  CONSTRAINT `big_fk` FOREIGN KEY (`TABLE_SCHEMA`) REFERENCES `schema_names` (`schema_name`) ON DELETE CASC
) ENGINE=InnoDB AUTO_INCREMENT=1718273 DEFAULT CHARSET=latin1
1 row in set (0.01 sec)

+--------------------+
| schema_name        |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| test               |
+--------------------+
4 rows in set (0.00 sec)

+---------+--------------------+
| howmany | table_schema       |
+---------+--------------------+
|     563 | information_schema |
|     286 | mysql              |
|     786 | performance_schema |
|      43 | test               |
+---------+--------------------+
4 rows in set (0.01 sec)

+---------+--------------------+
| howmany | table_schema       |
+---------+--------------------+
|  576512 | information_schema |
|  292864 | mysql              |
|  804864 | performance_schema |
|   44032 | test               |
+---------+--------------------+
4 rows in set (2.10 sec)

Query OK, 1 row affected (1.52 sec)

+--------------------+
| schema_name        |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
+--------------------+
3 rows in set (0.00 sec)
```

```
+---------+--------------------+
| howmany | table_schema       |
+---------+--------------------+
|     563 | information_schema |
|     286 | mysql              |
|     786 | performance_schema |
+---------+--------------------+
3 rows in set (0.00 sec)

+---------+--------------------+
| howmany | table_schema       |
+---------+--------------------+
|  576512 | information_schema |
|  292864 | mysql              |
|  804864 | performance_schema |
+---------+--------------------+
3 rows in set (1.74 sec)

Query OK, 0 rows affected (0.60 sec)

+--------------------+
| schema_name        |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| test               |
+--------------------+
4 rows in set (0.00 sec)

+---------+--------------------+
| howmany | table_schema       |
+---------+--------------------+
|     563 | information_schema |
|     286 | mysql              |
|     786 | performance_schema |
|      43 | test               |
+---------+--------------------+
4 rows in set (0.01 sec)

+---------+--------------------+
| howmany | table_schema       |
+---------+--------------------+
|  576512 | information_schema |
|  292864 | mysql              |
|  804864 | performance_schema |
|   44032 | test               |
+---------+--------------------+
4 rows in set (1.59 sec)

DROP FOREIGN KEY and INDEX from small_table:
Query OK, 0 rows affected (0.02 sec)
Records: 0  Duplicates: 0  Warnings: 0

DROP FOREIGN KEY from big_table:
Query OK, 0 rows affected (0.02 sec)
Records: 0  Duplicates: 0  Warnings: 0

*************************** 1. row ***************************
       Table: small_table
Create Table: CREATE TABLE `small_table` (
  `TABLE_CATALOG` varchar(512) CHARACTER SET utf8 NOT NULL DEFAULT '',
  `TABLE_SCHEMA` varchar(64) CHARACTER SET utf8 NOT NULL DEFAULT '',
  `TABLE_NAME` varchar(64) CHARACTER SET utf8 NOT NULL DEFAULT '',
  `COLUMN_NAME` varchar(64) CHARACTER SET utf8 NOT NULL DEFAULT '',
  `ORDINAL_POSITION` bigint(21) unsigned NOT NULL DEFAULT '0',
  `COLUMN_DEFAULT` longtext CHARACTER SET utf8,
  `IS_NULLABLE` varchar(3) CHARACTER SET utf8 NOT NULL,
  `DATA_TYPE` varchar(64) CHARACTER SET utf8 NOT NULL DEFAULT '',
  `CHARACTER_MAXIMUM_LENGTH` bigint(21) unsigned DEFAULT NULL,
  `CHARACTER_OCTET_LENGTH` bigint(21) unsigned DEFAULT NULL,
  `NUMERIC_PRECISION` bigint(21) unsigned DEFAULT NULL,
  `NUMERIC_SCALE` bigint(21) unsigned DEFAULT NULL,
  `DATETIME_PRECISION` bigint(21) unsigned DEFAULT NULL,
  `CHARACTER_SET_NAME` varchar(32) CHARACTER SET utf8 DEFAULT NULL,
  `COLLATION_NAME` varchar(32) CHARACTER SET utf8 DEFAULT NULL,
  `COLUMN_TYPE` longtext CHARACTER SET utf8 NOT NULL,
  `COLUMN_KEY` varchar(3) CHARACTER SET utf8 NOT NULL DEFAULT '',
  `EXTRA` varchar(30) CHARACTER SET utf8 NOT NULL DEFAULT '',
  `PRIVILEGES` varchar(80) CHARACTER SET utf8 NOT NULL DEFAULT '',
  `COLUMN_COMMENT` varchar(1024) CHARACTER SET utf8 NOT NULL DEFAULT '',
  `id` int(10) unsigned NOT NULL AUTO_INCREMENT,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=1679 DEFAULT CHARSET=latin1
1 row in set (0.00 sec)

*************************** 1. row ***************************
       Table: big_table
Create Table: CREATE TABLE `big_table` (
  `TABLE_CATALOG` varchar(512) CHARACTER SET utf8 NOT NULL DEFAULT '',
  `TABLE_SCHEMA` varchar(64) CHARACTER SET utf8 NOT NULL DEFAULT '',
  `TABLE_NAME` varchar(64) CHARACTER SET utf8 NOT NULL DEFAULT '',
  `COLUMN_NAME` varchar(64) CHARACTER SET utf8 NOT NULL DEFAULT '',
  `ORDINAL_POSITION` bigint(21) unsigned NOT NULL DEFAULT '0',
  `COLUMN_DEFAULT` longtext CHARACTER SET utf8,
  `IS_NULLABLE` varchar(3) CHARACTER SET utf8 NOT NULL,
  `DATA_TYPE` varchar(64) CHARACTER SET utf8 NOT NULL DEFAULT '',
  `CHARACTER_MAXIMUM_LENGTH` bigint(21) unsigned DEFAULT NULL,
  `CHARACTER_OCTET_LENGTH` bigint(21) unsigned DEFAULT NULL,
  `NUMERIC_PRECISION` bigint(21) unsigned DEFAULT NULL,
  `NUMERIC_SCALE` bigint(21) unsigned DEFAULT NULL,
  `DATETIME_PRECISION` bigint(21) unsigned DEFAULT NULL,
  `CHARACTER_SET_NAME` varchar(32) CHARACTER SET utf8 DEFAULT NULL,
  `COLLATION_NAME` varchar(32) CHARACTER SET utf8 DEFAULT NULL,
  `COLUMN_TYPE` longtext CHARACTER SET utf8 NOT NULL,
  `COLUMN_KEY` varchar(3) CHARACTER SET utf8 NOT NULL DEFAULT '',
  `EXTRA` varchar(30) CHARACTER SET utf8 NOT NULL DEFAULT '',
  `PRIVILEGES` varchar(80) CHARACTER SET utf8 NOT NULL DEFAULT '',
  `COLUMN_COMMENT` varchar(1024) CHARACTER SET utf8 NOT NULL DEFAULT '',
  `id` int(10) unsigned NOT NULL AUTO_INCREMENT,
  PRIMARY KEY (`id`),
  KEY `big_fk` (`TABLE_SCHEMA`)
) ENGINE=InnoDB AUTO_INCREMENT=1718273 DEFAULT CHARSET=latin1
1 row in set (0.00 sec)
```

Example 14.6 Changing Auto-Increment Value

Here is an illustration of increasing the auto-increment lower limit for a table column, demonstrating how this operation now avoids a table rebuild, plus some other fun facts about InnoDB auto-increment columns.

```
/*
If this script is run after foreign_key.sql, the schema_names table is
already set up. But to allow this script to run multiple times without
running into duplicate ID errors, we set up the schema_names table
all over again.
*/

\! clear

\! echo "=== Adjusting the Auto-Increment Limit for a Table ==="
\! echo

drop table if exists schema_names;
create table schema_names (id int unsigned not null primary key auto_increment,
  schema_name varchar(64) character set utf8 not null, index i_schema (schema_name))
  as select distinct table_schema schema_name from small_table;

\! echo "Initial state of schema_names table. AUTO_INCREMENT is included in SHOW CREATE TABLE output."
\! echo "Note how MySQL reserved a block of IDs, but only needed 4 of them in this transaction, so the next in
show create table schema_names\G
select * from schema_names order by id;

\! echo "Inserting even a tiny amount of data can produce gaps in the ID sequence."
insert into schema_names (schema_name) values ('eight'), ('nine');

\! echo "Bumping auto-increment lower limit to 20 (fast mechanism):"
alter table schema_names auto_increment=20, algorithm=inplace;

\! echo "Inserting 2 rows that should get IDs 20 and 21:"
insert into schema_names (schema_name) values ('foo'), ('bar');
commit;

\! echo "Bumping auto-increment lower limit to 30 (slow mechanism):"
alter table schema_names auto_increment=30, algorithm=copy;

\! echo "Inserting 2 rows that should get IDs 30 and 31:"
insert into schema_names (schema_name) values ('bletch'),('baz');
commit;

select * from schema_names order by id;

\! echo "Final state of schema_names table. AUTO_INCREMENT value shows the next inserted row would get ID=32."
show create table schema_names\G
```

Running this code gives this output, condensed for brevity and with the most important points bolded:

```

=== Adjusting the Auto-Increment Limit for a Table ===

Query OK, 0 rows affected (0.01 sec)

Query OK, 4 rows affected (0.02 sec)
Records: 4  Duplicates: 0  Warnings: 0

Initial state of schema_names table. AUTO_INCREMENT is included in SHOW CREATE TABLE output.
Note how MySQL reserved a block of IDs, but only needed 4 of them in this transaction, so the next inserted va
*************************** 1. row ***************************
       Table: schema_names
Create Table: CREATE TABLE `schema_names` (
  `id` int(10) unsigned NOT NULL AUTO_INCREMENT,
  `schema_name` varchar(64) CHARACTER SET utf8 NOT NULL,
  PRIMARY KEY (`id`),
  KEY `i_schema` (`schema_name`)
) ENGINE=InnoDB AUTO_INCREMENT=8 DEFAULT CHARSET=latin1
1 row in set (0.00 sec)

+----+--------------------+
| id | schema_name        |
+----+--------------------+
|  1 | information_schema |
|  2 | mysql              |
|  3 | performance_schema |
|  4 | test               |
+----+--------------------+
4 rows in set (0.00 sec)

Inserting even a tiny amount of data can produce gaps in the ID sequence.
```

```
Query OK, 2 rows affected (0.00 sec)
Records: 2  Duplicates: 0  Warnings: 0

Query OK, 0 rows affected (0.00 sec)

Bumping auto-increment lower limit to 20 (fast mechanism):
Query OK, 0 rows affected (0.01 sec)
Records: 0  Duplicates: 0  Warnings: 0

Inserting 2 rows that should get IDs 20 and 21:
Query OK, 2 rows affected (0.00 sec)
Records: 2  Duplicates: 0  Warnings: 0

Query OK, 0 rows affected (0.00 sec)

Query OK, 0 rows affected (0.00 sec)

Bumping auto-increment lower limit to 30 (slow mechanism):
Query OK, 8 rows affected (0.02 sec)
Records: 8  Duplicates: 0  Warnings: 0

Inserting 2 rows that should get IDs 30 and 31:
Query OK, 2 rows affected (0.00 sec)
Records: 2  Duplicates: 0  Warnings: 0

Query OK, 0 rows affected (0.01 sec)

+----+--------------------+
| id | schema_name        |
+----+--------------------+
|  1 | information_schema |
|  2 | mysql              |
|  3 | performance_schema |
|  4 | test               |
|  8 | eight              |
|  9 | nine               |
| 20 | foo                |
| 21 | bar                |
| 30 | bletch             |
| 31 | baz                |
+----+--------------------+
10 rows in set (0.00 sec)

Query OK, 0 rows affected (0.00 sec)

Final state of schema_names table. AUTO_INCREMENT value shows the next inserted row would get ID=32.
*************************** 1. row ***************************
       Table: schema_names
Create Table: CREATE TABLE `schema_names` (
  `id` int(10) unsigned NOT NULL AUTO_INCREMENT,
  `schema_name` varchar(64) CHARACTER SET utf8 NOT NULL,
  PRIMARY KEY (`id`),
  KEY `i_schema` (`schema_name`)
) ENGINE=InnoDB AUTO_INCREMENT=32 DEFAULT CHARSET=latin1
1 row in set (0.00 sec)
```

Example 14.7 Controlling Concurrency with the LOCK Clause

This example shows how to use the LOCK clause of the ALTER TABLE statement to allow or deny concurrent access to the table while an online DDL operation is in progress. The clause has settings that allow queries and DML statements (LOCK=NONE), just queries (LOCK=SHARED), or no concurrent access at all (LOCK=EXCLUSIVE).

In one session, we run a succession of ALTER TABLE statements to create and drop an index, using different values for the LOCK clause to see what happens with waiting or deadlocking in either session.

We are using the same BIG_TABLE table as in previous examples, starting with approximately 1.7 million rows. For illustration purposes, we will index and query the IS_NULLABLE column. (Although in real life it would be silly to make an index for a tiny column with only 2 distinct values.)

```
mysql: desc big_table;
+--------------------------+---------------------+------+-----+---------+----------------+
| Field                    | Type                | Null | Key | Default | Extra          |
+--------------------------+---------------------+------+-----+---------+----------------+
| TABLE_CATALOG            | varchar(512)        | NO   |     |         |                |
| TABLE_SCHEMA             | varchar(64)         | NO   |     |         |                |
| TABLE_NAME               | varchar(64)         | NO   |     |         |                |
| COLUMN_NAME              | varchar(64)         | NO   |     |         |                |
| ORDINAL_POSITION         | bigint(21) unsigned | NO   |     | 0       |                |
| COLUMN_DEFAULT           | longtext            | YES  |     | NULL    |                |

| IS_NULLABLE              | varchar(3)          | NO   |     |         |                |
...
+--------------------------+---------------------+------+-----+---------+----------------+
21 rows in set (0.14 sec)

mysql: alter table big_table add index i1(is_nullable);
Query OK, 0 rows affected (20.71 sec)

mysql: alter table big_table drop index i1;
Query OK, 0 rows affected (0.02 sec)

mysql: alter table big_table add index i1(is_nullable), lock=exclusive;
Query OK, 0 rows affected (19.44 sec)

mysql: alter table big_table drop index i1;
Query OK, 0 rows affected (0.03 sec)

mysql: alter table big_table add index i1(is_nullable), lock=shared;
Query OK, 0 rows affected (16.71 sec)

mysql: alter table big_table drop index i1;
Query OK, 0 rows affected (0.05 sec)

mysql: alter table big_table add index i1(is_nullable), lock=none;
Query OK, 0 rows affected (12.26 sec)

mysql: alter table big_table drop index i1;
Query OK, 0 rows affected (0.01 sec)

... repeat statements like the above while running queries ...
... and DML statements at the same time in another session ...
```

Nothing dramatic happens in the session running the DDL statements. Sometimes, an ALTER TABLE takes unusually long because it is waiting for another transaction to finish, when that transaction modified the table during the DDL or queried the table before the DDL:

```
mysql: alter table big_table add index i1(is_nullable), lock=none;

Query OK, 0 rows affected (59.27 sec)

mysql: -- The previous ALTER took so long because it was waiting for all the concurrent
mysql: -- transactions to commit or roll back.

mysql: alter table big_table drop index i1;
Query OK, 0 rows affected (41.05 sec)

mysql: -- Even doing a SELECT on the table in the other session first causes
mysql: -- the ALTER TABLE above to stall until the transaction
mysql: -- surrounding the SELECT is committed or rolled back.
```

Here is the log from another session running concurrently, where we issue queries and DML statements against the table before, during, and after the DDL operations shown in the previous listings. This first listing shows queries only. We expect the queries to be allowed during DDL operations using LOCK=NONE or LOCK=SHARED, and for the query to wait until the DDL is finished if the ALTER TABLE statement includes LOCK=EXCLUSIVE.

```
mysql: show variables like 'autocommit';
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| autocommit    | ON    |
+---------------+-------+
1 row in set (0.01 sec)

mysql: -- A trial query before any ADD INDEX in the other session:
mysql: -- Note: because autocommit is enabled, each
mysql: -- transaction finishes immediately after the query.
mysql: select distinct is_nullable from big_table;
+-------------+
| is_nullable |
+-------------+
| NO          |
| YES         |
+-------------+
2 rows in set (4.49 sec)

mysql: -- Index is being created with LOCK=EXCLUSIVE on the ALTER statement.
mysql: -- The query waits until the DDL is finished before proceeding.
mysql: select distinct is_nullable from big_table;
+-------------+
| is_nullable |
+-------------+
| NO          |
| YES         |
+-------------+

2 rows in set (17.26 sec)

mysql: -- Index is being created with LOCK=SHARED on the ALTER statement.
mysql: -- The query returns its results while the DDL is in progress.
mysql: -- The same thing happens with LOCK=NONE on the ALTER statement.
mysql: select distinct is_nullable from big_table;
+-------------+
| is_nullable |
+-------------+
| NO          |
| YES         |
+-------------+
2 rows in set (3.11 sec)

mysql: -- Once the index is created, and with no DDL in progress,
mysql: -- queries referencing the indexed column are very fast:
mysql: select count(*) from big_table where is_nullable = 'YES';
+----------+
| count(*) |
+----------+
|   411648 |
+----------+
1 row in set (0.20 sec)

mysql: select distinct is_nullable from big_table;
+-------------+
| is_nullable |
+-------------+
| NO          |
| YES         |
+-------------+
2 rows in set (0.00 sec)
```

Now in this concurrent session, we run some transactions including DML statements, or a combination of DML statements and queries. We use DELETE statements to illustrate predictable, verifiable changes to the table. Because the transactions in this part can span multiple statements, we run these tests with autocommit turned off.

```

mysql: set global autocommit = off;
Query OK, 0 rows affected (0.00 sec)

mysql: -- Count the rows that will be involved in our DELETE statements:
mysql: select count(*) from big_table where is_nullable = 'YES';
+----------+
| count(*) |
+----------+
|   411648 |
+----------+
1 row in set (0.95 sec)

mysql: -- After this point, any DDL statements back in the other session
mysql: -- stall until we commit or roll back.

mysql: delete from big_table where is_nullable = 'YES' limit 11648;
Query OK, 11648 rows affected (0.14 sec)

mysql: select count(*) from big_table where is_nullable = 'YES';
+----------+
| count(*) |
+----------+
|   400000 |
+----------+
1 row in set (1.04 sec)

mysql: rollback;
Query OK, 0 rows affected (0.09 sec)

mysql: select count(*) from big_table where is_nullable = 'YES';
+----------+
| count(*) |
+----------+
|   411648 |
+----------+
1 row in set (0.93 sec)

mysql: -- OK, now we're going to try that during index creation with LOCK=NONE.
mysql: delete from big_table where is_nullable = 'YES' limit 11648;
Query OK, 11648 rows affected (0.21 sec)

mysql: -- We expect that now there will be 400000 'YES' rows left:
mysql: select count(*) from big_table where is_nullable = 'YES';
+----------+
| count(*) |
+----------+
|   400000 |
+----------+
1 row in set (1.25 sec)

mysql: -- In the other session, the ALTER TABLE is waiting before finishing,
mysql: -- because _this_ transaction hasn't committed or rolled back yet.
mysql: rollback;
Query OK, 0 rows affected (0.11 sec)

mysql: select count(*) from big_table where is_nullable = 'YES';
```

```
+----------+
| count(*) |
+----------+
|   411648 |
+----------+
1 row in set (0.19 sec)

mysql: -- The ROLLBACK left the table in the same state we originally found it.
mysql: -- Now let's make a permanent change while the index is being created,
mysql: -- again with ALTER TABLE ... , LOCK=NONE.
mysql: -- First, commit so the DROP INDEX in the other shell can finish;
mysql: -- the previous SELECT started a transaction that accessed the table.
mysql: commit;
Query OK, 0 rows affected (0.00 sec)

mysql: -- Now we add the index back in the other shell, then issue DML in this one
mysql: -- while the DDL is running.
mysql: delete from big_table where is_nullable = 'YES' limit 11648;
Query OK, 11648 rows affected (0.23 sec)

mysql: commit;
Query OK, 0 rows affected (0.01 sec)

mysql: -- In the other shell, the ADD INDEX has finished.
mysql: select count(*) from big_table where is_nullable = 'YES';
+----------+
| count(*) |
+----------+
|   400000 |
+----------+
1 row in set (0.19 sec)

mysql: -- At the point the new index is finished being created, it contains entries
mysql: -- only for the 400000 'YES' rows left when all concurrent transactions are finished.
mysql:
mysql: -- Now we will run a similar test, while ALTER TABLE ... , LOCK=SHARED is running.
mysql: -- We expect a query to complete during the ALTER TABLE, but for the DELETE
mysql: -- to run into some kind of issue.
mysql: commit;
Query OK, 0 rows affected (0.00 sec)

mysql: -- As expected, the query returns results while the LOCK=SHARED DDL is running:
mysql: select count(*) from big_table where is_nullable = 'YES';
+----------+
| count(*) |
+----------+
|   400000 |
+----------+
1 row in set (2.07 sec)

mysql: -- The DDL in the other session is not going to finish until this transaction
mysql: -- is committed or rolled back. If we tried a DELETE now and it waited because
mysql: -- of LOCK=SHARED on the DDL, both transactions would wait forever (deadlock).
mysql: -- MySQL detects this condition and cancels the attempted DML statement.
mysql: delete from big_table where is_nullable = 'YES' limit 100000;
ERROR 1213 (40001): Deadlock found when trying to get lock; try restarting transaction
mysql: -- The transaction here is still going, so in the other shell, the ADD INDEX operation
mysql: -- is waiting for this transaction to commit or roll back.
mysql: rollback;
Query OK, 0 rows affected (0.00 sec)

mysql: -- Now let's try issuing a query and some DML, on one line, while running
mysql: -- ALTER TABLE ... , LOCK=EXCLUSIVE in the other shell.
mysql: -- Notice how even the query is held up until the DDL is finished.
mysql: -- By the time the DELETE is issued, there is no conflicting access
mysql: -- to the table and we avoid the deadlock error.
mysql: select count(*) from big_table where is_nullable = 'YES'; delete from big_table where is_nullable =
+----------+
| count(*) |
+----------+
|   400000 |
+----------+

1 row in set (15.98 sec)

Query OK, 100000 rows affected (2.81 sec)

mysql: select count(*) from big_table where is_nullable = 'YES';
+----------+
| count(*) |
+----------+
|   300000 |
+----------+
1 row in set (0.17 sec)

mysql: rollback;
Query OK, 0 rows affected (1.36 sec)

mysql: select count(*) from big_table where is_nullable = 'YES';
+----------+
| count(*) |
+----------+
|   400000 |
+----------+
1 row in set (0.19 sec)

mysql: commit;
Query OK, 0 rows affected (0.00 sec)

mysql: -- Next, we try ALTER TABLE ... , LOCK=EXCLUSIVE in the other session
mysql: -- and only issue DML, not any query, in the concurrent transaction here.
mysql: delete from big_table where is_nullable = 'YES' limit 100000;
Query OK, 100000 rows affected (16.37 sec)

mysql: -- That was OK because the ALTER TABLE did not have to wait for the transaction
mysql: -- here to complete. The DELETE in this session waited until the index was ready.
mysql: select count(*) from big_table where is_nullable = 'YES';
+----------+
| count(*) |
+----------+
|   300000 |
+----------+
1 row in set (0.16 sec)

mysql: commit;
Query OK, 0 rows affected (0.00 sec)
```

In the preceding example listings, we learned that:

- The LOCK clause for ALTER TABLE is set off from the rest of the statement by a comma.
- Online DDL operations might wait before starting, until any prior transactions that access the table are committed or rolled back.
- Online DDL operations might wait before completing, until any concurrent transactions that access the table are committed or rolled back.
- While an online DDL operation is running, concurrent queries are relatively straightforward, as long as the ALTER TABLE statement uses LOCK=NONE or LOCK=SHARED.
- Pay attention to whether autocommit is turned on or off. If it is turned off, be careful to end transactions in other sessions (even just queries) before performing DDL operations on the table.

- With LOCK=SHARED, concurrent transactions that mix queries and DML could encounter deadlock errors and have to be restarted after the DDL is finished.
- With LOCK=NONE, concurrent transactions can freely mix queries and DML. The DDL operation waits until the concurrent transactions are committed or rolled back.
- With LOCK=NONE, concurrent transactions can freely mix queries and DML, but those transactions wait until the DDL operation is finished before they can access the table.

Example 14.8 Schema Setup Code for Online DDL Experiments

You can create multiple indexes on a table with one ALTER TABLE statement. This is relatively efficient, because the clustered index of the table needs to be scanned only once (although the data is sorted separately for each new index). For example:

```
CREATE TABLE T1(A INT PRIMARY KEY, B INT, C CHAR(1)) ENGINE=InnoDB;
INSERT INTO T1 VALUES (1,2,'a'), (2,3,'b'), (3,2,'c'), (4,3,'d'), (5,2,'e');
COMMIT;
ALTER TABLE T1 ADD INDEX (B), ADD UNIQUE INDEX (C);
```

The above statements create table T1 with the primary key on column A, insert several rows, then build two new indexes on columns B and C. If there were many rows inserted into T1 before the ALTER TABLE statement, this approach is much more efficient than creating all the secondary indexes before loading the data.

Because dropping InnoDB secondary indexes also does not require any copying of table data, it is equally efficient to drop multiple indexes with a single ALTER TABLE statement or multiple DROP INDEX statements:

```
ALTER TABLE T1 DROP INDEX B, DROP INDEX C;
```

or:

```
DROP INDEX B ON T1;
DROP INDEX C ON T1;
```

Example 14.9 Creating and Dropping the Primary Key

Restructuring the clustered index for an InnoDB table always requires copying the table data. Thus, it is best to define the primary key when you create a table, rather than issuing ALTER TABLE ... ADD PRIMARY KEY later, to avoid rebuilding the table.

Defining a PRIMARY KEY later causes the data to be copied, as in the following example:

```
CREATE TABLE T2 (A INT, B INT);
INSERT INTO T2 VALUES (NULL, 1);
ALTER TABLE T2 ADD PRIMARY KEY (B);
```

When you create a UNIQUE or PRIMARY KEY index, MySQL must do some extra work. For UNIQUE indexes, MySQL checks that the table contains no duplicate values for the key. For a PRIMARY KEY index, MySQL also checks that none of the PRIMARY KEY columns contains a NULL.

When you add a primary key using the ALGORITHM=COPY clause, MySQL actually converts NULL values in the associated columns to default values: 0 for numbers, the empty string for character-based columns and BLOBs, and January 1, 1975 for dates. This is a non-standard behavior that Oracle recommends you not rely on. Adding a primary key using ALGORITHM=INPLACE is only allowed when the SQL_MODE setting includes the strict_trans_tables or strict_all_tables flags; when the SQL_MODEsetting is strict, ADD PRIMARY KEY ... , ALGORITHM=INPLACE is allowed, but the statement can still fail if the requested primary key columns contain any NULL values. The ALGORITHM=INPLACE behavior is more standard-compliant.

The following example shows the different possibilities for the ADD PRIMARY KEY clause. With the ALGORITHM=COPY clause, the operation succeeds despite the presence of NULL values in the primary key columns; the data is silently changed, which could cause problems. With the ALGORITHM=INPLACE clause, the operation could fail for different reasons, because this setting considers data integrity a high priority: the statement gives an error if the SQL_MODE setting is not “strict” enough, or if the primary key columns contain any NULL values. Once we address both of those requirements, the ALTER TABLE operation succeeds.

```
CREATE TABLE add_pk_via_copy (c1 INT, c2 VARCHAR(10), c3 DATETIME);
INSERT INTO add_pk_via_copy VALUES (1,'a','...'),(NULL,NULL,NULL);
ALTER TABLE add_pk_via_copy ADD PRIMARY KEY (c1,c2,c3), ALGORITHM=COPY;
SELECT * FROM add_pk_via_copy;

CREATE TABLE add_pk_via_inplace (c1 INT, c2 VARCHAR(10), c3 DATETIME);
INSERT INTO add_pk_via_inplace VALUES (1,'a','...'),(NULL,NULL,NULL);
SET sql_mode = 'strict_trans_tables';
ALTER TABLE add_pk_via_inplace ADD PRIMARY KEY (c1,c2,c3), ALGORITHM=COPY;
SET sql_mode = '';
ALTER TABLE add_pk_via_inplace ADD PRIMARY KEY (c1,c2,c3), ALGORITHM=COPY;
DELETE FROM add_pk_via_inplace WHERE c1 IS NULL OR c2 IS NULL OR c3 IS NULL;
ALTER TABLE add_pk_via_inplace ADD PRIMARY KEY (c1,c2,c3), ALGORITHM=COPY;
SELECT * FROM add_pk_via_inplace;
```

If you create a table without a primary key, InnoDB chooses one for you, which can be the first UNIQUE key defined on NOT NULL columns, or a system-generated key. To avoid any uncertainty and the potential space requirement for an extra hidden column, specify the PRIMARY KEY clause as part of the CREATE TABLE statement.

14.2.11.6 Implementation Details of Online DDL

Each ALTER TABLE operation for an InnoDB table is governed by several aspects:

- Whether there is any change to the physical representation of the table, or whether it purely a change to metadata that can be done without touching the table itself.

- Whether the volume of data in the table stays the same, increases, or decreases.

- Whether a change in table data involves the clustered index, secondary indexes, or both.

- Whether there are any foreign key relationships between the table being altered and some other table. The mechanics differ depending on whether the foreign_key_checks configuration option is enabled or disabled.

- Whether the table is partitioned. Partitioning clauses of ALTER TABLE are turned into low-level operations involving one or more tables, and those operations follow the regular rules for online DDL.

- Whether the table data must be copied, whether the table can be reorganized “in-place”, or a combination of both.

- Whether the table contains any auto-increment columns.

- What degree of locking is required, either by the nature of the underlying database operations, or a LOCK clause that you specify in the ALTER TABLE statement.

This section explains how these factors affect the different kinds of ALTER TABLE operations on InnoDB tables.

Error Conditions for Online DDL

Here are the primary reasons why an online DDL operation could fail:

- If a LOCK clause specifies a low degree of locking (SHARED or NONE) that is not compatible with the particular type of DDL operation.
- If a timeout occurs while waiting to get an exclusive lock on the table, which is needed briefly during the initial and final phases of the DDL operation.
- If the tmpdir file system runs out of disk space, while MySQL writes temporary sort files on disk during index creation.
- If the ALTER TABLE takes so long, and concurrent DML modifies the table so much, that the size of the temporary online log exceeds the value of the innodb_online_alter_log_max_size configuration option. This condition causes a DB_ONLINE_LOG_TOO_BIG error.
- If concurrent DML makes changes to the table that are allowed with the original table definition, but not with the new one. The operation only fails at the very end, when MySQL tries to apply all the changes from concurrent DML statements. For example, you might insert duplicate values into a column while a unique index is being created, or you might insert NULL values into a column while creating a primary key index on that column. The changes made by the concurrent DML take precedence, and the ALTER TABLE operation is effectively rolled back.

Although the configuration option innodb_file_per_table has a dramatic effect on the representation for an InnoDB table, all online DDL operations work equally well whether that option is enabled or disabled, and whether the table is physically located in its own .ibd file or inside the system tablespace.

InnoDB has two types of indexes: the clustered index representing all the data in the table, and optional secondary indexes to speed up queries. Since the clustered index contains the data values in its B-tree nodes, adding or dropping a clustered index does involve copying the data, and creating a new copy of the table. A secondary index, however, contains only the index key and the value of the primary key. This type of index can be created or dropped without copying the data in the clustered index. Because each secondary index contains copies of the primary key values (used to access the clustered index when needed), when you change the definition of the primary key, all secondary indexes are recreated as well.

Dropping a secondary index is simple. Only the internal InnoDB system tables and the MySQL data dictionary tables are updated to reflect the fact that the index no longer exists. InnoDB returns the storage used for the index to the tablespace that contained it, so that new indexes or additional table rows can use the space.

To add a secondary index to an existing table, InnoDB scans the table, and sorts the rows using memory buffers and temporary files in order by the values of the secondary index key columns. The B-tree is then built in key-value order, which is more efficient than inserting rows into an index in random order. Because the B-tree nodes are split when they fill, building the index in this way results in a higher fill-factor for the index, making it more efficient for subsequent access.

Primary Key and Secondary Key Indexes

Historically, the MySQL server and InnoDB have each kept their own metadata about table and index structures. The MySQL server stores this information in .frm files that are not protected by a transactional mechanism, while InnoDB has its own data dictionary as part of the system tablespace. If a DDL operation was interrupted by a crash or other unexpected event partway through, the metadata could be left inconsistent between these two locations, causing problems such as startup errors or inability to access the table that was being altered. Now that InnoDB is the default storage engine, addressing such issues is a high priority. These enhancements to DDL operations reduce the window of opportunity for such issues to occur.

14.2.11.7 How Crash Recovery Works with Online DDL

Although no data is lost if the server crashes while an ALTER TABLE statement is executing, the crash recovery process is different for clustered indexes and secondary indexes.

If the server crashes while creating an InnoDB secondary index, upon recovery, MySQL drops any partially created indexes. You must re-run the ALTER TABLE or CREATE INDEX statement.

When a crash occurs during the creation of an InnoDB clustered index, recovery is more complicated, because the data in the table must be copied to an entirely new clustered index. Remember that all InnoDB tables are stored as clustered indexes. In the following discussion, we use the word table and clustered index interchangeably.

MySQL creates the new clustered index by copying the existing data from the original InnoDB table to a temporary table that has the desired index structure. Once the data is completely copied to this temporary table, the original table is renamed with a different temporary table name. The temporary table comprising the new clustered index is renamed with the name of the original table, and the original table is dropped from the database.

If a system crash occurs while creating a new clustered index, no data is lost, but you must complete the recovery process using the temporary tables that exist during the process. Since it is rare to re-create a clustered index or re-define primary keys on large tables, or to encounter a system crash during this operation, this manual does not provide information on recovering from this scenario.

14.2.11.8 Online DDL for Partitioned InnoDB Tables

With the exception of ALTER TABLE partitioning clauses, online DDL operations for partitioned InnoDB tables follow the same rules that apply to regular InnoDB tables. Online DDL rules are outlined in Table 14.5, “Summary of Online Status for DDL Operations”.

ALTER TABLE partitioning clauses do not go through the same internal online DDL API as regular non-partitioned InnoDB tables, and are only allowed in conjunction with ALGORITHM=DEFAULT and LOCK=DEFAULT.

If you use an ALTER TABLE partitioning clause in an ALTER TABLE statement, the partitioned table will be “re-partitioned” using the ALTER TABLE COPY algorithm. In other words, a new partitioned table is created with the new partitioning scheme. The newly created table will include any changes applied by the ALTER TABLE statement and the table data will be copied into the new table structure.

If you do not change the table's partitioning using ALTER TABLE partitioning clauses or perform any other partition management in your ALTER TABLE statement, ALTER TABLE will use the INPLACE algorithm on each table partition. Be aware, however, that when INPLACE ALTER TABLE operations are performed on each partition, there will be increased demand on system resources due to operations being performed on multiple partitions.

Even though partitioning clauses of the ALTER TABLE statement do not go through the same internal online DDL API as regular non-partitioned InnoDB tables, MySQL still attempts to minimize data copying and locking where possible:

- ADD PARTITION and DROP PARTITION for tables partitioned by RANGE or LIST do not copy any existing data.
- TRUNCATE PARTITION does not copy any existing data, for all types of partitioned tables.

- Concurrent queries are allowed during ADD PARTITION and COALESCE PARTITION for tables partitioned by HASH or LIST. MySQL copies the data while holding a shared lock.
- For REORGANIZE PARTITION, REBUILD PARTITION, or ADD PARTITION or COALESCE PARTITION for a table partitioned by LINEAR HASH or LIST, concurrent queries are allowed. Data from the affected partitions is copied while holding a shared metadata (read) lock at the table level.

Note

Full-text search (FTS) and foreign keys are not supported by InnoDB partitioned tables. For more information, see Section 12.9.5, "Full-Text Restrictions" and Section 17.6.2, "Partitioning Limitations Relating to Storage Engines".

14.2.11.9 Limitations of Online DDL

Take the following limitations into account when running online DDL operations:

- During an online DDL operation that copies the table, files are written to the temporary directory ($TMPDIR on Unix, %TEMP% on Windows, or the directory specified by the --tmpdir configuration variable). Each temporary file is large enough to hold one column in the new table or index, and each one is removed as soon as it is merged into the final table or index.
- An ALTER TABLE statement that contains DROP INDEX and ADD INDEX clauses that both name the same index uses a table copy, not Fast Index Creation.
- The table is copied, rather than using Fast Index Creation when you create an index on a TEMPORARY TABLE. This has been reported as MySQL Bug #39833.
- InnoDB handles error cases when users attempt to drop indexes needed for foreign keys. See Section 14.2.17.5, "InnoDB Error Codes" for information related to error 1553.
- The ALTER TABLE clause LOCK=NONE is not allowed if there are ON...CASCADE or ON...SET NULL constraints on the table.
- During each online DDL ALTER TABLE statement, regardless of the LOCK clause, there are brief periods at the beginning and end requiring an exclusive lock on the table (the same kind of lock specified by the LOCK=EXCLUSIVE clause). Thus, an online DDL operation might wait before starting if there is a long-running transaction performing inserts, updates, deletes, or SELECT ... FOR UPDATE on that table; and an online DDL operation might wait before finishing if a similar long-running transaction was started while the ALTER TABLE was in progress.
- When running an online ALTER TABLE operation, the thread that runs the ALTER TABLE operation will apply an "online log" of DML operations that were run concurrently on the same table from other connection threads. When the DML operations are applied, it is possible to encounter a duplicate key entry error (ERROR 1062 (23000): Duplicate entry), even if the duplicate entry is only temporary and would be reverted by a later entry in the "online log". This is similar to the idea of a foreign key constraint check in InnoDB in which constraints must hold during a transaction.
- OPTIMIZE TABLE for an InnoDB table is mapped to an ALTER TABLE operation to rebuild the table and update index statistics and free unused space in the clustered index. Prior to 5.7.4, there is no online DDL support for this operation. Secondary indexes are not created as efficiently because keys are inserted in the order they appeared in the primary key. As of 5.7.4, OPTIMIZE TABLE is supported with the addition of online DDL support for rebuilding regular and partitioned InnoDB tables. For additional information, see Section 14.2.11.1, "Overview of Online DDL".
- InnoDB tables created before MySQL 5.6 do not support ALTER TABLE ... ALGORITHM=INPLACE for tables that include temporal columns (DATE, DATETIME or TIMESTAMP) and have not been

rebuilt using ALTER TABLE ... ALGORITHM=COPY. In this case, an ALTER TABLE ... ALGORITHM=INPLACE operation returns the following error:

```
ERROR 1846 (0A000): ALGORITHM=INPLACE is not supported.
Reason: Cannot change column type INPLACE. Try ALGORITHM=COPY.
```

14.2.12 InnoDB Performance Tuning

14.2.12.1 InnoDB Performance Tuning Tips

With InnoDB becoming the default storage engine in MySQL 5.5 and higher, the tips and guidelines for InnoDB tables are now part of the main optimization chapter. See Section 8.5, “Optimizing for InnoDB Tables”.

14.2.12.2 InnoDB Performance and Scalability Enhancements

This section summarizes the major InnoDB features and enhancements for performance and scalability. This information is useful to any DBA or developer who is concerned with performance and scalability. Although some of the enhancements do not require any action on your part, knowing this information can still help you diagnose performance issues more quickly and modernize systems and applications that rely on older, inefficient behavior.

Overview of InnoDB Performance

InnoDB has always been highly efficient, and includes several unique architectural elements to assure high performance and scalability. The latest InnoDB storage engine includes new features that take advantage of advances in operating systems and hardware platforms, such as multi-core processors and improved memory allocation systems. In addition, new configuration options let you better control some InnoDB internal subsystems to achieve the best performance with your workload.

Starting with MySQL 5.5 and InnoDB 1.1, the built-in InnoDB storage engine within MySQL is upgraded to the full feature set and performance of the former InnoDB Plugin. This change makes these performance and scalability enhancements available to a much wider audience than before, and eliminates the separate installation step of the InnoDB Plugin. After learning about the InnoDB performance features in this section, continue with Chapter 8, *Optimization* to learn the best practices for overall MySQL performance, and Section 8.5, “Optimizing for InnoDB Tables” in particular for InnoDB tips and guidelines.

Compression Enhancements for OLTP Workloads

Traditionally, the InnoDB compression feature was recommended primarily for read-only or read-mostly workloads, such as in a data warehouse configuration. The rise of SSD storage devices, which are fast but relatively small and expensive, makes compression attractive also for OLTP workloads: high-traffic, interactive web sites can reduce their storage requirements and their I/O operations per second (IOPS) by using compressed tables with applications that do frequent INSERT, UPDATE, and DELETE operations.

New configuration options in MySQL 5.6 let you adjust the way compression works for a particular MySQL instance, with an emphasis on performance and scalability for write-intensive operations:

- innodb_compression_level lets you turn the degree of compression up or down. A higher value lets you fit more data onto a storage device, at the expense of more CPU overhead during compression. A lower value lets you reduce CPU overhead when storage space is not critical, or you expect the data is not especially compressible.

- innodb_compression_failure_threshold_pct specifies a cutoff point for compression failures during updates to a compressed table. When this threshold is passed, MySQL begins to leave additional free space within each new compressed page, dynamically adjusting the amount of free space up to the percentage of page size specified by innodb_compression_pad_pct_max

- innodb_compression_pad_pct_max lets you adjust the maximum amount of space reserved within each page to record changes to compressed rows, without needing to compress the entire page again. The higher the value, the more changes can be recorded without recompressing the page. MySQL uses a variable amount of free space for the pages within each compressed table, only when a designated percentage of compression operations "fail" at runtime, requiring an expensive operation to split the compressed page.

Because working with compressed data sometimes involves keeping both compressed and uncompressed versions of a page in memory at the same time, when using compression with an OLTP-style workload, be prepared to increase the value of the innodb_buffer_pool_size configuration option.

For more information on MySQL data compression, see Section 14.2.7, "InnoDB Compressed Tables". For the performance aspects, especially see the section Section 14.2.7.3, "Tuning Compression for InnoDB Tables".

Optimizations for Read-Only Transactions

When a transaction is known to be read-only, InnoDB can avoid the overhead associated with setting up the transaction ID (TRX_ID field). The transaction ID is only needed for a transaction that might perform write operations or locking reads such as SELECT ... FOR UPDATE. Eliminating these unnecessary transaction IDs reduces the size of internal data structures that are consulted each time a query or DML statement constructs a read view.

Currently, InnoDB detects the read-only nature of the transaction and applies this optimization when any of the following conditions are met:

- The transaction is started with the START TRANSACTION READ ONLY statement. In this case, attempting to make any changes to the database (for InnoDB, MyISAM, or other types of tables) causes an error, and the transaction continues in read-only state:

  ```
  ERROR 1792 (25006): Cannot execute statement in a READ ONLY transaction.
  ```

  You can still make changes to session-specific temporary tables in a read-only transaction, or issue locking queries for them, because those changes and locks are not visible to any other transaction.

- The autocommit setting is turned on, so that the transaction is guaranteed to be a single statement, and the single statement making up the transaction is a "non-locking" SELECT statement. That is, a SELECT that does not use a FOR UPDATE or LOCK IN SHARED MODE clause.

- The transaction is started without the READ ONLY option, but no updates or statements that explicitly lock rows have been executed yet. Until updates or explicit locks are required, a transaction stays in read-only mode.

Thus, for a read-intensive application such as a report generator, you can tune a sequence of InnoDB queries by grouping them inside START TRANSACTION READ ONLY and COMMIT, or by turning on the autocommit setting before running the SELECT statements, or simply by avoiding any DML statements interspersed with the queries.

Note

Transactions that qualify as auto-commit, non-locking, and read-only (AC-NL-RO) are kept out of certain internal InnoDB data structures and are therefore not listed in SHOW ENGINE INNODB STATUS output. These transactions are only visible in the Information Schema.

Separate Tablespaces for InnoDB Undo Logs

This feature allows you to store the InnoDB undo log in one or more separate tablespaces outside of the system tablespace. The I/O patterns for the undo log make these tablespaces good candidates to move to SSD storage, while keeping the system tablespace on hard disk storage. Users cannot drop the separate tablespaces created to hold InnoDB undo logs, or the individual segments inside those tablespaces.

Because these files handle I/O operations formerly done inside the system tablespace, we broaden the definition of system tablespace to include these new files.

The undo logs are also known as the rollback segments.

This feature involves the following new or renamed configuration options:

- innodb_undo_tablespaces.
- innodb_undo_directory.
- innodb_rollback_segments becomes innodb_undo_logs. The old name is still available for compatibility.

Because the InnoDB undo log feature involves setting two non-dynamic startup variables (innodb_undo_tablespaces and innodb_undo_directory), this feature can only be enabled when initializing a MySQL instance.

Usage Notes

To use this feature, follow these steps:

1. Decide on a path on a fast storage device to hold the undo logs. You will specify that path as the argument to the innodb_undo_directory option in your MySQL configuration file or startup script.

2. Decide on a non-zero starting value for the innodb_undo_logs option. You can start with a relatively low value and increase it over time to examine the effect on performance.

3. Decide on a non-zero value for the innodb_undo_tablespaces option. The multiple undo logs specified by the innodb_undo_logs value are divided between this number of separate tablespaces (represented by .ibd files). This value is fixed for the life of the MySQL instance, so if you are uncertain about the optimal value, estimate on the high side.

4. Create a new MySQL instance, using the values you chose in the configuration file or in your MySQL startup script. Use a realistic workload with data volume similar to your production servers. Alternatively, use the transportable tablespaces feature to copy existing database tables to your newly configured MySQL instance. See Section 14.2.5.5, “Copying Tablespaces to Another Server (Transportable Tablespaces)” for more information.

5. Benchmark the performance of I/O intensive workloads.

6. Periodically increase the value of innodb_undo_logs and re-do the performance tests. Find the value where you stop experiencing gains in I/O performance.

7. Deploy a new production instance using the ideal settings for these options. Set it up as a slave server in a replication configuration, or transfer data from an earlier production instance.

Performance and Scalability Considerations

Keeping the undo logs in separate files allows the MySQL team to implement I/O and memory optimizations related to this transactional data. For example, because the undo data is written to disk and then rarely used (only in case of crash recovery), it does not need to be kept in the filesystem memory cache, in turn allowing a higher percentage of system memory to be devoted to the InnoDB buffer pool.

The typical SSD best practice of keeping the InnoDB system tablespace on a hard drive and moving the per-table tablespaces to SSD, is assisted by moving the undo information into separate tablespace files.

Internals

The physical tablespace files are named `undoN`, where *N* is the space ID, including leading zeros.

Currently, MySQL instances containing separate undo tablespaces cannot be downgraded to earlier releases such as MySQL 5.5 or 5.1.

Faster Extension for InnoDB Data Files

The benefits of the InnoDB file-per-table setting come with the tradeoff that each `.ibd` file is extended as the data inside the table grows. This I/O operation can be a bottleneck for busy systems with many InnoDB tables. When all InnoDB tables are stored inside the system tablespace, this extension operation happens less frequently, as space freed by `DELETE` or `TRUNCATE` operations within one table can be reused by another table.

MySQL 5.6 improves the concurrency of the extension operation, so that multiple `.ibd` files can be extended simultaneously, and this operation does not block read or write operations performed by other threads.

Non-Recursive Deadlock Detection

The code that detects deadlocks in InnoDB transactions has been modified to use a fixed-size work area rather than a recursive algorithm. The resulting detection operation is faster as a result. You do not need to do anything to take advantage of this enhancement.

Under both the old and new detection mechanisms, you might encounter a `search too deep` error that is not a true deadlock, but requires you to re-try the transaction the same way as with a deadlock.

Fast CRC32 Checksum Algorithm

You can enable the configuration option `innodb_checksum_algorithm=crc32` configuration setting to change the checksum algorithm to a faster one that scans the block 32 bits at a time rather than 8 bits at a time. When the CRC32 algorithm is enabled, data blocks that are written to disk by InnoDB contain different values in their checksum fields than before. This process could be gradual, with a mix of old and new checksum values within the same table or database.

For maximum downward compatibility, this setting is off by default:

- Current versions of MySQL Enterprise Backup (up to 3.8.0) do not support backing up tablespaces that use crc32 checksums.
- `.ibd` files containing crc32 checksums could cause problems downgrading to MySQL versions prior to 5.6.3. MySQL 5.6.3 and up recognizes either the new or old checksum values for the block as correct when reading the block from disk, ensuring that data blocks are compatible during upgrade and downgrade regardless of the algorithm setting. If data written with new checksum values is processed by an level of MySQL earlier than 5.6.3, it could be reported as corrupted.

When you set up a new MySQL instance, and can be sure that all the InnoDB data is created using the CRC32 checksum algorithm, you can use the setting innodb_checksum_algorithm=strict_crc32, which can be faster than the crc32 setting because it does not do the extra checksum calculations to support both old and new values.

The innodb_checksum_algorithm option has other values that allow it to replace the innodb_checksums option. innodb_checksum_algorithm=none is the same as innodb_checksums=OFF. innodb_checksum_algorithm=innodb is the same as innodb_checksums=ON. To avoid conflicts, remove references to innodb_checksums from your configuration file and MySQL startup scripts. The new option also accepts values strict_none and strict_innodb, again offering better performance in situations where all InnoDB data in an instance is created with the same checksum algorithm.

The following table illustrates the difference between the none, innodb, and crc32 option values, and their strict_ counterparts. none, innodb, and crc32 write the specified type checksum value into each data block, but for compatibility accept any of the other checksum values when verifying a block during a read operation. The strict_ form of each parameter only recognizes one kind of checksum, which makes verification faster but requires that all InnoDB data files in an instance be created under the identical innodb_checksum_algorithm value.

Table 14.6 Allowed Settings for innodb_checksum_algorithm

Value	Generated checksum (when writing)	Allowed checksums (when reading)
none	A constant number.	Any of the checksums generated by none, innodb, or crc32.
innodb	A checksum calculated in software, using the original algorithm from InnoDB.	Any of the checksums generated by none, innodb, or crc32.
crc32	A checksum calculated using the crc32 algorithm, possibly done with a hardware assist.	Any of the checksums generated by none, innodb, or crc32.
strict_none	A constant number	Only the checksum generated by none.
strict_innodb	A checksum calculated in software, using the original algorithm from InnoDB.	Only the checksum generated by innodb.
strict_crc32	A checksum calculated using the crc32 algorithm, possibly done with a hardware assist.	Only the checksum generated by crc32.

Faster Restart by Preloading the InnoDB Buffer Pool

After you restart a busy server, there is typically a warmup period with steadily increasing throughput, as disk pages that were in the InnoDB buffer pool are brought back into memory as the same data is queried, updated, and so on. Once the buffer pool holds a similar set of pages as before the restart, many operations are performed in memory rather than involving disk I/O, and throughput stabilizes at a high level.

This feature shortens the warmup period by immediately reloading disk pages that were in the buffer pool before the restart, rather than waiting for DML operations to access the corresponding rows. The I/O requests can be performed in large batches, making the overall I/O faster. The page loading happens in the background, and does not delay the database startup.

In addition to saving the buffer pool state at shutdown and restoring it at startup, you can also save or restore the state at any time. For example, you might save the state of the buffer pool after reaching a stable throughput under a steady workload. You might restore the previous buffer pool state after running reports or maintenance jobs that bring data pages into the buffer pool that are only needed during the time period for those operations, or after some other period with a non-typical workload.

Although the buffer pool itself could be many gigabytes in size, the data that `InnoDB` saves on disk to restore the buffer pool is tiny by comparison: just the tablespace and page IDs necessary to locate the appropriate pages on disk. This information is derived from the `information_schema` table `innodb_buffer_page_lru`.

Because the data is cached in and aged out of the buffer pool the same as with regular database operations, there is no problem if the disk pages were updated recently, or if a DML operation involves data that has not yet been loaded. The loading mechanism skips any requested pages that no longer exist.

This feature involves the configuration variables:

- `innodb_buffer_pool_dump_now`
- `innodb_buffer_pool_load_now`
- `innodb_buffer_pool_dump_at_shutdown`
- `innodb_buffer_pool_load_at_startup`
- `innodb_buffer_pool_filename`
- `innodb_buffer_pool_load_abort`

and the status variables:

- `Innodb_buffer_pool_dump_status`
- `Innodb_buffer_pool_load_status`

To save the current state of the `InnoDB` buffer pool, issue the statement:

```
SET GLOBAL innodb_buffer_pool_dump_now=ON;
```

The underlying mechanism involves a background thread that is dispatched to perform the dump and load operations.

By default, the buffer pool state is saved in a file `ib_buffer_pool` in the `InnoDB` data directory.

Disk pages from compressed tables are loaded into the buffer pool in their compressed form. Uncompression happens as usual when the page contents are accessed in the course of DML operations. Because decompression is a CPU-intensive process, it is more efficient for concurrency to perform that operation in one of the connection threads rather than the single thread that performs the buffer pool restore operation.

Example 14.10 Examples of Dumping and Restoring the InnoDB Buffer Pool

Trigger a dump of the buffer pool manually:

```
SET GLOBAL innodb_buffer_pool_dump_now=ON;
```

Specify that a dump should be taken at shutdown:

```
SET GLOBAL innodb_buffer_pool_dump_at_shutdown=ON;
```

Specify that a dump should be loaded at startup. This variable is set at server startup.

```
mysqld --innodb_buffer_pool_load_at_startup=ON;
```

Trigger a load of the buffer pool manually:

```
SET GLOBAL innodb_buffer_pool_load_now=ON;
```

Specify which filename to use for storing the dump to and loading the dump from:

```
SET GLOBAL innodb_buffer_pool_filename='filename';
```

Display progress of dump:

```
SHOW STATUS LIKE 'innodb_buffer_pool_dump_status';
```

or:

```
SELECT variable_value FROM information_schema.global_status WHERE
variable_name = 'INNODB_BUFFER_POOL_DUMP_STATUS';
```

Outputs any of: not started, Dumping buffer pool 5/7, page 237/2873, Finished at 110505 12:18:02

Display progress of load:

```
SHOW STATUS LIKE 'innodb_buffer_pool_load_status';
```

or:

```
SELECT variable_value FROM information_schema.global_status WHERE
variable_name = 'INNODB_BUFFER_POOL_LOAD_STATUS';
```

Outputs any of: not started, Loaded 123/22301 pages, Finished at 110505 12:23:24

Abort a buffer pool load:

```
SET innodb_buffer_pool_load_abort=ON;
```

Improvements to Buffer Pool Flushing

The new configuration options innodb_flush_neighbors and innodb_lru_scan_depth let you fine-tune certain aspects of the flushing process for the InnoDB buffer pool. These options primarily help write-intensive workloads. With heavy DML activity, flushing can fall behind if it is not aggressive enough, resulting in excessive memory use in the buffer pool; or, disk writes due to flushing can saturate your I/O capacity if that mechanism is too aggressive. The ideal settings depend on your workload, data access patterns, and storage configuration (for example, whether data is stored on HDD or SSD devices).

For systems with constant heavy workloads, or workloads that fluctuate widely, several new configuration options let you fine-tune the flushing behavior for InnoDB tables: innodb_adaptive_flushing_lwm, innodb_max_dirty_pages_pct_lwm, innodb_io_capacity_max, and innodb_flushing_avg_loops. These options feed into an improved formula used by the innodb_adaptive_flushing option.

The existing innodb_adaptive_flushing, innodb_io_capacity and innodb_max_dirty_pages_pct options work as before, except that they are limited or extended by other options: innodb_adaptive_flushing_lwm, innodb_io_capacity_max and innodb_max_dirty_pages_pct_lwm:

- The InnoDB adaptive flushing mechanism is not appropriate in all cases. It gives the most benefit when the redo log is in danger of filling up. The innodb_adaptive_flushing_lwm option specifies a “low water mark” percentage of redo log capacity; when that threshold is crossed, InnoDB turns on adaptive flushing even if not specified by the innodb_adaptive_flushing option.

- If flushing activity falls far behind, InnoDB can flush more aggressively than specified by innodb_io_capacity. innodb_io_capacity_max represents an upper limit on the I/O capacity used in such emergency situations, so that the spike in I/O does not consume all the capacity of the server.

- InnoDB tries to flush data from the buffer pool so that the percentage of dirty pages does not exceed the value of innodb_max_dirty_pages_pct. The default value for innodb_max_dirty_pages_pct is 75.

  Note

  The innodb_max_dirty_pages_pct setting establishes a target for flushing activity. It does not affect the rate of flushing. For information about managing the rate of flushing, see Controlling the Flushing Rate of Dirty Pages from the InnoDB Buffer Pool.

  The innodb_max_dirty_pages_pct_lwm option specifies a “low water mark” value that represents the percentage of dirty pages where pre-flushing is enabled to control the dirty page ratio and ideally prevent the percentage of dirty pages from reaching innodb_max_dirty_pages_pct. A value of innodb_max_dirty_pages_pct_lwm=0 disables the “pre-flushing” behavior.

Most of the options referenced above are most applicable to servers that run write-heavy workloads for long periods of time and have little reduced load time to catch up with changes waiting to be written to disk.

innodb_flushing_avg_loops defines the number of iterations for which InnoDB keeps the previously calculated snapshot of the flushing state, which controls how quickly adaptive flushing responds to foreground load changes. Setting a high value for innodb_flushing_avg_loops means that InnoDB keeps the previously calculated snapshot longer, so adaptive flushing responds more slowly. A high value also reduces positive feedback between foreground and background work, but when setting a high value it is important to ensure that InnoDB redo log utilization does not reach 75% (the hardcoded limit at which async flushing starts) and that the innodb_max_dirty_pages_pct setting keeps the number of dirty pages to a level that is appropriate for the workload.

Systems with consistent workloads, a large innodb_log_file_size, and small spikes that do not reach 75% redo log space utilization should use a high innodb_flushing_avg_loops value to keep flushing as smooth as possible. For systems with extreme load spikes or log files that do not provide a lot of space, consider a smaller innodb_flushing_avg_loops value. The smaller value will allow flushing to closely track the load and help avoid reaching 75% redo log space utilization.

Persistent Optimizer Statistics for InnoDB Tables

Plan stability is a desirable goal for your biggest and most important queries. InnoDB has always computed statistics for each InnoDB table to help the optimizer find the most efficient query execution plan. Now you can make these statistics persistent, so that the index usage and join order for a particular query is less likely to change.

This feature is on by default, enabled by the configuration option innodb_stats_persistent.

You control how much sampling is done to collect the statistics by setting the `innodb_stats_persistent_sample_pages` configuration option.

The configuration option `innodb_stats_auto_recalc` determines whether the statistics are calculated automatically whenever a table undergoes substantial changes (to more than 10% of the rows).

> **Note**
>
> Because of the asynchronous nature of automatic statistics recalculation (which occurs in the background), statistics may not be recalculated instantly after running a DML operation that affects more than 10% of a table, even when `innodb_stats_auto_recalc` is enabled. In some cases, statistics recalculation may be delayed by a few seconds. If up-to-date statistics are required immediately after changing significant portions of a table, run `ANALYZE TABLE` to initiate a synchronous (foreground) recalculation of statistics.

If `innodb_stats_auto_recalc` is disabled, ensure the accuracy of optimizer statistics by issuing the `ANALYZE TABLE` statement for each applicable table after making substantial changes to indexed columns. You might run this statement in your setup scripts after representative data has been loaded into the table, and run it periodically after DML operations significantly change the contents of indexed columns, or on a schedule at times of low activity. When a new index is added to an existing table, index statistics are calculated and added to the `innodb_index_stats` table regardless of the value of `innodb_stats_auto_recalc`.

> **Caution**
>
> To ensure statistics are gathered when a new index is created, either enable the `innodb_stats_auto_recalc` option, or run `ANALYZE TABLE` after creating each new index when the persistent statistics mode is enabled.

You can set `innodb_stats_persistent`, `innodb_stats_auto_recalc` options at the global level before creating a table, or use the `STATS_PERSISTENT`, `STATS_AUTO_RECALC`, and `STATS_SAMPLE_PAGES` clauses on the `CREATE TABLE` and `ALTER TABLE` statements, to override the system-wide setting and configure persistent statistics for individual tables.

Formerly, these statistics were cleared on each server restart and after some other operations, and recomputed when the table was next accessed. The statistics are computed using a random sampling technique that could produce different estimates the next time, leading to different choices in the execution plan and thus variations in query performance.

To revert to the previous method of collecting statistics that are periodically erased, run the command `ALTER TABLE` *`tbl_name`* `STATS_PERSISTENT=0`.

`InnoDB` Persistent Statistics Tables

The persistent statistics feature relies on the internally managed tables in the `mysql` database, named `innodb_table_stats` and `innodb_index_stats`. These tables are set up automatically in all install, upgrade, and build-from-source procedures.

Table 14.7 Columns of `innodb_table_stats`

Column name	Description
`database_name`	Database name
`table_name`	Table name, partition name, or subpartition name
`last_update`	A timestamp indicating the last time that `InnoDB` updated this row

Column name	Description
n_rows	The number of rows in the table
clustered_index_size	The size of the primary index, in pages
sum_of_other_index_sizes	The total size of other (non-primary) indexes, in pages

Table 14.8 Columns of innodb_index_stats

Column name	Description
database_name	Database name
table_name	Table name, partition name, or subpartition name
index_name	Index name
last_update	A timestamp indicating the last time that InnoDB updated this row
stat_name	The name of the statistic, whose value is reported in the stat_value column
stat_value	The value of the statistic that is named in stat_name column
sample_size	The number of pages sampled for the estimate provided in the stat_value column
stat_description	Description of the statistic that is named in the stat_name column

Both the innodb_table_stats and innodb_index_stats tables include a last_update column showing when InnoDB last updated index statistics, as shown in the following example:

```
mysql> select * from innodb_table_stats \G
*************************** 1. row ***************************
           database_name: sakila
              table_name: actor
             last_update: 2014-05-28 16:16:44
                  n_rows: 200
    clustered_index_size: 1
sum_of_other_index_sizes: 1
...
```

```
mysql> select * from innodb_index_stats \G
*************************** 1. row ***************************
   database_name: sakila
      table_name: actor
      index_name: PRIMARY
     last_update: 2014-05-28 16:16:44
       stat_name: n_diff_pfx01
      stat_value: 200
     sample_size: 1
     ...
```

The innodb_table_stats and innodb_index_stats tables are ordinary tables and can be updated manually. The ability to manually update statistics make it possible to force a specific query optimization plan or test alternative plans without modifying the database. If you manually update statistics, issue the FLUSH TABLE *tbl_name* command to make MySQL reload the updated statistics.

InnoDB Persistent Statistics Tables Example

The innodb_table_stats table contains one row per table. The data collected is demonstrated in the following example.

Table t1 contains a primary index (columns a, b) secondary index (columns c, d), and unique index (columns e, f):

```
CREATE TABLE t (
a INT, b INT, c INT, d INT, e INT, f INT,
PRIMARY KEY (a, b), KEY i1 (c, d), UNIQUE KEY i2uniq (e, f)
) ENGINE=INNODB;
```

After inserting five rows of sample data, the table appears as follows:

```
mysql> SELECT * FROM t1;
+---+---+------+------+------+------+
| a | b | c    | d    | e    | f    |
+---+---+------+------+------+------+
| 1 | 1 |   10 |   11 |  100 |  101 |
| 1 | 2 |   10 |   11 |  200 |  102 |
| 1 | 3 |   10 |   11 |  100 |  103 |
| 1 | 4 |   10 |   12 |  200 |  104 |
| 1 | 5 |   10 |   12 |  100 |  105 |
+---+---+------+------+------+------+
5 rows in set (0.00 sec)
```

To immediately update statistics, run ANALYZE TABLE: (if innodb_stats_auto_recalc is enabled, statistics are updated automatically within a few seconds assuming that the 10% threshold for change table rows is reached).

```
mysql> ANALYZE TABLE t1;
+---------+---------+----------+----------+
| Table   | Op      | Msg_type | Msg_text |
+---------+---------+----------+----------+
| test.t1 | analyze | status   | OK       |
+---------+---------+----------+----------+
1 row in set (0.02 sec)
```

Table statistics for table t1 show the last time InnoDB updated the table statistics (2014-03-14 14:36:34), the number of rows in the table (5), the clustered index size (1 page), and the combined size of the other indexes (2 pages).

```
mysql> SELECT * FROM mysql.innodb_table_stats WHERE table_name like 't1'\G
*************************** 1. row ***************************
           database_name: test
              table_name: t1
             last_update: 2014-03-14 14:36:34
                  n_rows: 5
    clustered_index_size: 1
sum_of_other_index_sizes: 2
1 row in set (0.00 sec)
```

The innodb_index_stats table contains multiple rows for each index. Each row in the innodb_index_stats table provides data related to a particular index statistic which is named in the stat_name column and described in the stat_description column. For example:

```
mysql> SELECT index_name, stat_name, stat_value, stat_description
    -> FROM mysql.innodb_index_stats WHERE table_name like 't1';
+------------+--------------+------------+-----------------------------------+
| index_name | stat_name    | stat_value | stat_description                  |
+------------+--------------+------------+-----------------------------------+
| PRIMARY    | n_diff_pfx01 |          1 | a                                 |
| PRIMARY    | n_diff_pfx02 |          5 | a,b                               |
| PRIMARY    | n_leaf_pages |          1 | Number of leaf pages in the index |
| PRIMARY    | size         |          1 | Number of pages in the index      |
| i1         | n_diff_pfx01 |          1 | c                                 |
| i1         | n_diff_pfx02 |          2 | c,d                               |
| i1         | n_diff_pfx03 |          2 | c,d,a                             |
| i1         | n_diff_pfx04 |          5 | c,d,a,b                           |
| i1         | n_leaf_pages |          1 | Number of leaf pages in the index |
| i1         | size         |          1 | Number of pages in the index      |
| i2uniq     | n_diff_pfx01 |          2 | e                                 |
| i2uniq     | n_diff_pfx02 |          5 | e,f                               |
| i2uniq     | n_leaf_pages |          1 | Number of leaf pages in the index |
| i2uniq     | size         |          1 | Number of pages in the index      |
+------------+--------------+------------+-----------------------------------+
14 rows in set (0.00 sec)
```

The stat_name column shows the following types of statistics:

- size: Where stat_name=size, the stat_value column displays the total number of pages in the index.

- n_leaf_pages: Where stat_name=n_leaf_pages, the stat_value column displays the number of leaf pages in the index.

- n_diff_pfx*NN*: Where stat_name=n_diff_pfx01, the stat_value column displays the number of distinct values in the first column of the index. Where stat_name=n_diff_pfx02, the stat_value column displays the number of distinct values in the first two columns of the index, and so on. Additionally, where stat_name=n_diff_pfx*NN*, the stat_description column shows a comma separated list of the index columns that are counted.

To further illustrate the n_diff_pfx*NN* statistic, which provides cardinality data, consider the t1 table example. As shown below, the t1 table is created with a primary index (columns a, b), a secondary index (columns c, d), and a unique index (columns e, f):

```
CREATE TABLE t (
  a INT, b INT, c INT, d INT, e INT, f INT,
  PRIMARY KEY (a, b), KEY i1 (c, d), UNIQUE KEY i2uniq (e, f)
) ENGINE=INNODB;
```

After inserting five rows of sample data, the table appears as follows:

```
mysql> SELECT * FROM t1;
+---+---+------+------+------+------+
| a | b | c    | d    | e    | f    |
+---+---+------+------+------+------+
| 1 | 1 |   10 |   11 |  100 |  101 |
| 1 | 2 |   10 |   11 |  200 |  102 |
| 1 | 3 |   10 |   11 |  100 |  103 |
| 1 | 4 |   10 |   12 |  200 |  104 |
| 1 | 5 |   10 |   12 |  100 |  105 |
+---+---+------+------+------+------+
5 rows in set (0.00 sec)
```

When you query the index_name, stat_name, stat_value, and stat_description where stat_name LIKE 'n_diff%', the following result set is returned:

```
mysql> SELECT index_name, stat_name, stat_value, stat_description
    -> FROM mysql.innodb_index_stats
    -> WHERE table_name like 't1' AND stat_name LIKE 'n_diff%';
+------------+--------------+------------+------------------+
| index_name | stat_name    | stat_value | stat_description |
+------------+--------------+------------+------------------+
| PRIMARY    | n_diff_pfx01 |          1 | a                |
| PRIMARY    | n_diff_pfx02 |          5 | a,b              |
| i1         | n_diff_pfx01 |          1 | c                |
| i1         | n_diff_pfx02 |          2 | c,d              |
| i1         | n_diff_pfx03 |          2 | c,d,a            |
| i1         | n_diff_pfx04 |          5 | c,d,a,b          |
| i2uniq     | n_diff_pfx01 |          2 | e                |
| i2uniq     | n_diff_pfx02 |          5 | e,f              |
+------------+--------------+------------+------------------+
8 rows in set (0.00 sec)
```

For the PRIMARY index, there are two n_diff% rows. The number of rows is equal to the number of columns in the index.

Note

For non-unique indexes, InnoDB appends the columns of the primary key.

- Where index_name=PRIMARY and stat_name=n_diff_pfx01, the stat_value is 1, which indicates that there is a single distinct value in the first column of the index (column a). The number of distinct values in column a is confirmed by viewing the data in column a in table t1, in which there is a single distinct value (1). The counted column (a) is shown in the stat_description column of the result set.

- Where index_name=PRIMARY and stat_name=n_diff_pfx02, the stat_value is 5, which indicates that there are five distinct values in the two columns of the index (a,b). The number of distinct values in columns a and b is confirmed by viewing the data in columns a and b in table t1, in which there are five distinct values: (1,1), (1,2), (1,3), (1,4) and (1,5). The counted columns (a,b) are shown in the stat_description column of the result set.

For the secondary index (i1), there are four n_diff% rows. Only two columns are defined for the secondary index (c,d) but there are four n_diff% rows for the secondary index because InnoDB suffixes all non-unique indexes with the primary key. As a result, there are four n_diff% rows instead of two to account for the both the secondary index columns (c,d) and the primary key columns (a,b).

- Where index_name=i1 and stat_name=n_diff_pfx01, the stat_value is 1, which indicates that there is a single distinct value in the first column of the index (column c). The number of distinct values in column c is confirmed by viewing the data in column c in table t1, in which there is a single distinct value: (10). The counted column (c) is shown in the stat_description column of the result set.

- Where index_name=i1 and stat_name=n_diff_pfx02, the stat_value is 2, which indicates that there are two distinct values in the first two columns of the index (c,d). The number of distinct values in columns c an d is confirmed by viewing the data in columns c and d in table t1, in which there are two distinct values: (10,11) and (10,12). The counted columns (c,d) are shown in the stat_description column of the result set.

- Where index_name=i1 and stat_name=n_diff_pfx03, the stat_value is 2, which indicates that there are two distinct values in the first three columns of the index (c,d,a). The number of distinct values in columns c, d, and a is confirmed by viewing the data in column c, d, and a in table t1, in which there are two distinct values: (10,11,1) and (10,12,1). The counted columns (c,d,a) are shown in the stat_description column of the result set.

- Where index_name=i1 and stat_name=n_diff_pfx04, the stat_value is 5, which indicates that there are five distinct values in the four columns of the index (c,d,a,b). The number of distinct values in columns c, d, a and b is confirmed by viewing the data in columns c, d, a, and b in table t1, in which there are five distinct values: (10,11,1,1), (10,11,1,2), (10,11,1,3), (10,12,1,4) and (10,12,1,5). The counted columns (c,d,a,b) are shown in the stat_description column of the result set.

For the unique index (i2uniq), there are two n_diff% rows.

- Where index_name=i2uniq and stat_name=n_diff_pfx01, the stat_value is 2, which indicates that there are two distinct values in the first column of the index (column e). The number of distinct values in column e is confirmed by viewing the data in column e in table t1, in which there are two distinct values: (100) and (200). The counted column (e) is shown in the stat_description column of the result set.

- Where index_name=i2uniq and stat_name=n_diff_pfx02, the stat_value is 5, which indicates that there are five distinct values in the two columns of the index (e,f). The number of distinct values in columns e and f is confirmed by viewing the data in columns e and f in table t1, in which there are five distinct values: (100,101), (200,102), (100,103), (200,104) and (100,105). The counted columns (e,f) are shown in the stat_description column of the result set.

Retrieving Index Size Using the innodb_index_stats Table

The size of indexes for tables, partitions, or subpartitions can be retrieved using the innodb_index_stats table. In the following example, index sizes are retrieved for table t1:

```
mysql> SELECT SUM(stat_value) pages, index_name,
    -> SUM(stat_value)*@@innodb_page_size size
    -> FROM mysql.innodb_index_stats WHERE table_name='t1'
    -> AND stat_name = 'size' GROUP BY index_name;
+-------+------------+-------+
| pages | index_name | size  |
+-------+------------+-------+
|     1 | PRIMARY    | 16384 |
|     1 | i1         | 16384 |
|     1 | i2uniq     | 16384 |
+-------+------------+-------+
3 rows in set (0.00 sec)
```

For partitions or subpartitions, the same query with a modified WHERE clause can be used to retrieve index sizes. For example, the following query retrieves index sizes for partitions of table t1:

```
mysql> SELECT SUM(stat_value) pages, index_name,
    -> SUM(stat_value)*@@innodb_page_size size
    -> FROM mysql.innodb_index_stats WHERE table_name like 't1#P%'
    -> AND AND stat_name = 'size' GROUP BY index_name;
```

Faster Locking for Improved Scalability

In MySQL and InnoDB, multiple threads of execution access shared data structures. InnoDB synchronizes these accesses with its own implementation of mutexes and read/write locks. InnoDB has historically protected the internal state of a read/write lock with an InnoDB mutex. On Unix and Linux platforms, the internal state of an InnoDB mutex is protected by a Pthreads mutex, as in IEEE Std 1003.1c (POSIX.1c).

On many platforms, there is a more efficient way to implement mutexes and read/write locks. Atomic operations can often be used to synchronize the actions of multiple threads more efficiently than Pthreads. Each operation to acquire or release a lock can be done in fewer CPU instructions, and thus result in less wasted time when threads are contending for access to shared data structures. This in turn means greater scalability on multi-core platforms.

InnoDB implements mutexes and read/write locks with the built-in functions provided by the GNU Compiler Collection (GCC) for atomic memory access instead of using the Pthreads approach previously used. More specifically, an InnoDB that is compiled with GCC version 4.1.2 or later uses the atomic builtins instead of a pthread_mutex_t to implement InnoDB mutexes and read/write locks.

On 32-bit Microsoft Windows, InnoDB has implemented mutexes (but not read/write locks) with hand-written assembler instructions. Beginning with Microsoft Windows 2000, functions for Interlocked Variable

Access are available that are similar to the built-in functions provided by GCC. On Windows 2000 and higher, InnoDB makes use of the Interlocked functions. Unlike the old hand-written assembler code, the new implementation supports read/write locks and 64-bit platforms.

Solaris 10 introduced library functions for atomic operations, and InnoDB uses these functions by default. When MySQL is compiled on Solaris 10 with a compiler that does not support the built-in functions provided by the GNU Compiler Collection (GCC) for atomic memory access, InnoDB uses the library functions.

This change improves the scalability of InnoDB on multi-core systems. This feature is enabled out-of-the-box on the platforms where it is supported. You do not have to set any parameter or option to take advantage of the improved performance. On platforms where the GCC, Windows, or Solaris functions for atomic memory access are not available, InnoDB uses the traditional Pthreads method of implementing mutexes and read/write locks.

When MySQL starts, InnoDB writes a message to the log file indicating whether atomic memory access is used for mutexes, for mutexes and read/write locks, or neither. If suitable tools are used to build InnoDB and the target CPU supports the atomic operations required, InnoDB uses the built-in functions for mutexing. If, in addition, the compare-and-swap operation can be used on thread identifiers (`pthread_t`), then InnoDB uses the instructions for read-write locks as well.

Note: If you are building from source, ensure that the build process properly takes advantage of your platform capabilities.

For more information about the performance implications of locking, see Section 8.10, “Optimizing Locking Operations”.

Using Operating System Memory Allocators

When InnoDB was developed, the memory allocators supplied with operating systems and run-time libraries were often lacking in performance and scalability. At that time, there were no memory allocator libraries tuned for multi-core CPUs. Therefore, InnoDB implemented its own memory allocator in the `mem` subsystem. This allocator is guarded by a single mutex, which may become a bottleneck. InnoDB also implements a wrapper interface around the system allocator (`malloc` and `free`) that is likewise guarded by a single mutex.

Today, as multi-core systems have become more widely available, and as operating systems have matured, significant improvements have been made in the memory allocators provided with operating systems. New memory allocators perform better and are more scalable than they were in the past. The leading high-performance memory allocators include `Hoard`, `libumem`, `mtmalloc`, `ptmalloc`, `tbbmalloc`, and `TCMalloc`. Most workloads, especially those where memory is frequently allocated and released (such as multi-table joins), benefit from using a more highly tuned memory allocator as opposed to the internal, InnoDB-specific memory allocator.

You can control whether InnoDB uses its own memory allocator or an allocator of the operating system, by setting the value of the system configuration parameter `innodb_use_sys_malloc` in the MySQL option file (`my.cnf` or `my.ini`). If set to `ON` or `1` (the default), InnoDB uses the `malloc` and `free` functions of the underlying system rather than manage memory pools itself. This parameter is not dynamic, and takes effect only when the system is started. To continue to use the InnoDB memory allocator, set `innodb_use_sys_malloc` to `0`.

When the InnoDB memory allocator is disabled, InnoDB ignores the value of the parameter `innodb_additional_mem_pool_size`. The InnoDB memory allocator uses an additional memory pool for satisfying allocation requests without having to fall back to the system memory allocator. When the InnoDB memory allocator is disabled, all such allocation requests are fulfilled by the system memory allocator.

On Unix-like systems that use dynamic linking, replacing the memory allocator may be as easy as making the environment variable LD_PRELOAD or LD_LIBRARY_PATH point to the dynamic library that implements the allocator. On other systems, some relinking may be necessary. Please refer to the documentation of the memory allocator library of your choice.

Since InnoDB cannot track all memory use when the system memory allocator is used (innodb_use_sys_malloc is ON), the section “BUFFER POOL AND MEMORY” in the output of the SHOW ENGINE INNODB STATUS command only includes the buffer pool statistics in the “Total memory allocated”. Any memory allocated using the mem subsystem or using ut_malloc is excluded.

> **Note**
>
> innodb_use_sys_malloc and innodb_additional_mem_pool_size were deprecated in MySQL 5.6.3 and are removed in MySQL 5.7.4.

For more information about the performance implications of InnoDB memory usage, see Section 8.9, “Buffering and Caching”.

Controlling InnoDB Change Buffering

When INSERT, UPDATE, and DELETE operations are done to a table, often the values of indexed columns (particularly the values of secondary keys) are not in sorted order, requiring substantial I/O to bring secondary indexes up to date. InnoDB has an insert buffer that caches changes to secondary index entries when the relevant page is not in the buffer pool, thus avoiding I/O operations by not reading in the page from the disk. The buffered changes are merged when the page is loaded to the buffer pool, and the updated page is later flushed to disk using the normal mechanism. The InnoDB main thread merges buffered changes when the server is nearly idle, and during a slow shutdown.

Because it can result in fewer disk reads and writes, this feature is most valuable for workloads that are I/O-bound, for example applications with a high volume of DML operations such as bulk inserts.

However, the insert buffer occupies a part of the buffer pool, reducing the memory available to cache data pages. If the working set almost fits in the buffer pool, or if your tables have relatively few secondary indexes, it may be useful to disable insert buffering. If the working set entirely fits in the buffer pool, insert buffering does not impose any extra overhead, because it only applies to pages that are not in the buffer pool.

You can control the extent to which InnoDB performs insert buffering with the system configuration parameter innodb_change_buffering. You can turn on and off buffering for inserts, delete operations (when index records are initially marked for deletion) and purge operations (when index records are physically deleted). An update operation is represented as a combination of an insert and a delete. In MySQL 5.5 and higher, the default value is changed from inserts to all.

The allowed values of innodb_change_buffering are:

- **all**

  The default value: buffer inserts, delete-marking operations, and purges.

- **none**

  Do not buffer any operations.

- **inserts**

  Buffer insert operations.

- **deletes**

Buffer delete-marking operations.

- **changes**

  Buffer both inserts and delete-marking.

- **purges**

  Buffer the physical deletion operations that happen in the background.

You can set the value of this parameter in the MySQL option file (my.cnf or my.ini) or change it dynamically with the SET GLOBAL command, which requires the SUPER privilege. Changing the setting affects the buffering of new operations; the merging of already buffered entries is not affected.

For more information about speeding up INSERT, UPDATE, and DELETE statements, see Section 8.2.2, “Optimizing DML Statements”.

Controlling Adaptive Hash Indexing

If a table fits almost entirely in main memory, the fastest way to perform queries on it is to use hash indexes rather than B-tree lookups. MySQL monitors searches on each index defined for an InnoDB table. If it notices that certain index values are being accessed frequently, it automatically builds an in-memory hash table for that index. See Adaptive Hash Indexes for background information and usage guidelines for the adaptive hash index feature and the innodb_adaptive_hash_index configuration option.

Changes Regarding Thread Concurrency

InnoDB uses operating system threads to process requests from user transactions. (Transactions may issue many requests to InnoDB before they commit or roll back.) On modern operating systems and servers with multi-core processors, where context switching is efficient, most workloads run well without any limit on the number of concurrent threads. Scalability improvements in MySQL 5.5 and up reduce the need to limit the number of concurrently executing threads inside InnoDB.

In situations where it is helpful to minimize context switching between threads, InnoDB can use a number of techniques to limit the number of concurrently executing operating system threads (and thus the number of requests that are processed at any one time). When InnoDB receives a new request from a user session, if the number of threads concurrently executing is at a pre-defined limit, the new request sleeps for a short time before it tries again. A request that cannot be rescheduled after the sleep is put in a first-in/first-out queue and eventually is processed. Threads waiting for locks are not counted in the number of concurrently executing threads.

You can limit the number of concurrent threads by setting the configuration parameter innodb_thread_concurrency. Once the number of executing threads reaches this limit, additional threads sleep for a number of microseconds, set by the configuration parameter innodb_thread_sleep_delay, before being placed into the queue.

Previously, it required experimentation to find the optimal value for innodb_thread_sleep_delay, and the optimal value could change depending on the workload. In MySQL 5.6.3 and higher, you can set the configuration option innodb_adaptive_max_sleep_delay to the highest value you would allow for innodb_thread_sleep_delay, and InnoDB automatically adjusts innodb_thread_sleep_delay up or down depending on the current thread-scheduling activity. This dynamic adjustment helps the thread scheduling mechanism to work smoothly during times when the system is lightly loaded and when it is operating near full capacity.

The default value for innodb_thread_concurrency and the implied default limit on the number of concurrent threads has been changed in various releases of MySQL and InnoDB. Currently, the default value of innodb_thread_concurrency is 0, so that by default there is no limit on the number of concurrently executing threads, as shown in Table 14.9, “Changes to innodb_thread_concurrency”.

Table 14.9 Changes to innodb_thread_concurrency

InnoDB Version	MySQL Version	Default value	Default limit of concurrent threads	Value to allow unlimited threads
Built-in	Earlier than 5.1.11	20	No limit	20 or higher
Built-in	5.1.11 and newer	8	8	0
InnoDB before 1.0.3	(corresponding to Plugin)	8	8	0
InnoDB 1.0.3 and newer	(corresponding to Plugin)	0	No limit	0

Note that InnoDB causes threads to sleep only when the number of concurrent threads is limited. When there is no limit on the number of threads, all contend equally to be scheduled. That is, if innodb_thread_concurrency is 0, the value of innodb_thread_sleep_delay is ignored.

When there is a limit on the number of threads, InnoDB reduces context switching overhead by permitting multiple requests made during the execution of a single SQL statement to enter InnoDB without observing the limit set by innodb_thread_concurrency. Since an SQL statement (such as a join) may comprise multiple row operations within InnoDB, InnoDB assigns “tickets” that allow a thread to be scheduled repeatedly with minimal overhead.

When a new SQL statement starts, a thread has no tickets, and it must observe innodb_thread_concurrency. Once the thread is entitled to enter InnoDB, it is assigned a number of tickets that it can use for subsequently entering InnoDB. If the tickets run out, innodb_thread_concurrency is observed again and further tickets are assigned. The number of tickets to assign is specified by the global option innodb_concurrency_tickets, which is 500 by default. A thread that is waiting for a lock is given one ticket once the lock becomes available.

The correct values of these variables depend on your environment and workload. Try a range of different values to determine what value works for your applications. Before limiting the number of concurrently executing threads, review configuration options that may improve the performance of InnoDB on multi-core and multi-processor computers, such as innodb_use_sys_malloc and innodb_adaptive_hash_index.

For general performance information about MySQL thread handling, see Section 8.11.5.1, “How MySQL Uses Threads for Client Connections”.

Changes in the Read-Ahead Algorithm

A read-ahead request is an I/O request to prefetch multiple pages in the buffer pool asynchronously, in anticipation that these pages will be needed soon. The requests bring in all the pages in one extent. InnoDB uses two read-ahead algorithms to improve I/O performance:

Linear read-ahead is a technique that predicts what pages might be needed soon based on pages in the buffer pool being accessed sequentially. You control when InnoDB performs a read-ahead operation by adjusting the number of sequential page accesses required to trigger an asynchronous read request, using the configuration parameter innodb_read_ahead_threshold. Before this parameter was added, InnoDB would only calculate whether to issue an asynchronous prefetch request for the entire next extent when it read in the last page of the current extent.

The new configuration parameter innodb_read_ahead_threshold controls how sensitive InnoDB is in detecting patterns of sequential page access. If the number of pages read sequentially from an extent is greater than or equal to `innodb_read_ahead_threshold`, InnoDB initiates an asynchronous read-ahead operation of the entire following extent. It can be set to any value from 0-64. The default value is 56. The higher the value, the more strict the access pattern check. For example, if you set the value to 48, InnoDB triggers a linear read-ahead request only when 48 pages in the current extent have been accessed sequentially. If the value is 8, InnoDB would trigger an asynchronous read-ahead even if as few as 8 pages in the extent were accessed sequentially. You can set the value of this parameter in the MySQL configuration file, or change it dynamically with the `SET GLOBAL` command, which requires the `SUPER` privilege.

Random read-ahead is a technique that predicts when pages might be needed soon based on pages already in the buffer pool, regardless of the order in which those pages were read. If 13 consecutive pages from the same extent are found in the buffer pool, InnoDB asynchronously issues a request to prefetch the remaining pages of the extent. This feature was initially turned off in MySQL 5.5. It is available once again starting in MySQL 5.1.59 and 5.5.16 and higher, turned off by default. To enable this feature, set the configuration variable `innodb_random_read_ahead`.

The `SHOW ENGINE INNODB STATUS` command displays statistics to help you evaluate the effectiveness of the read-ahead algorithm. Statistics include counter information for the `Innodb_buffer_pool_read_ahead`, `Innodb_buffer_pool_read_ahead_evicted`, and `Innodb_buffer_pool_read_ahead_rnd` global status variables. This information can be useful when fine-tuning the `innodb_random_read_ahead` setting.

For more information about I/O performance, see Section 8.5.7, “Optimizing `InnoDB` Disk I/O” and Section 8.11.3, “Optimizing Disk I/O”.

Multiple Background InnoDB I/O Threads

InnoDB uses background threads to service various types of I/O requests. You can configure the number of background threads that service read and write I/O on data pages, using the configuration parameters `innodb_read_io_threads` and `innodb_write_io_threads`. These parameters signify the number of background threads used for read and write requests respectively. They are effective on all supported platforms. You can set the value of these parameters in the MySQL option file (`my.cnf` or `my.ini`); you cannot change them dynamically. The default value for these parameters is `4` and the permissible values range from `1-64`.

The purpose of this change is to make InnoDB more scalable on high end systems. Each background thread can handle up to 256 pending I/O requests. A major source of background I/O is the read-ahead requests. InnoDB tries to balance the load of incoming requests in such way that most of the background threads share work equally. InnoDB also attempts to allocate read requests from the same extent to the same thread to increase the chances of coalescing the requests together. If you have a high end I/O subsystem and you see more than 64 × `innodb_read_io_threads` pending read requests in `SHOW ENGINE INNODB STATUS`, you might gain by increasing the value of `innodb_read_io_threads`.

For more information about InnoDB I/O performance, see Section 8.5.7, “Optimizing `InnoDB` Disk I/O”.

Asynchronous I/O on Linux

Starting in InnoDB 1.1 with MySQL 5.5, the asynchronous I/O capability that InnoDB has had on Windows systems is now available on Linux systems. (Other Unix-like systems continue to use synchronous I/O calls.) This feature improves the scalability of heavily I/O-bound systems, which typically show many pending reads/writes in the output of the command `SHOW ENGINE INNODB STATUS\G`.

Running with a large number of `InnoDB` I/O threads, and especially running multiple such instances on the same server machine, can exceed capacity limits on Linux systems. In this case, you can fix the error:

```
EAGAIN: The specified maxevents exceeds the user's limit of available events.
```

by writing a higher limit to /proc/sys/fs/aio-max-nr.

In general, if a problem with the asynchronous I/O subsystem in the OS prevents InnoDB from starting, set the option innodb_use_native_aio=0 in the configuration file. This new configuration option applies to Linux systems only, and cannot be changed once the server is running.

For more information about InnoDB I/O performance, see Section 8.5.7, “Optimizing InnoDB Disk I/O”.

Group Commit

InnoDB, like any other ACID-compliant database engine, flushes the redo log of a transaction before it is committed. Historically, InnoDB used group commit functionality to group multiple such flush requests together to avoid one flush for each commit. With group commit, InnoDB issues a single write to the log file to perform the commit action for multiple user transactions that commit at about the same time, significantly improving throughput.

Group commit in InnoDB worked until MySQL 4.x, and works once again with MySQL 5.1 with the InnoDB Plugin, and MySQL 5.5 and higher. The introduction of support for the distributed transactions and Two Phase Commit (2PC) in MySQL 5.0 interfered with the InnoDB group commit functionality. This issue is now resolved.

The group commit functionality inside InnoDB works with the Two Phase Commit protocol in MySQL. Re-enabling of the group commit functionality fully ensures that the ordering of commit in the MySQL binlog and the InnoDB logfile is the same as it was before. It means it is **totally safe to use the MySQL Enterprise Backup product with InnoDB 1.0.4** (that is, the InnoDB Plugin with MySQL 5.1) and above. When the binlog is enabled, you typically also set the configuration option sync_binlog=0, because group commit for the binary log is only supported if it is set to 0.

Group commit is transparent; you do not need to do anything to take advantage of this significant performance improvement.

For more information about performance of COMMIT and other transactional operations, see Section 8.5.2, “Optimizing InnoDB Transaction Management”.

Controlling the InnoDB Master Thread I/O Rate

The master thread in InnoDB is a thread that performs various tasks in the background. Most of these tasks are I/O related, such as flushing dirty pages from the buffer pool or writing changes from the insert buffer to the appropriate secondary indexes. The master thread attempts to perform these tasks in a way that does not adversely affect the normal working of the server. It tries to estimate the free I/O bandwidth available and tune its activities to take advantage of this free capacity. Historically, InnoDB has used a hard coded value of 100 IOPs (input/output operations per second) as the total I/O capacity of the server.

The parameter innodb_io_capacity indicates the overall I/O capacity available to InnoDB. This parameter should be set to approximately the number of I/O operations that the system can perform per second. The value depends on your system configuration. When innodb_io_capacity is set, the master threads estimates the I/O bandwidth available for background tasks based on the set value. Setting the value to 100 reverts to the old behavior.

You can set the value of innodb_io_capacity to any number 100 or greater. The default value is 200, reflecting that the performance of typical modern I/O devices is higher than in the early days of MySQL. Typically, values around the previous default of 100 are appropriate for consumer-level storage devices, such as hard drives up to 7200 RPMs. Faster hard drives, RAID configurations, and SSDs benefit from higher values.

The innodb_io_capacity setting is a total limit for all buffer pool instances. When dirty pages are flushed, the innodb_io_capacity limit is divided equally among buffer pool instances. For more information, see the innodb_io_capacity system variable description.

You can set the value of this parameter in the MySQL option file (my.cnf or my.ini) or change it dynamically with the SET GLOBAL command, which requires the SUPER privilege.

Formerly, the InnoDB master thread also performed any needed purge operations. In MySQL 5.6.5 and higher, those I/O operations are moved to other background threads, whose number is controlled by the innodb_purge_threads configuration option.

For more information about InnoDB I/O performance, see Section 8.5.7, “Optimizing InnoDB Disk I/O”.

Controlling the Flushing Rate of Dirty Pages from the InnoDB Buffer Pool

InnoDB performs certain tasks in the background, including flushing of dirty pages (those pages that have been changed but are not yet written to the database files) from the buffer pool. InnoDB flushes buffer pool pages if the percentage of dirty pages in the buffer pool exceeds innodb_max_dirty_pages_pct. As of MySQL 5.7.5, InnoDB flushes buffer pool pages if the percentage of dirty pages in the buffer pool is greater than or equal to innodb_max_dirty_pages_pct (Bug#13029450).

InnoDB uses a new algorithm to estimate the required rate of flushing, based on the speed of redo log generation and the current rate of flushing. The intent is to smooth overall performance by ensuring that buffer flush activity keeps up with the need to keep the buffer pool “clean”. Automatically adjusting the rate of flushing can help to avoid sudden dips in throughput, when excessive buffer pool flushing limits the I/O capacity available for ordinary read and write activity.

InnoDB uses its log files in a circular fashion. Before reusing a portion of a log file, InnoDB flushes to disk all dirty buffer pool pages whose redo entries are contained in that portion of the log file, a process known as a sharp checkpoint. If a workload is write-intensive, it generates a lot of redo information, all written to the log file. If all available space in the log files is used up, a sharp checkpoint occurs, causing a temporary reduction in throughput. This situation can happen even though innodb_max_dirty_pages_pct is not reached.

InnoDB uses a heuristic-based algorithm to avoid such a scenario, by measuring the number of dirty pages in the buffer pool and the rate at which redo is being generated. Based on these numbers, InnoDB decides how many dirty pages to flush from the buffer pool each second. This self-adapting algorithm is able to deal with sudden changes in the workload.

Internal benchmarking has also shown that this algorithm not only maintains throughput over time, but can also improve overall throughput significantly.

Because adaptive flushing is a new feature that can significantly affect the I/O pattern of a workload, a new configuration parameter lets you turn off this feature. The default value of the boolean parameter innodb_adaptive_flushing is TRUE, enabling the new algorithm. You can set the value of this parameter in the MySQL option file (my.cnf or my.ini) or change it dynamically with the SET GLOBAL command, which requires the SUPER privilege.

For more information about InnoDB I/O performance, see Section 8.5.7, “Optimizing InnoDB Disk I/O”.

Using the PAUSE Instruction in InnoDB Spin Loops

Synchronization inside InnoDB frequently involves the use of spin loops: while waiting, InnoDB executes a tight loop of instructions repeatedly to avoid having the InnoDB process and threads be rescheduled by the operating system. If the spin loops are executed too quickly, system resources are wasted, imposing a performance penalty on transaction throughput. Most modern processors implement the PAUSE instruction for use in spin loops, so the processor can be more efficient.

InnoDB uses a PAUSE instruction in its spin loops on all platforms where such an instruction is available. This technique increases overall performance with CPU-bound workloads, and has the added benefit of minimizing power consumption during the execution of the spin loops.

You do not have to do anything to take advantage of this performance improvement.

For performance considerations for InnoDB locking operations, see Section 8.10, “Optimizing Locking Operations”.

Control of Spin Lock Polling

Many InnoDB mutexes and rw-locks are reserved for a short time. On a multi-core system, it can be more efficient for a thread to continuously check if it can acquire a mutex or rw-lock for a while before sleeping. If the mutex or rw-lock becomes available during this polling period, the thread can continue immediately, in the same time slice. However, too-frequent polling by multiple threads of a shared object can cause “cache ping pong”, different processors invalidating portions of each others' cache. InnoDB minimizes this issue by waiting a random time between subsequent polls. The delay is implemented as a busy loop.

You can control the maximum delay between testing a mutex or rw-lock using the parameter innodb_spin_wait_delay. The duration of the delay loop depends on the C compiler and the target processor. (In the 100MHz Pentium era, the unit of delay was one microsecond.) On a system where all processor cores share a fast cache memory, you might reduce the maximum delay or disable the busy loop altogether by setting innodb_spin_wait_delay=0. On a system with multiple processor chips, the effect of cache invalidation can be more significant and you might increase the maximum delay.

The default value of innodb_spin_wait_delay is 6. The spin wait delay is a dynamic global parameter that you can specify in the MySQL option file (my.cnf or my.ini) or change at runtime with the command SET GLOBAL innodb_spin_wait_delay=*delay*, where *delay* is the desired maximum delay. Changing the setting requires the SUPER privilege.

For performance considerations for InnoDB locking operations, see Section 8.10, “Optimizing Locking Operations”.

Making the Buffer Pool Scan Resistant

Rather than using a strictly LRU algorithm, InnoDB uses a technique to minimize the amount of data that is brought into the buffer pool and never accessed again. The goal is to make sure that frequently accessed (“hot”) pages remain in the buffer pool, even as read-ahead and full table scans bring in new blocks that might or might not be accessed afterward.

Newly read blocks are inserted into the middle of the LRU list. All newly read pages are inserted at a location that by default is 3/8 from the tail of the LRU list. The pages are moved to the front of the list (the most-recently used end) when they are accessed in the buffer pool for the first time. Thus pages that are never accessed never make it to the front portion of the LRU list, and “age out” sooner than with a strict LRU approach. This arrangement divides the LRU list into two segments, where the pages downstream of the insertion point are considered “old” and are desirable victims for LRU eviction.

For an explanation of the inner workings of the InnoDB buffer pool and the specifics of its LRU replacement algorithm, see Section 8.9.1, “The InnoDB Buffer Pool”.

You can control the insertion point in the LRU list, and choose whether InnoDB applies the same optimization to blocks brought into the buffer pool by table or index scans. The configuration parameter innodb_old_blocks_pct controls the percentage of “old” blocks in the LRU list. The default value of innodb_old_blocks_pct is 37, corresponding to the original fixed ratio of 3/8. The value range is 5 (new pages in the buffer pool age out very quickly) to 95 (only 5% of the buffer pool is reserved for hot pages, making the algorithm close to the familiar LRU strategy).

The optimization that keeps the buffer pool from being churned by read-ahead can avoid similar problems due to table or index scans. In these scans, a data page is typically accessed a few times in quick succession and is never touched again. The configuration parameter innodb_old_blocks_time specifies the time window (in milliseconds) after the first access to a page during which it can be accessed without being moved to the front (most-recently used end) of the LRU list. The default value of innodb_old_blocks_time is 1000. Increasing this value makes more and more blocks likely to age out faster from the buffer pool.

Both innodb_old_blocks_pct and innodb_old_blocks_time are dynamic, global and can be specified in the MySQL option file (my.cnf or my.ini) or changed at runtime with the SET GLOBAL command. Changing the setting requires the SUPER privilege.

To help you gauge the effect of setting these parameters, the SHOW ENGINE INNODB STATUS command reports additional statistics. The BUFFER POOL AND MEMORY section looks like:

```
Total memory allocated 1107296256; in additional pool allocated 0
Dictionary memory allocated 80360
Buffer pool size   65535
Free buffers       0
Database pages     63920
Old database pages 23600
Modified db pages  34969
Pending reads 32
Pending writes: LRU 0, flush list 0, single page 0
Pages made young 414946, not young 2930673
1274.75 youngs/s, 16521.90 non-youngs/s
Pages read 486005, created 3178, written 160585
2132.37 reads/s, 3.40 creates/s, 323.74 writes/s
Buffer pool hit rate 950 / 1000, young-making rate 30 / 1000 not 392 / 1000
Pages read ahead 1510.10/s, evicted without access 0.00/s
LRU len: 63920, unzip_LRU len: 0
I/O sum[43690]:cur[221], unzip sum[0]:cur[0]
```

- Old database pages is the number of pages in the “old” segment of the LRU list.
- Pages made young and not young is the total number of “old” pages that have been made young or not respectively.
- youngs/s and non-young/s is the rate at which page accesses to the “old” pages have resulted in making such pages young or otherwise respectively since the last invocation of the command.
- young-making rate and not provides the same rate but in terms of overall buffer pool accesses instead of accesses just to the “old” pages.

Note

Per second averages provided in InnoDB Monitor output are based on the elapsed time between the current time and the last time InnoDB Monitor output was printed.

Because the effects of these parameters can vary widely based on your hardware configuration, your data, and the details of your workload, always benchmark to verify the effectiveness before changing these settings in any performance-critical or production environment.

In mixed workloads where most of the activity is OLTP type with periodic batch reporting queries which result in large scans, setting the value of innodb_old_blocks_time during the batch runs can help keep the working set of the normal workload in the buffer pool.

When scanning large tables that cannot fit entirely in the buffer pool, setting innodb_old_blocks_pct to a small value keeps the data that is only read once from consuming a significant portion of the buffer pool.

For example, setting innodb_old_blocks_pct=5 restricts this data that is only read once to 5% of the buffer pool.

When scanning small tables that do fit into memory, there is less overhead for moving pages around within the buffer pool, so you can leave innodb_old_blocks_pct at its default value, or even higher, such as innodb_old_blocks_pct=50.

The effect of the innodb_old_blocks_time parameter is harder to predict than the innodb_old_blocks_pct parameter, is relatively small, and varies more with the workload. To arrive at an optimal value, conduct your own benchmarks if the performance improvement from adjusting innodb_old_blocks_pct is not sufficient.

For more information about the InnoDB buffer pool, see Section 8.9.1, “The InnoDB Buffer Pool”.

Improvements to Crash Recovery Performance

A number of optimizations speed up certain steps of the recovery that happens on the next startup after a crash. In particular, scanning the redo log and applying the redo log are faster than in MySQL 5.1 and earlier, due to improved algorithms for memory management. You do not need to take any actions to take advantage of this performance enhancement. If you kept the size of your redo log files artificially low because recovery took a long time, you can consider increasing the file size.

For more information about InnoDB recovery, see Section 14.2.14.1, “The InnoDB Recovery Process”.

Integration with the MySQL Performance Schema

Starting with InnoDB 1.1 with MySQL 5.5, you can profile certain internal InnoDB operations using the MySQL Performance Schema feature. This type of tuning is primarily for expert users, those who push the limits of MySQL performance, read the MySQL source code, and evaluate optimization strategies to overcome performance bottlenecks. DBAs can also use this feature for capacity planning, to see whether their typical workload encounters any performance bottlenecks with a particular combination of CPU, RAM, and disk storage; and if so, to judge whether performance can be improved by increasing the capacity of some part of the system.

To use this feature to examine InnoDB performance:

- You must be running MySQL 5.5 or higher with the Performance Schema feature available and enabled, as described in Section 20.2, “Performance Schema Configuration”. Since the Performance Schema feature introduces some performance overhead, you should use it on a test or development system rather than on a production system.

- You must be running InnoDB 1.1 or higher.

- You must be generally familiar with how to use the Performance Schema feature, for example to query tables in the performance_schema database.

- Examine the following kinds of InnoDB objects by querying the appropriate performance_schema tables. The items associated with InnoDB all contain the substring innodb in the EVENT_NAME column.

  For the definitions of the *_instances tables, see Section 20.9.3, “Performance Schema Instance Tables”. For the definitions of the *_summary_* tables, see Section 20.9.12, “Performance Schema Summary Tables”. For the definition of the thread table, see Section 20.9.13, “Performance Schema Miscellaneous Tables”. For the definition of the *_current_* and *_history_* tables, see Section 20.9.4, “Performance Schema Wait Event Tables”.

  - Mutexes in the mutex_instances table. (Mutexes and RW-locks related to the InnoDB buffer pool are not included in this coverage; the same applies to the output of the SHOW ENGINE INNODB MUTEX command.)

  - RW-locks in the rwlock_instances table.
  - File I/O operations in the file_instances, file_summary_by_event_name, and file_summary_by_instance tables.
  - Threads in the PROCESSLIST table.

- During performance testing, examine the performance data in the events_waits_current and events_waits_history_long tables. If you are interested especially in InnoDB-related objects, use the clause WHERE EVENT_NAME LIKE '%innodb%' to see just those entries; otherwise, examine the performance statistics for the overall MySQL server.

For more information about the MySQL Performance Schema, see Chapter 20, *MySQL Performance Schema*.

Improvements to Performance from Multiple Buffer Pools

This performance enhancement is primarily useful for people with a large buffer pool size, typically in the multi-gigabyte range. To take advantage of this speedup, you must set the new innodb_buffer_pool_instances configuration option, and you might also adjust the innodb_buffer_pool_size value.

When the InnoDB buffer pool is large, many data requests can be satisfied by retrieving from memory. You might encounter bottlenecks from multiple threads trying to access the buffer pool at once. Starting in InnoDB 1.1 and MySQL 5.5, you can enable multiple buffer pools to minimize this contention. Each page that is stored in or read from the buffer pool is assigned to one of the buffer pools randomly, using a hashing function. Each buffer pool manages its own free lists, flush lists, LRUs, and all other data structures connected to a buffer pool, and is protected by its own buffer pool mutex.

To enable this feature, set the innodb_buffer_pool_instances configuration option to a value greater than 1 (the default) up to 64 (the maximum). This option takes effect only when you set the innodb_buffer_pool_size to a size of 1 gigabyte or more. The total size you specify is divided among all the buffer pools. For best efficiency, specify a combination of innodb_buffer_pool_instances and innodb_buffer_pool_size so that each buffer pool instance is at least 1 gigabyte.

For more information about the InnoDB buffer pool, see Section 8.9.1, “The InnoDB Buffer Pool”.

Better Scalability with Multiple Rollback Segments

Starting in InnoDB 1.1 with MySQL 5.5, the limit on concurrent transactions is greatly expanded, removing a bottleneck with the InnoDB rollback segment that affected high-capacity systems. The limit applies to concurrent transactions that change any data; read-only transactions do not count against that maximum.

The single rollback segment was divided into 128 segments. As of MySQL 5.7.2, 32 of the 128 segments are reserved for temporary table transactions. This leaves 96 segments, each of which can support up to 1023 transactions that perform writes, for a total of approximately 96K concurrent transactions. The original transaction limit prior to InnoDB 1.1 with MySQL 5.5 was 1023.

Each transaction is assigned to one of the rollback segments, and remains tied to that rollback segment for the duration. This enhancement improves both scalability (higher number of concurrent transactions) and performance (less contention when different transactions access the rollback segments).

To take advantage of this feature, you do not need to create any new database or tables, or reconfigure anything. You must do a slow shutdown before upgrading from MySQL 5.1 or earlier, or some time afterward. InnoDB makes the required changes inside the system tablespace automatically, the first time you restart after performing a slow shutdown.

If your workload was not constrained by the original limit of 1023 concurrent transactions, you can reduce the number of rollback segments used within a MySQL instance or within a session by setting the configuration option innodb_rollback_segments.

For more information about performance of InnoDB under high transactional load, see Section 8.5.2, “Optimizing InnoDB Transaction Management”.

Better Scalability with Improved Purge Scheduling

The purge operations (a type of garbage collection) that InnoDB performs automatically is now done in one or more separate threads, rather than as part of the master thread. This change improves scalability, because the main database operations run independently from maintenance work happening in the background.

To control this feature, increase the value of the configuration option innodb_purge_threads. If DML action is concentrated on a single table or a few tables, keep the setting low so that the threads do not contend with each other for access to the busy tables. If DML operations are spread across many tables, increase the setting. Its maximum is 32.

There is another related configuration option, innodb_purge_batch_size with a default of 20 and maximum of 5000. This option is mainly intended for experimentation and tuning of purge operations, and should not be interesting to typical users.

For more information about InnoDB I/O performance, see Section 8.5.7, “Optimizing InnoDB Disk I/O”.

Improved Log Sys Mutex

This is another performance improvement that comes for free, with no user action or configuration needed. The details here are intended for performance experts who delve into the InnoDB source code, or interpret reports with keywords such as “mutex” and “log_sys”.

The mutex known as the log sys mutex has historically done double duty, controlling access to internal data structures related to log records and the LSN, as well as pages in the buffer pool that are changed when a mini-transaction is committed. Starting in InnoDB 1.1 with MySQL 5.5, these two kinds of operations are protected by separate mutexes, with a new log_buf mutex controlling writes to buffer pool pages due to mini-transactions.

For performance considerations for InnoDB locking operations, see Section 8.10, “Optimizing Locking Operations”.

Separate Flush List Mutex

Starting with InnoDB 1.1 with MySQL 5.5, concurrent access to the buffer pool is faster. Operations involving the flush list, a data structure related to the buffer pool, are now controlled by a separate mutex and do not block access to the buffer pool. You do not need to configure anything to take advantage of this speedup; it is fully automatic.

For more information about the InnoDB buffer pool, see Section 8.9.1, “The InnoDB Buffer Pool”.

memcached Plugin for InnoDB

The memcached daemon is frequently used as an in-memory caching layer in front of a MySQL database server. Now MySQL allows direct access to InnoDB tables using the familiar memcached protocol and client libraries. Instead of formulating queries in SQL, you can perform simple get, set, and increment operations that avoid the performance overhead of SQL parsing and constructing a query optimization plan. You can also access the underlying InnoDB tables through SQL to load data, generate reports, or perform multi-step transactional computations.

This technique allows the data to be stored in MySQL for reliability and consistency, while coding application logic that uses the database as a fast key-value store.

This feature combines the best of both worlds:

- Data that is written using the `memcached` protocol is transparently written to an InnoDB table, without going through the MySQL SQL layer. You can control the frequency of writes to achieve higher raw performance when updating non-critical data.
- Data that is requested data through the `memcached` protocol is transparently queried from an InnoDB table, without going through the MySQL SQL layer.
- Subsequent requests for the same data will be served from the `InnoDB` buffer pool. The buffer pool handles the in-memory caching. You can tune the performance of data-intensive operations using the familiar `InnoDB` configuration options.
- InnoDB can handle composing and decomposing multiple column values into a single `memcached` item value, reducing the amount of string parsing and concatenation required in your application. For example, you might store a string value `2|4|6|8` in the `memcached` cache, and InnoDB splits that value based on a separator character, then stores the result into four numeric columns.

For details on using this NoSQL-style interface to MySQL, see Section 14.2.16, “`InnoDB` Integration with `memcached`”. For additional background on `memcached` and considerations for writing applications for its API, see Section 15.6, “Using MySQL with `memcached`”.

Online DDL

This feature is a continuation of the “Fast Index Creation” feature introduced in `InnoDB` Fast Index Creation. Now you can perform other kinds of DDL operations on InnoDB tables online: that is, with minimal delay for operations on that table, without rebuilding the entire table, or both. This enhancement improves responsiveness and availability in busy production environments, where making a table unavailable for minutes or hours whenever its column definitions change is not practical.

For full details, see Section 14.2.11, “`InnoDB` and Online DDL”.

The DDL operations enhanced by this feature are these variations on the `ALTER TABLE` statement:

- Create secondary indexes: `CREATE INDEX *name* ON *table* (*col_list*)` or `ALTER TABLE *table* ADD INDEX *name* (*col_list*)`. (Creating a primary key or a `FULLTEXT` index still requires locking the table.)

  Drop secondary indexes: `DROP INDEX *name* ON *table*;` or `ALTER TABLE *table* DROP INDEX *name*`

  Creating and dropping secondary indexes on `InnoDB` tables has avoided the table-copying behavior since the days of MySQL 5.1 with the `InnoDB` Plugin. Now, the table remains available for read and write operations while the index is being created or dropped. The `CREATE TABLE` or `DROP TABLE` statement only finishes after all transactions that are modifying the table are completed, so that the initial state of the index reflects the most recent contents of the table.

  Previously, modifying the table while an index was being created or dropped typically resulted in a deadlock that cancelled the insert, update, or delete statement on the table.

- Changing the auto-increment value for a column: `ALTER TABLE *table* AUTO_INCREMENT=*next_value*;`

  Especially in a distributed system using replication or sharding, you sometimes reset the auto-increment counter for a table to a specific value. The next row inserted into the table uses the specified value for its auto-increment column. You might also use this technique in a data warehousing environment where you periodically empty all the tables and reload them, and you can restart the auto-increment sequence from 1.

- Adding or dropping a foreign key constraint:

  ```
  ALTER TABLE tbl1 ADD CONSTRAINT fk_name FOREIGN KEY index (col1) REFERENCES tbl2(col2) referential_actio
  ALTER TABLE tbl DROP FOREIGN KEY fk_name;
  ```

  Dropping a foreign key can be performed online with the foreign_key_checks option enabled or disabled. Creating a foreign key online requires foreign_key_checks to be disabled.

  If you do not know the names of the foreign key constraints on a particular table, issue the following statement and find the constraint name in the CONSTRAINT clause for each foreign key:

  ```
  show create table table\G
  ```

  Or, query the information_schema.table_constraints table and use the constraint_name and constraint_type columns to identify the foreign key names.

  As a consequence of this enhancement, you can now also drop a foreign key and its associated index in a single statement, which previously required separate statements in a strict order:

  ```
  ALTER TABLE table DROP FOREIGN KEY constraint, DROP INDEX index;
  ```

- Renaming a column: ALTER TABLE *tbl* CHANGE *old_col_name new_col_name datatype*

  When you keep the same data type and only change the column name, this operation can always be performed online. As part of this enhancement, you can now rename a column that is part of a foreign key constraint, which was not allowed before.

- Some other ALTER TABLE operations are non-blocking, and are faster than before because the table-copying operation is optimized, even though a table copy is still required:

  - Changing the ROW_FORMAT or KEY_BLOCK_SIZE properties for a table.
  - Changing the nullable status for a column.
  - Adding, dropping, or reordering columns.

Note

As your database schema evolves with new columns, data types, constraints, indexes, and so on, keep your CREATE TABLE statements up to date with the latest table definitions. Even with the performance improvements of online DDL, it is more efficient to create stable database structures at the beginning, rather than creating part of the schema and then issuing ALTER TABLE statements afterward.

The main exception to this guideline is for secondary indexes on tables with large numbers of rows. It is typically most efficient to create the table with all details specified except the secondary indexes, load the data, then create the secondary indexes.

Whatever sequence of CREATE TABLE, CREATE INDEX, ALTER TABLE, and similar statements went into putting a table together, you can capture the SQL needed to reconstruct the current form of the table by issuing the statement SHOW

CREATE TABLE *table*\G (uppercase \G required for tidy formatting). This output shows clauses such as numeric precision, NOT NULL, and CHARACTER SET that are sometimes added behind the scenes, and you might otherwise leave out when cloning the table on a new system or setting up foreign key columns with identical type.

14.2.12.3 InnoDB INFORMATION_SCHEMA tables

The INFORMATION_SCHEMA is a MySQL feature that helps you monitor server activity to diagnose capacity and performance issues. Several InnoDB-related INFORMATION_SCHEMA tables (INNODB_CMP, INNODB_CMP_RESET, INNODB_CMPMEM, INNODB_CMPMEM_RESET, INNODB_TRX, INNODB_LOCKS and INNODB_LOCK_WAITS) contain live information about compressed InnoDB tables, the compressed InnoDB buffer pool, all transactions currently executing inside InnoDB, the locks that transactions hold and those that are blocking transactions waiting for access to a resource (a table or row).

This section describes the InnoDB-related Information Schema tables and shows some examples of their use.

Information Schema Tables about Compression

Two new pairs of Information Schema tables can give you some insight into how well compression is working overall. One pair of tables contains information about the number of compression operations and the amount of time spent performing compression. Another pair of tables contains information on the way memory is allocated for compression.

INNODB_CMP and INNODB_CMP_RESET

The INNODB_CMP and INNODB_CMP_RESET tables contain status information on the operations related to compressed tables, which are covered in Section 14.2.7, “InnoDB Compressed Tables”. The compressed page size is in the column PAGE_SIZE.

These two tables have identical contents, but reading from INNODB_CMP_RESET resets the statistics on compression and uncompression operations. For example, if you archive the output of INNODB_CMP_RESET every 60 minutes, you see the statistics for each hourly period. If you monitor the output of INNODB_CMP (making sure never to read INNODB_CMP_RESET), you see the cumulated statistics since InnoDB was started.

For the table definition, see Table 19.1, “Columns of INNODB_CMP and INNODB_CMP_RESET”.

INNODB_CMPMEM and INNODB_CMPMEM_RESET

The INNODB_CMPMEM and INNODB_CMPMEM_RESET tables contain status information on the compressed pages that reside in the buffer pool. Please consult Section 14.2.7, “InnoDB Compressed Tables” for further information on compressed tables and the use of the buffer pool. The INNODB_CMP and INNODB_CMP_RESET tables should provide more useful statistics on compression.

Internal Details

InnoDB uses a buddy allocator system to manage memory allocated to pages of various sizes, from 1KB to 16KB. Each row of the two tables described here corresponds to a single page size.

These two tables have identical contents, but reading from INNODB_CMPMEM_RESET resets the statistics on relocation operations. For example, if every 60 minutes you archived the output of INNODB_CMPMEM_RESET, it would show the hourly statistics. If you never read INNODB_CMPMEM_RESET and monitored the output of INNODB_CMPMEM instead, it would show the cumulated statistics since InnoDB was started.

For the table definition, see Table 19.3, “Columns of INNODB_CMPMEM and INNODB_CMPMEM_RESET”.

Using the Compression Information Schema Tables

Example 14.11 Using the Compression Information Schema Tables

The following is sample output from a database that contains compressed tables (see Section 14.2.7, “InnoDB Compressed Tables”, INNODB_CMP, INNODB_CMP_PER_INDEX, and INNODB_CMPMEM).

The following table shows the contents of INFORMATION_SCHEMA.INNODB_CMP under a light workload. The only compressed page size that the buffer pool contains is 8K. Compressing or uncompressing pages has consumed less than a second since the time the statistics were reset, because the columns COMPRESS_TIME and UNCOMPRESS_TIME are zero.

page size	compress ops	compress ops ok	compress time	uncompress ops	uncompress time
1024	0	0	0	0	0
2048	0	0	0	0	0
4096	0	0	0	0	0
8192	1048	921	0	61	0
16384	0	0	0	0	0

According to INNODB_CMPMEM, there are 6169 compressed 8KB pages in the buffer pool. The only other allocated block size is 64 bytes. The smallest PAGE_SIZE in INNODB_CMPMEM is used for block descriptors of those compressed pages for which no uncompressed page exists in the buffer pool. We see that there are 5910 such pages. Indirectly, we see that 259 (6169-5910) compressed pages also exist in the buffer pool in uncompressed form.

The following table shows the contents of INFORMATION_SCHEMA.INNODB_CMPMEM under a light workload. Some memory is unusable due to fragmentation of the memory allocator for compressed pages: SUM(PAGE_SIZE*PAGES_FREE)=6784. This is because small memory allocation requests are fulfilled by splitting bigger blocks, starting from the 16K blocks that are allocated from the main buffer pool, using the buddy allocation system. The fragmentation is this low because some allocated blocks have been relocated (copied) to form bigger adjacent free blocks. This copying of SUM(PAGE_SIZE*RELOCATION_OPS) bytes has consumed less than a second (SUM(RELOCATION_TIME)=0).

page size	pages used	pages free	relocation ops	relocation time
64	5910	0	2436	0
128	0	1	0	0
256	0	0	0	0
512	0	1	0	0
1024	0	0	0	0
2048	0	1	0	0
4096	0	1	0	0
8192	6169	0	5	0
16384	0	0	0	0

Information Schema Tables about Transactions

Three InnoDB-related Information Schema tables make it easy to monitor transactions and diagnose possible locking problems. The three tables are INNODB_TRX, INNODB_LOCKS, and INNODB_LOCK_WAITS.

- INNODB_TRX

  Contains information about every transaction currently executing inside InnoDB, including whether the transaction is waiting for a lock, when the transaction started, and the particular SQL statement the transaction is executing.

  For the table definition, see Table 19.4, “INNODB_TRX Columns”.

- INNODB_LOCKS

  Each transaction in InnoDB that is waiting for another transaction to release a lock (INNODB_TRX.TRX_STATE='LOCK WAIT') is blocked by exactly one “blocking lock request”. That blocking lock request is for a row or table lock held by another transaction in an incompatible mode. The waiting or blocked transaction cannot proceed until the other transaction commits or rolls back, thereby releasing the requested lock. For every blocked transaction, INNODB_LOCKS contains one row that describes each lock the transaction has requested, and for which it is waiting. INNODB_LOCKS also contains one row for each lock that is blocking another transaction, whatever the state of the transaction that holds the lock ('RUNNING', 'LOCK WAIT', 'ROLLING BACK' or 'COMMITTING'). The lock that is blocking a transaction is always held in a mode (read vs. write, shared vs. exclusive) incompatible with the mode of requested lock.

  For the table definition, see Table 19.5, “INNODB_LOCKS Columns”.

- INNODB_LOCK_WAITS

  Using this table, you can tell which transactions are waiting for a given lock, or for which lock a given transaction is waiting. This table contains one or more rows for each *blocked* transaction, indicating the lock it has requested and any locks that are blocking that request. The REQUESTED_LOCK_ID refers to the lock that a transaction is requesting, and the BLOCKING_LOCK_ID refers to the lock (held by another transaction) that is preventing the first transaction from proceeding. For any given blocked transaction, all rows in INNODB_LOCK_WAITS have the same value for REQUESTED_LOCK_ID and different values for BLOCKING_LOCK_ID.

  For the table definition, see Table 19.6, “INNODB_LOCK_WAITS Columns”.

Using the Transaction Information Schema Tables

Example 14.12 Identifying Blocking Transactions

It is sometimes helpful to be able to identify which transaction is blocking another. You can use the Information Schema tables to find out which transaction is waiting for another, and which resource is being requested.

Suppose you have the following scenario, with three users running concurrently. Each user (or session) corresponds to a MySQL thread, and executes one transaction after another. Consider the state of the system when these users have issued the following commands, but none has yet committed its transaction:

- User A:

```
BEGIN;
SELECT a FROM t FOR UPDATE;
SELECT SLEEP(100);
```

- User B:

```
SELECT b FROM t FOR UPDATE;
```

- User C:

```
SELECT c FROM t FOR UPDATE;
```

In this scenario, you can use this query to see who is waiting for whom:

```
SELECT r.trx_id waiting_trx_id,
       r.trx_mysql_thread_id waiting_thread,
       r.trx_query waiting_query,
       b.trx_id blocking_trx_id,
       b.trx_mysql_thread_id blocking_thread,
       b.trx_query blocking_query
   FROM       information_schema.innodb_lock_waits w
   INNER JOIN information_schema.innodb_trx b  ON
    b.trx_id = w.blocking_trx_id
  INNER JOIN information_schema.innodb_trx r  ON
    r.trx_id = w.requesting_trx_id;
```

waiting trx id	waiting thread	waiting query	blocking trx id	blocking thread	blocking query
A4	6	SELECT b FROM t FOR UPDATE	A3	5	SELECT SLEEP(100)
A5	7	SELECT c FROM t FOR UPDATE	A3	5	SELECT SLEEP(100)
A5	7	SELECT c FROM t FOR UPDATE	A4	6	SELECT b FROM t FOR UPDATE

In the above result, you can identify users by the “waiting query” or “blocking query”. As you can see:

- User B (trx id 'A4', thread 6) and User C (trx id 'A5', thread 7) are both waiting for User A (trx id 'A3', thread 5).

- User C is waiting for User B as well as User A.

You can see the underlying data in the tables INNODB_TRX, INNODB_LOCKS, and INNODB_LOCK_WAITS.

The following table shows some sample contents of INFORMATION_SCHEMA.INNODB_TRX.

trx id	trx state	trx started	trx requested lock id	trx wait started	trx weight	trx mysql thread id	trx query
A3	RUNNING	2008-01-15 16:44:54	NULL	NULL	2	5	SELECT SLEEP(100)
A4	LOCK WAIT	2008-01-15 16:45:09	A4:1:3:2	2008-01-15 16:45:09	2	6	SELECT b FROM t FOR UPDATE

trx id	trx state	trx started	trx requested lock id	trx wait started	trx weight	trx mysql thread id	trx query
A5	LOCK WAIT	2008-01-15 16:45:14	A5:1:3:2	2008-01-15 16:45:14	2	7	SELECT c FROM t FOR UPDATE

The following table shows some sample contents of INFORMATION_SCHEMA.INNODB_LOCKS.

lock id	lock trx id	lock mode	lock type	lock table	lock index	lock space	lock page	lock rec	lock data
A3:1:3:2	A3	X	RECORD	\`test\`.\`t\`	\`PRIMARY\`	1	3	2	0x0200
A4:1:3:2	A4	X	RECORD	\`test\`.\`t\`	\`PRIMARY\`	1	3	2	0x0200
A5:1:3:2	A5	X	RECORD	\`test\`.\`t\`	\`PRIMARY\`	1	3	2	0x0200

The following table shows some sample contents of INFORMATION_SCHEMA.INNODB_LOCK_WAITS.

requesting trx id	requested lock id	blocking trx id	blocking lock id
A4	A4:1:3:2	A3	A3:1:3:2
A5	A5:1:3:2	A3	A3:1:3:2
A5	A5:1:3:2	A4	A4:1:3:2

Example 14.13 More Complex Example of Transaction Data in Information Schema Tables

Sometimes you would like to correlate the internal InnoDB locking information with session-level information maintained by MySQL. For example, you might like to know, for a given InnoDB transaction ID, the corresponding MySQL session ID and name of the user that may be holding a lock, and thus blocking another transaction.

The following output from the INFORMATION_SCHEMA tables is taken from a somewhat loaded system.

As can be seen in the following tables, there are several transactions running.

The following INNODB_LOCKS and INNODB_LOCK_WAITS tables shows that:

- Transaction *77F* (executing an INSERT) is waiting for transactions *77E*, *77D* and *77B* to commit.
- Transaction *77E* (executing an INSERT) is waiting for transactions *77D* and *77B* to commit.
- Transaction *77D* (executing an INSERT) is waiting for transaction *77B* to commit.
- Transaction *77B* (executing an INSERT) is waiting for transaction *77A* to commit.
- Transaction *77A* is running, currently executing SELECT.
- Transaction *E56* (executing an INSERT) is waiting for transaction *E55* to commit.
- Transaction *E55* (executing an INSERT) is waiting for transaction *19C* to commit.
- Transaction *19C* is running, currently executing an INSERT.

Note that there may be an inconsistency between queries shown in the two tables INNODB_TRX.TRX_QUERY and PROCESSLIST.INFO. The current transaction ID for a thread, and the query being executed in that transaction, may be different in these two tables for any given thread. See Possible Inconsistency with PROCESSLIST for an explanation.

The following table shows the contents of INFORMATION_SCHEMA.PROCESSLIST in a system running a heavy workload.

ID	USER	HOST	DB	COMMAND	TIME	STATE	INFO
384	root	localhost	test	Query	10	update	insert into t2 values …
257	root	localhost	test	Query	3	update	insert into t2 values …
130	root	localhost	test	Query	0	update	insert into t2 values …
61	root	localhost	test	Query	1	update	insert into t2 values …
8	root	localhost	test	Query	1	update	insert into t2 values …
4	root	localhost	test	Query	0	preparing	SELECT * FROM processlist
2	root	localhost	test	Sleep	566		NULL

The following table shows the contents of INFORMATION_SCHEMA.INNODB_TRX in a system running a heavy workload.

trx id	trx state	trx started	trx requested lock id	trx wait started	trx weight	trx mysql thread id	trx query
77F	LOCK WAIT	2008-01-15 13:10:16	*77F*:806	2008-01-15 13:10:16	1	876	insert into t09 (D, B, C) values …
77E	LOCK WAIT	2008-01-15 13:10:16	*77E*:806	2008-01-15 13:10:16	1	875	insert into t09 (D, B, C) values …
77D	LOCK WAIT	2008-01-15 13:10:16	*77D*:806	2008-01-15 13:10:16	1	874	insert into t09 (D, B, C) values …
77B	LOCK WAIT	2008-01-15 13:10:16	*77B*:733:12:1	2008-01-15 13:10:16	4	873	insert into t09 (D, B, C) values …
77A	RUNNING	2008-01-15 13:10:16	NULL	NULL	4	872	select b, c from t09 where …
E56	LOCK WAIT	2008-01-15 13:10:06	*E56*:743:6:2	2008-01-15 13:10:06	5	384	insert into t2 values …
E55	LOCK WAIT	2008-01-15 13:10:06	*E55*:743:38:2	2008-01-15 13:10:13	965	257	insert into t2 values …
19C	RUNNING	2008-01-15 13:09:10	NULL	NULL	2900	130	insert into t2 values …

trx id	trx state	trx started	trx requested lock id	trx wait started	trx weight	trx mysql thread id	trx query
E15	RUNNING	2008-01-15 13:08:59	NULL	NULL	5395	61	insert into t2 values …
51D	RUNNING	2008-01-15 13:08:47	NULL	NULL	9807	8	insert into t2 values …

The following table shows the contents of INFORMATION_SCHEMA.INNODB_LOCK_WAITS in a system running a heavy workload.

requesting trx id	requested lock id	blocking trx id	blocking lock id
77F	*77F*:806	*77E*	*77E*:806
77F	*77F*:806	*77D*	*77D*:806
77F	*77F*:806	*77B*	*77B*:806
77E	*77E*:806	*77D*	*77D*:806
77E	*77E*:806	*77B*	*77B*:806
77D	*77D*:806	*77B*	*77B*:806
77B	*77B*:733:12:1	*77A*	*77A*:733:12:1
E56	*E56*:743:6:2	*E55*	*E55*:743:6:2
E55	*E55*:743:38:2	*19C*	*19C*:743:38:2

The following table shows the contents of INFORMATION_SCHEMA.INNODB_LOCKS in a system running a heavy workload.

lock id	lock trx id	lock mode	lock type	lock table	lock index	lock space	lock page	lock rec	lock data
77F:806	*77F*	AUTO_INC	TABLE	\`test\`.\`t09\`	NULL	NULL	NULL	NULL	NULL
77E:806	*77E*	AUTO_INC	TABLE	\`test\`.\`t09\`	NULL	NULL	NULL	NULL	NULL
77D:806	*77D*	AUTO_INC	TABLE	\`test\`.\`t09\`	NULL	NULL	NULL	NULL	NULL
77B:806	*77B*	AUTO_INC	TABLE	\`test\`.\`t09\`	NULL	NULL	NULL	NULL	NULL
77B:733:12:1	*77B*	X	RECORD	\`test\`.\`t09\`	\`PRIMARY\`	733	12	1	supremum pseudo-record
77A:733:12:1	*77A*	X	RECORD	\`test\`.\`t09\`	\`PRIMARY\`	733	12	1	supremum pseudo-record
E56:743:6:2	*E56*	S	RECORD	\`test\`.\`t2\`	\`PRIMARY\`	743	6	2	0, 0
E55:743:6:2	*E55*	X	RECORD	\`test\`.\`t2\`	\`PRIMARY\`	743	6	2	0, 0

lock id	lock trx id	lock mode	lock type	lock table	lock index	lock space	lock page	lock rec	lock data
E55:743 :38:2	*E55*	S	RECORD	\`test\` .\`t2\`	\`PRIMARY\`	743	38	2	1922, 1922
19C:743 :38:2	*19C*	X	RECORD	\`test\` .\`t2\`	\`PRIMARY\`	743	38	2	1922, 1922

Information Schema Tables about Full-Text Search

A set of related INFORMATION_SCHEMA tables contains information about FULLTEXT search indexes on InnoDB tables:

- INNODB_FT_CONFIG
- INNODB_FT_INDEX_TABLE
- INNODB_FT_INDEX_CACHE
- INNODB_FT_DEFAULT_STOPWORD
- INNODB_FT_DELETED
- INNODB_FT_BEING_DELETED

Special Locking Considerations for InnoDB INFORMATION_SCHEMA Tables

Understanding InnoDB Locking

When a transaction updates a row in a table, or locks it with SELECT FOR UPDATE, InnoDB establishes a list or queue of locks on that row. Similarly, InnoDB maintains a list of locks on a table for table-level locks transactions hold. If a second transaction wants to update a row or lock a table already locked by a prior transaction in an incompatible mode, InnoDB adds a lock request for the row to the corresponding queue. For a lock to be acquired by a transaction, all incompatible lock requests previously entered into the lock queue for that row or table must be removed (the transactions holding or requesting those locks either commit or roll back).

A transaction may have any number of lock requests for different rows or tables. At any given time, a transaction may be requesting a lock that is held by another transaction, in which case it is blocked by that other transaction. The requesting transaction must wait for the transaction that holds the blocking lock to commit or rollback. If a transaction is not waiting for a a lock, it is in the 'RUNNING' state. If a transaction is waiting for a lock, it is in the 'LOCK WAIT' state.

The INNODB_LOCKS table holds one or more row for each 'LOCK WAIT' transaction, indicating any lock requests that are preventing its progress. This table also contains one row describing each lock in a queue of locks pending for a given row or table. The INNODB_LOCK_WAITS table shows which locks already held by a transaction are blocking locks requested by other transactions.

Granularity of INFORMATION_SCHEMA Data

The data exposed by the transaction and locking tables represent a glimpse into fast-changing data. This is not like other (user) tables, where the data changes only when application-initiated updates occur. The underlying data is internal system-managed data, and can change very quickly.

For performance reasons, and to minimize the chance of misleading JOINs between the INFORMATION_SCHEMA tables, InnoDB collects the required transaction and locking information into an intermediate buffer whenever a SELECT on any of the tables is issued. This buffer is refreshed only if more than 0.1 seconds has elapsed since the last time the buffer was read. The data needed to fill the three tables is fetched atomically and consistently and is saved in this global internal buffer, forming a point-in-time “snapshot”. If multiple table accesses occur within 0.1 seconds (as they almost certainly do when MySQL processes a join among these tables), then the same snapshot is used to satisfy the query.

A correct result is returned when you JOIN any of these tables together in a single query, because the data for the three tables comes from the same snapshot. Because the buffer is not refreshed with every query of any of these tables, if you issue separate queries against these tables within a tenth of a second, the results are the same from query to query. On the other hand, two separate queries of the same or different tables issued more than a tenth of a second apart may see different results, since the data come from different snapshots.

Because InnoDB must temporarily stall while the transaction and locking data is collected, too frequent queries of these tables can negatively impact performance as seen by other users.

As these tables contain sensitive information (at least INNODB_LOCKS.LOCK_DATA and INNODB_TRX.TRX_QUERY), for security reasons, only the users with the PROCESS privilege are allowed to SELECT from them.

Possible Inconsistency with PROCESSLIST

As just described, while the transaction and locking data is correct and consistent when these INFORMATION_SCHEMA tables are populated. For example, the query in INNODB_TRX is always consistent with the rest of INNODB_TRX, INNODB_LOCKS and INNODB_LOCK_WAITS when the data comes from the same snapshot. However, the underlying data changes so fast that similar glimpses at other, similarly fast-changing data, may not be in synchrony. Thus, you should be careful in comparing the data in the InnoDB transaction and locking tables with that in the PROCESSLIST table. The data from the PROCESSLIST table does not come from the same snapshot as the data about locking and transactions. Even if you issue a single SELECT (joining INNODB_TRX and PROCESSLIST, for example), the content of those tables is generally not consistent. INNODB_TRX may reference rows that are not present in PROCESSLIST or the currently executing SQL query of a transaction, shown in INNODB_TRX.TRX_QUERY may differ from the one in PROCESSLIST.INFO.

14.2.12.4 InnoDB Monitors

InnoDB monitors provide information about the InnoDB internal state. This information is useful for performance tuning.

InnoDB Monitor Types

There are four types of InnoDB monitors:

- The standard InnoDB Monitor displays the following types of information:
  - Table and record locks held by each active transaction.
  - Lock waits of a transaction.
  - Semaphore waits of threads.
  - Pending file I/O requests.
  - Buffer pool statistics.
  - Purge and insert buffer merge activity of the main InnoDB thread.

- The InnoDB Lock Monitor is like the standard InnoDB Monitor but also provides extensive lock information.
- The InnoDB Tablespace Monitor prints a list of file segments in the shared tablespace and validates the tablespace allocation data structures.
- The InnoDB Table Monitor prints the contents of the InnoDB internal data dictionary.

Note

The Tablespace Monitor and Table Monitor were deprecated in MySQL 5.6.3 and have been removed in MySQL 5.7.4. For the Tablespace Monitor, equivalent functionality will be introduced before the GA release of MySQL 5.7. For the Table Monitor, equivalent information can be obtained from InnoDB INFORMATION_SCHEMA tables.

For additional information about InnoDB monitors, see:

- Mark Leith: InnoDB Table and Tablespace Monitors

Enabling InnoDB Monitors

When you enable InnoDB monitors for periodic output, InnoDB writes their output to the mysqld server standard error output (stderr). In this case, no output is sent to clients. When switched on, InnoDB monitors print data about every 15 seconds. Server output usually is directed to the error log (see Section 5.2.2, “The Error Log”). This data is useful in performance tuning. On Windows, start the server from a command prompt in a console window with the --console option if you want to direct the output to the window rather than to the error log.

InnoDB sends diagnostic output to stderr or to files rather than to stdout or fixed-size memory buffers, to avoid potential buffer overflows. As a side effect, the output of SHOW ENGINE INNODB STATUS is written to a status file in the MySQL data directory every fifteen seconds. The name of the file is innodb_status.*pid*, where *pid* is the server process ID. InnoDB removes the file for a normal shutdown. If abnormal shutdowns have occurred, instances of these status files may be present and must be removed manually. Before removing them, you might want to examine them to see whether they contain useful information about the cause of abnormal shutdowns. The innodb_status.*pid* file is created only if the configuration option innodb-status-file=1 is set.

InnoDB monitors should be enabled only when you actually want to see monitor information because output generation does result in some performance decrement. Also, if you enable monitor output by creating the associated table, your error log may become quite large if you forget to remove the table later.

Note

To assist with troubleshooting, InnoDB temporarily enables standard InnoDB Monitor output under certain conditions. For more information, see Section 14.2.17, “InnoDB Troubleshooting”.

Each monitor begins with a header containing a timestamp and the monitor name. For example:

```
=================================================
090407 12:06:19 INNODB TABLESPACE MONITOR OUTPUT
=================================================
```

The header for the standard InnoDB Monitor (INNODB MONITOR OUTPUT) is also used for the Lock Monitor because the latter produces the same output with the addition of extra lock information.

Enabling an InnoDB monitor for periodic output involves using a CREATE TABLE statement to create a specially named InnoDB table that is associated with the monitor. For example, to enable the standard InnoDB Monitor, you would create an InnoDB table named innodb_monitor.

Using CREATE TABLE syntax is just a way to pass a command to the InnoDB engine through MySQL's SQL parser. The only things that matter are the table name and that it be an InnoDB table. The structure of the table is not relevant. If you shut down the server, the monitor does not restart automatically when you restart the server. Drop the monitor table and issue a new CREATE TABLE statement to start the monitor.

Note

The CREATE TABLE method of enabling InnoDB monitors is removed in MySQL 5.7.4. As of MySQL 5.7.4, use the innodb_status_output and innodb_status_output_locks system variables to enable the standard InnoDB Monitor and InnoDB Lock Monitor.

The PROCESS privilege is required to enable and disable InnoDB Monitors.

Enabling the Standard InnoDB Monitor

Prior to MySQL 5.7.4, enable the standard InnoDB Monitor for periodic output by creating the innodb_monitor table:

```
CREATE TABLE innodb_monitor (a INT) ENGINE=INNODB;
```

To disable the standard InnoDB Monitor, drop the table:

```
DROP TABLE innodb_monitor;
```

As of MySQL 5.7.4, enable the standard InnoDB Monitor by setting the innodb_status_output system variable to ON.

```
set GLOBAL innodb_status_output=ON;
```

To disable the standard InnoDB Monitor, set innodb_status_output to OFF.

When you shut down the server, the innodb_status_output variable is set to the default OFF value.

Obtaining Standard InnoDB Monitor Output On Demand

As an alternative to enabling the standard InnoDB Monitor for periodic output, you can obtain standard InnoDB Monitor output on demand using the SHOW ENGINE INNODB STATUS SQL statement, which fetches the output to your client program. If you are using the mysql interactive client, the output is more readable if you replace the usual semicolon statement terminator with \G:

```
mysql> SHOW ENGINE INNODB STATUS\G
```

Enabling the InnoDB Lock Monitor

Prior to MySQL 5.7.4, enable the InnoDB Lock Monitor for periodic output by creating the innodb_lock_monitor table:

```
CREATE TABLE innodb_lock_monitor (a INT) ENGINE=INNODB;
```

To disable the `InnoDB` Lock Monitor, drop the table:

```
DROP TABLE innodb_lock_monitor;
```

As of MySQL 5.7.4, enable the `InnoDB` Lock Monitor by setting both the `innodb_status_output` and `innodb_status_output_locks` system variables to `ON`. Because Lock Monitor output is printed with the standard `InnoDB` Monitor output, both monitors must be enabled to enable Lock Monitor output.

```
set GLOBAL innodb_status_output=ON;
set GLOBAL innodb_status_output_locks=ON;
```

When you shut down the server, the `innodb_status_output` and `innodb_status_output_locks` variables are set to the default `OFF` value.

To disable the `InnoDB` Lock Monitor, set `innodb_status_output_locks` to `OFF`. Set set `innodb_status_output` to OFF to also disable the standard `InnoDB` Monitor.

Enabling the `InnoDB` Tablespace Monitor

To enable the `InnoDB` Tablespace Monitor for periodic output, create the `innodb_tablespace_monitor` table:

```
CREATE TABLE innodb_tablespace_monitor (a INT) ENGINE=INNODB;
```

To disable the standard `InnoDB` Tablespace Monitor, drop the table:

```
DROP TABLE innodb_tablespace_monitor;
```

> **Note**
>
> The Tablespace Monitor is removed in MySQL 5.7.4. Equivalent functionality will be introduced before the GA release of MySQL 5.7.

Enabling the `InnoDB` Table Monitor

To enable the `InnoDB` Table Monitor for periodic output, create the `innodb_table_monitor` table:

```
CREATE TABLE innodb_table_monitor (a INT) ENGINE=INNODB;
```

To disable the `InnoDB` Table Monitor, drop the table:

```
DROP TABLE innodb_table_monitor;
```

> **Note**
>
> The Tablespace Monitor is removed in MySQL 5.7.4. Equivalent functionality will be introduced before the GA release of MySQL 5.7.

`InnoDB` Standard Monitor and Lock Monitor Output

The Lock Monitor is the same as the standard Monitor except that it includes additional lock information. Enabling either monitor for periodic output turns on the same output stream, but the stream includes extra information if the Lock Monitor is enabled. For example, if you enable the standard `InnoDB` Monitor and `InnoDB` Lock Monitor, that turns on a single output stream. The stream includes extra lock information until you disable the Lock Monitor.

Example standard InnoDB Monitor output:

```
mysql> SHOW ENGINE INNODB STATUS\G
*************************** 1. row ***************************
Status:
=====================================
030709 13:00:59 INNODB MONITOR OUTPUT
=====================================
Per second averages calculated from the last 18 seconds
----------
BACKGROUND THREAD
----------
srv_master_thread loops: 53 1_second, 44 sleeps, 5 10_second, 7 background,
  7 flush
srv_master_thread log flush and writes: 48
----------
SEMAPHORES
----------
OS WAIT ARRAY INFO: reservation count 413452, signal count 378357
--Thread 32782 has waited at btr0sea.c line 1477 for 0.00 seconds the
semaphore: X-lock on RW-latch at 41a28668 created in file btr0sea.c line 135
a writer (thread id 32782) has reserved it in mode wait exclusive
number of readers 1, waiters flag 1
Last time read locked in file btr0sea.c line 731
Last time write locked in file btr0sea.c line 1347
Mutex spin waits 0, rounds 0, OS waits 0
RW-shared spins 2, rounds 60, OS waits 2
RW-excl spins 0, rounds 0, OS waits 0
Spin rounds per wait: 0.00 mutex, 20.00 RW-shared, 0.00 RW-excl
------------------------
LATEST FOREIGN KEY ERROR
------------------------
030709 13:00:59 Transaction:
TRANSACTION 0 290328284, ACTIVE 0 sec, process no 3195
inserting
15 lock struct(s), heap size 2496, undo log entries 9
MySQL thread id 25, query id 4668733 localhost heikki update
insert into ibtest11a (D, B, C) values (5, 'khDk' ,'khDk')
Foreign key constraint fails for table test/ibtest11a:
,
  CONSTRAINT `0_219242` FOREIGN KEY (`A`, `D`) REFERENCES `ibtest11b` (`A`,
  `D`) ON DELETE CASCADE ON UPDATE CASCADE
Trying to add in child table, in index PRIMARY tuple:
 0: len 4; hex 80000101; asc ....;; 1: len 4; hex 80000005; asc ....;; 2:
 len 4; hex 6b68446b; asc khDk;; 3: len 6; hex 0000114e0edc; asc ...N..;; 4:
 len 7; hex 00000000c3e0a7; asc .......;; 5: len 4; hex 6b68446b; asc khDk;;
But in parent table test/ibtest11b, in index PRIMARY,
the closest match we can find is record:
RECORD: info bits 0 0: len 4; hex 8000015b; asc ...[;; 1: len 4; hex
80000005; asc ....;; 2: len 3; hex 6b6864; asc khd;; 3: len 6; hex
0000111ef3eb; asc ......;; 4: len 7; hex 800001001e0084; asc .......;; 5:
len 3; hex 6b6864; asc khd;;
------------------------
LATEST DETECTED DEADLOCK
------------------------
030709 12:59:58
*** (1) TRANSACTION:
TRANSACTION 0 290252780, ACTIVE 1 sec, process no 3185
inserting
LOCK WAIT 3 lock struct(s), heap size 320, undo log entries 146
MySQL thread id 21, query id 4553379 localhost heikki update
INSERT INTO alex1 VALUES(86, 86, 794,'aA35818','bb','c79166','d4766t',
'e187358f','g84586','h794',date_format('2001-04-03 12:54:22','%Y-%m-%d
%H:%i'),7
*** (1) WAITING FOR THIS LOCK TO BE GRANTED:
RECORD LOCKS space id 0 page no 48310 n bits 568 table test/alex1 index
```

```
symbole trx id 0 290252780 lock mode S waiting
Record lock, heap no 324 RECORD: info bits 0 0: len 7; hex 61613335383138;
asc aa35818;; 1:
*** (2) TRANSACTION:
TRANSACTION 0 290251546, ACTIVE 2 sec, process no 3190
inserting
130 lock struct(s), heap size 11584, undo log entries 437
MySQL thread id 23, query id 4554396 localhost heikki update
REPLACE INTO alex1 VALUES(NULL, 32, NULL,'aa3572','','c3572','d6012t','',
NULL,'h396', NULL, NULL, 7.31,7.31,7.31,200)
*** (2) HOLDS THE LOCK(S):
RECORD LOCKS space id 0 page no 48310 n bits 568 table test/alex1 index
symbole trx id 0 290251546 lock_mode X locks rec but not gap
Record lock, heap no 324 RECORD: info bits 0 0: len 7; hex 61613335383138;
asc aa35818;; 1:
*** (2) WAITING FOR THIS LOCK TO BE GRANTED:
RECORD LOCKS space id 0 page no 48310 n bits 568 table test/alex1 index
symbole trx id 0 290251546 lock_mode X locks gap before rec insert intention
waiting
Record lock, heap no 82 RECORD: info bits 0 0: len 7; hex 61613335373230;
asc aa35720;; 1:
*** WE ROLL BACK TRANSACTION (1)
------------
TRANSACTIONS
------------
Trx id counter 0 290328385
Purge done for trx's n:o < 0 290315608 undo n:o < 0 17
History list length 20
Total number of lock structs in row lock hash table 70
LIST OF TRANSACTIONS FOR EACH SESSION:
---TRANSACTION 0 0, not started, process no 3491
MySQL thread id 32, query id 4668737 localhost heikki
show innodb status
---TRANSACTION 0 290328384, ACTIVE 0 sec, process no 3205
38929 inserting
1 lock struct(s), heap size 320
MySQL thread id 29, query id 4668736 localhost heikki update
insert into speedc values (1519229,1, 'hgjhjgghggjgjgjgjggjgjgjgjgggjgjg
jlhhgghggggghhjhghgggggghjhghghghghhhhghghghjhhjghjghjkghjghjghjghjfhjfh
---TRANSACTION 0 290328383, ACTIVE 0 sec, process no 3180
28684 committing
1 lock struct(s), heap size 320, undo log entries 1
MySQL thread id 19, query id 4668734 localhost heikki update
insert into speedcm values (1603393,1, 'hgjhjgghggjgjgjgjggjgjgjgjgggjgj
gjlhhgghggggghhjhghgggggghjhghghghghhhhghghghjhhjghjghjkghjghjghjghjfhjf
---TRANSACTION 0 290328327, ACTIVE 0 sec, process no 3200
36880 starting index read
LOCK WAIT 2 lock struct(s), heap size 320
MySQL thread id 27, query id 4668644 localhost heikki Searching rows for
update
update ibtest11a set B = 'kHdkkkk' where A = 89572
------- TRX HAS BEEN WAITING 0 SEC FOR THIS LOCK TO BE GRANTED:
RECORD LOCKS space id 0 page no 65556 n bits 232 table test/ibtest11a index
PRIMARY trx id 0 290328327 lock_mode X waiting
Record lock, heap no 1 RECORD: info bits 0 0: len 9; hex 73757072656d756d00;
asc supremum.;;
------------------
---TRANSACTION 0 290328284, ACTIVE 0 sec, process no 3195
34831 rollback of SQL statement
ROLLING BACK 14 lock struct(s), heap size 2496, undo log entries 9
MySQL thread id 25, query id 4668733 localhost heikki update
insert into ibtest11a (D, B, C) values (5, 'khDk' ,'khDk')
---TRANSACTION 0 290327208, ACTIVE 1 sec, process no 3190
32782
58 lock struct(s), heap size 5504, undo log entries 159
MySQL thread id 23, query id 4668732 localhost heikki update
REPLACE INTO alex1 VALUES(86, 46, 538,'aa95666','bb','c95666','d9486t',
'e200498f','g86814','h538',date_format('2001-04-03 12:54:22','%Y-%m-%d
%H:%i'),
---TRANSACTION 0 290323325, ACTIVE 3 sec, process no 3185
30733 inserting
4 lock struct(s), heap size 1024, undo log entries 165
MySQL thread id 21, query id 4668735 localhost heikki update
INSERT INTO alex1 VALUES(NULL, 49, NULL,'aa42837','','c56319','d1719t','',
NULL,'h321', NULL, NULL, 7.31,7.31,7.31,200)
--------
FILE I/O
--------
I/O thread 0 state: waiting for i/o request (insert buffer thread)
I/O thread 1 state: waiting for i/o request (log thread)
I/O thread 2 state: waiting for i/o request (read thread)
I/O thread 3 state: waiting for i/o request (write thread)
Pending normal aio reads: 0, aio writes: 0,
 ibuf aio reads: 0, log i/o's: 0, sync i/o's: 0
Pending flushes (fsync) log: 0; buffer pool: 0
151671 OS file reads, 94747 OS file writes, 8750 OS fsyncs
25.44 reads/s, 18494 avg bytes/read, 17.55 writes/s, 2.33 fsyncs/s
-------------------------------------
INSERT BUFFER AND ADAPTIVE HASH INDEX
-------------------------------------
Ibuf for space 0: size 1, free list len 19, seg size 21,
85004 inserts, 85004 merged recs, 26669 merges
Hash table size 207619, used cells 14461, node heap has 16 buffer(s)
1877.67 hash searches/s, 5121.10 non-hash searches/s
---
LOG
---
Log sequence number 18 1212842764
Log flushed up to   18 1212665295
Last checkpoint at  18 1135877290
0 pending log writes, 0 pending chkp writes
4341 log i/o's done, 1.22 log i/o's/second
----------------------
BUFFER POOL AND MEMORY
----------------------
Total memory allocated 84966343; in additional pool allocated 1402624
Buffer pool size   3200
Free buffers       110
Database pages     3074
Modified db pages  2674
Pending reads 0
Pending writes: LRU 0, flush list 0, single page 0
Pages read 171380, created 51968, written 194688
28.72 reads/s, 20.72 creates/s, 47.55 writes/s
Buffer pool hit rate 999 / 1000
--------------
ROW OPERATIONS
--------------
0 queries inside InnoDB, 0 queries in queue
Main thread process no. 3004, id 7176, state: purging
Number of rows inserted 3738558, updated 127415, deleted 33707, read 755779
1586.13 inserts/s, 50.89 updates/s, 28.44 deletes/s, 107.88 reads/s
----------------------------
END OF INNODB MONITOR OUTPUT
============================
```

Standard InnoDB Monitor output is limited to 1MB when produced using the SHOW ENGINE INNODB STATUS statement. This limit does not apply to output written to the server's error output.

Some notes on the output sections:

Status

This section shows the timestamp, the monitor name, and the number of seconds that per-second averages are based on. The number of seconds is the elapsed time between the current time and the last time InnoDB Monitor output was printed.

BACKGROUND THREAD

The srv_master_thread lines shows work done by the main background thread.

SEMAPHORES

This section reports threads waiting for a semaphore and statistics on how many times threads have needed a spin or a wait on a mutex or a rw-lock semaphore. A large number of threads waiting for semaphores may be a result of disk I/O, or contention problems inside InnoDB. Contention can be due to heavy parallelism of queries or problems in operating system thread scheduling. Setting the innodb_thread_concurrency system variable smaller than the default value might help in such situations. The Spin rounds per wait line shows the number of spinlock rounds per OS wait for a mutex.

LATEST FOREIGN KEY ERROR

This section provides information about the most recent foreign key constraint error. It is not present if no such error has occurred. The contents include the statement that failed as well as information about the constraint that failed and the referenced and referencing tables.

LATEST DETECTED DEADLOCK

This section provides information about the most recent deadlock. It is not present if no deadlock has occurred. The contents show which transactions are involved, the statement each was attempting to execute, the locks they have and need, and which transaction InnoDB decided to roll back to break the deadlock. The lock modes reported in this section are explained in Section 14.2.2.3, “InnoDB Lock Modes”.

TRANSACTIONS

If this section reports lock waits, your applications might have lock contention. The output can also help to trace the reasons for transaction deadlocks.

FILE I/O

This section provides information about threads that InnoDB uses to perform various types of I/O. The first few of these are dedicated to general InnoDB processing. The contents also display information for pending I/O operations and statistics for I/O performance.

The number of these threads are controlled by the innodb_read_io_threads and innodb_write_io_threads parameters. See Section 14.2.13, “InnoDB Startup Options and System Variables”.

INSERT BUFFER AND ADAPTIVE HASH INDEX

This section shows the status of the InnoDB insert buffer and adaptive hash index. (See Insert Buffering, and Adaptive Hash Indexes.) The contents include the number of operations performed for each, plus statistics for hash index performance.

LOG

This section displays information about the InnoDB log. The contents include the current log sequence number, how far the log has been flushed to disk, and the position at which InnoDB last took a checkpoint.

(See Section 14.2.10.3, “InnoDB Checkpoints”.) The section also displays information about pending writes and write performance statistics.

BUFFER POOL AND MEMORY

This section gives you statistics on pages read and written. You can calculate from these numbers how many data file I/O operations your queries currently are doing.

For additional information about the operation of the buffer pool, see Section 8.9.1, “The InnoDB Buffer Pool”.

ROW OPERATIONS

This section shows what the main thread is doing, including the number and performance rate for each type of row operation.

InnoDB Tablespace Monitor Output

Note

The InnoDB Tablespace Monitor is removed in MySQL 5.7.4. Equivalent functionality will be introduced before the GA release of MySQL 5.7.

The InnoDB Tablespace Monitor prints information about the file segments in the shared tablespace and validates the tablespace allocation data structures. The Tablespace Monitor does not describe file-per-table tablespaces created with the innodb_file_per_table option.

Example InnoDB Tablespace Monitor output:

```
================================================
090408 21:28:09 INNODB TABLESPACE MONITOR OUTPUT
================================================
FILE SPACE INFO: id 0
size 13440, free limit 3136, free extents 28
not full frag extents 2: used pages 78, full frag extents 3
first seg id not used 0 23845
SEGMENT id 0 1 space 0; page 2; res 96 used 46; full ext 0
fragm pages 32; free extents 0; not full extents 1: pages 14
SEGMENT id 0 2 space 0; page 2; res 1 used 1; full ext 0
fragm pages 1; free extents 0; not full extents 0: pages 0
SEGMENT id 0 3 space 0; page 2; res 1 used 1; full ext 0
fragm pages 1; free extents 0; not full extents 0: pages 0
...
SEGMENT id 0 15 space 0; page 2; res 160 used 160; full ext 2
fragm pages 32; free extents 0; not full extents 0: pages 0
SEGMENT id 0 488 space 0; page 2; res 1 used 1; full ext 0
fragm pages 1; free extents 0; not full extents 0: pages 0
SEGMENT id 0 17 space 0; page 2; res 1 used 1; full ext 0
fragm pages 1; free extents 0; not full extents 0: pages 0
...
SEGMENT id 0 171 space 0; page 2; res 592 used 481; full ext 7
fragm pages 16; free extents 0; not full extents 2: pages 17
SEGMENT id 0 172 space 0; page 2; res 1 used 1; full ext 0
fragm pages 1; free extents 0; not full extents 0: pages 0
SEGMENT id 0 173 space 0; page 2; res 96 used 44; full ext 0
fragm pages 32; free extents 0; not full extents 1: pages 12
...
SEGMENT id 0 601 space 0; page 2; res 1 used 1; full ext 0
fragm pages 1; free extents 0; not full extents 0: pages 0
NUMBER of file segments: 73
Validating tablespace
```

```
Validation ok
---------------------------------------
END OF INNODB TABLESPACE MONITOR OUTPUT
=======================================
```

The Tablespace Monitor output includes information about the shared tablespace as a whole, followed by a list containing a breakdown for each segment within the tablespace.

In this example using the default page size, the tablespace consists of database pages that are 16KB each. The pages are grouped into extents of size 1MB (64 consecutive pages).

The initial part of the output that displays overall tablespace information has this format:

```
FILE SPACE INFO: id 0
size 13440, free limit 3136, free extents 28
not full frag extents 2: used pages 78, full frag extents 3
first seg id not used 0 23845
```

Overall tablespace information includes these values:

- `id`: The tablespace ID. A value of 0 refers to the shared tablespace.
- `size`: The current tablespace size in pages.
- `free limit`: The minimum page number for which the free list has not been initialized. Pages at or above this limit are free.
- `free extents`: The number of free extents.
- `not full frag extents`, `used pages`: The number of fragment extents that are not completely filled, and the number of pages in those extents that have been allocated.
- `full frag extents`: The number of completely full fragment extents.
- `first seg id not used`: The first unused segment ID.

Individual segment information has this format:

```
SEGMENT id 0 15 space 0; page 2; res 160 used 160; full ext 2
fragm pages 32; free extents 0; not full extents 0: pages 0
```

Segment information includes these values:

`id`: The segment ID.

`space`, `page`: The tablespace number and page within the tablespace where the segment “inode” is located. A tablespace number of 0 indicates the shared tablespace. `InnoDB` uses inodes to keep track of segments in the tablespace. The other fields displayed for a segment (`id`, `res`, and so forth) are derived from information in the inode.

`res`: The number of pages allocated (reserved) for the segment.

`used`: The number of allocated pages in use by the segment.

`full ext`: The number of extents allocated for the segment that are completely used.

`fragm pages`: The number of initial pages that have been allocated to the segment.

free extents: The number of extents allocated for the segment that are completely unused.

not full extents: The number of extents allocated for the segment that are partially used.

pages: The number of pages used within the not-full extents.

When a segment grows, it starts as a single page, and InnoDB allocates the first pages for it one at a time, up to 32 pages (this is the fragm pages value). After that, InnoDB allocates complete extents. InnoDB can add up to 4 extents at a time to a large segment to ensure good sequentiality of data.

For the example segment shown earlier, it has 32 fragment pages, plus 2 full extents (64 pages each), for a total of 160 pages used out of 160 pages allocated. The following segment has 32 fragment pages and one partially full extent using 14 pages for a total of 46 pages used out of 96 pages allocated:

```
SEGMENT id 0 1 space 0; page 2; res 96 used 46; full ext 0
fragm pages 32; free extents 0; not full extents 1: pages 14
```

It is possible for a segment that has extents allocated to it to have a fragm pages value less than 32 if some of the individual pages have been deallocated subsequent to extent allocation.

InnoDB Table Monitor Output

Note

The InnoDB Table Monitor is removed in MySQL 5.7.4. Equivalent information can be obtained from InnoDB INFORMATION_SCHEMA tables. See Section 19.30, “INFORMATION_SCHEMA Tables for InnoDB”.

The InnoDB Table Monitor prints the contents of the InnoDB internal data dictionary.

The output contains one section per table. The SYS_FOREIGN and SYS_FOREIGN_COLS sections are for internal data dictionary tables that maintain information about foreign keys. There are also sections for the Table Monitor table and each user-created InnoDB table. Suppose that the following two tables have been created in the test database:

```
CREATE TABLE parent
(
  par_id    INT NOT NULL,
  fname      CHAR(20),
  lname      CHAR(20),
  PRIMARY KEY (par_id),
  UNIQUE INDEX (lname, fname)
) ENGINE = INNODB;

CREATE TABLE child
(
  par_id      INT NOT NULL,
  child_id    INT NOT NULL,
  name        VARCHAR(40),
  birth       DATE,
  weight      DECIMAL(10,2),
  misc_info   VARCHAR(255),
  last_update TIMESTAMP,
  PRIMARY KEY (par_id, child_id),
  INDEX (name),
  FOREIGN KEY (par_id) REFERENCES parent (par_id)
    ON DELETE CASCADE
    ON UPDATE CASCADE
) ENGINE = INNODB;
```

Then the Table Monitor output will look something like this (reformatted slightly):

```
===========================================
090420 12:09:32 INNODB TABLE MONITOR OUTPUT
===========================================
--------------------------------------
TABLE: name SYS_FOREIGN, id 0 11, columns 7, indexes 3, appr.rows 1
  COLUMNS: ID: DATA_VARCHAR DATA_ENGLISH len 0;
           FOR_NAME: DATA_VARCHAR DATA_ENGLISH len 0;
           REF_NAME: DATA_VARCHAR DATA_ENGLISH len 0;
           N_COLS: DATA_INT len 4;
           DB_ROW_ID: DATA_SYS prtype 256 len 6;
           DB_TRX_ID: DATA_SYS prtype 257 len 6;
  INDEX: name ID_IND, id 0 11, fields 1/6, uniq 1, type 3
   root page 46, appr.key vals 1, leaf pages 1, size pages 1
   FIELDS:  ID DB_TRX_ID DB_ROLL_PTR FOR_NAME REF_NAME N_COLS
  INDEX: name FOR_IND, id 0 12, fields 1/2, uniq 2, type 0
   root page 47, appr.key vals 1, leaf pages 1, size pages 1
   FIELDS:  FOR_NAME ID
  INDEX: name REF_IND, id 0 13, fields 1/2, uniq 2, type 0
   root page 48, appr.key vals 1, leaf pages 1, size pages 1
   FIELDS:  REF_NAME ID
--------------------------------------
TABLE: name SYS_FOREIGN_COLS, id 0 12, columns 7, indexes 1, appr.rows 1
  COLUMNS: ID: DATA_VARCHAR DATA_ENGLISH len 0;
           POS: DATA_INT len 4;
           FOR_COL_NAME: DATA_VARCHAR DATA_ENGLISH len 0;
           REF_COL_NAME: DATA_VARCHAR DATA_ENGLISH len 0;
           DB_ROW_ID: DATA_SYS prtype 256 len 6;
           DB_TRX_ID: DATA_SYS prtype 257 len 6;
  INDEX: name ID_IND, id 0 14, fields 2/6, uniq 2, type 3
   root page 49, appr.key vals 1, leaf pages 1, size pages 1
   FIELDS:  ID POS DB_TRX_ID DB_ROLL_PTR FOR_COL_NAME REF_COL_NAME
--------------------------------------
TABLE: name test/child, id 0 14, columns 10, indexes 2, appr.rows 201
  COLUMNS: par_id: DATA_INT DATA_BINARY_TYPE DATA_NOT_NULL len 4;
           child_id: DATA_INT DATA_BINARY_TYPE DATA_NOT_NULL len 4;
           name: DATA_VARCHAR prtype 524303 len 40;
           birth: DATA_INT DATA_BINARY_TYPE len 3;
           weight: DATA_FIXBINARY DATA_BINARY_TYPE len 5;
           misc_info: DATA_VARCHAR prtype 524303 len 255;
           last_update: DATA_INT DATA_UNSIGNED DATA_BINARY_TYPE DATA_NOT_NULL len 4;
           DB_ROW_ID: DATA_SYS prtype 256 len 6;
           DB_TRX_ID: DATA_SYS prtype 257 len 6;
  INDEX: name PRIMARY, id 0 17, fields 2/9, uniq 2, type 3
   root page 52, appr.key vals 201, leaf pages 5, size pages 6
   FIELDS:  par_id child_id DB_TRX_ID DB_ROLL_PTR name birth weight misc_info last_update
  INDEX: name name, id 0 18, fields 1/3, uniq 3, type 0
   root page 53, appr.key vals 210, leaf pages 1, size pages 1
   FIELDS:  name par_id child_id
  FOREIGN KEY CONSTRAINT test/child_ibfk_1: test/child ( par_id )
             REFERENCES test/parent ( par_id )
--------------------------------------
TABLE: name test/innodb_table_monitor, id 0 15, columns 4, indexes 1, appr.rows 0
  COLUMNS: i: DATA_INT DATA_BINARY_TYPE len 4;
           DB_ROW_ID: DATA_SYS prtype 256 len 6;
           DB_TRX_ID: DATA_SYS prtype 257 len 6;
  INDEX: name GEN_CLUST_INDEX, id 0 19, fields 0/4, uniq 1, type 1
   root page 193, appr.key vals 0, leaf pages 1, size pages 1
   FIELDS:  DB_ROW_ID DB_TRX_ID DB_ROLL_PTR i
--------------------------------------
TABLE: name test/parent, id 0 13, columns 6, indexes 2, appr.rows 299
  COLUMNS: par_id: DATA_INT DATA_BINARY_TYPE DATA_NOT_NULL len 4;
           fname: DATA_CHAR prtype 524542 len 20;
           lname: DATA_CHAR prtype 524542 len 20;
           DB_ROW_ID: DATA_SYS prtype 256 len 6;
           DB_TRX_ID: DATA_SYS prtype 257 len 6;
  INDEX: name PRIMARY, id 0 15, fields 1/5, uniq 1, type 3
   root page 50, appr.key vals 299, leaf pages 2, size pages 3
   FIELDS:  par_id DB_TRX_ID DB_ROLL_PTR fname lname
  INDEX: name lname, id 0 16, fields 2/3, uniq 2, type 2
   root page 51, appr.key vals 300, leaf pages 1, size pages 1
   FIELDS:  lname fname par_id
  FOREIGN KEY CONSTRAINT test/child_ibfk_1: test/child ( par_id )
             REFERENCES test/parent ( par_id )
-----------------------------------
END OF INNODB TABLE MONITOR OUTPUT
==================================
```

For each table, Table Monitor output contains a section that displays general information about the table and specific information about its columns, indexes, and foreign keys.

The general information for each table includes the table name (in *db_name/tbl_name* format except for internal tables), its ID, the number of columns and indexes, and an approximate row count.

The COLUMNS part of a table section lists each column in the table. Information for each column indicates its name and data type characteristics. Some internal columns are added by InnoDB, such as DB_ROW_ID (row ID), DB_TRX_ID (transaction ID), and DB_ROLL_PTR (a pointer to the rollback/undo data).

- DATA_*xxx*: These symbols indicate the data type. There may be multiple DATA_*xxx* symbols for a given column.

- prtype: The column's “precise” type. This field includes information such as the column data type, character set code, nullability, signedness, and whether it is a binary string. This field is described in the innobase/include/data0type.h source file.

- len: The column length in bytes.

Each INDEX part of the table section provides the name and characteristics of one table index:

- name: The index name. If the name is PRIMARY, the index is a primary key. If the name is GEN_CLUST_INDEX, the index is the clustered index that is created automatically if the table definition doesn't include a primary key or non-NULL unique index. See Clustered and Secondary Indexes.

- id: The index ID.

- fields: The number of fields in the index, as a value in *m/n* format:

  - *m* is the number of user-defined columns; that is, the number of columns you would see in the index definition in a CREATE TABLE statement.

  - *n* is the total number of index columns, including those added internally. For the clustered index, the total includes the other columns in the table definition, plus any columns added internally. For a secondary index, the total includes the columns from the primary key that are not part of the secondary index.

- uniq: The number of leading fields that are enough to determine index values uniquely.

- type: The index type. This is a bit field. For example, 1 indicates a clustered index and 2 indicates a unique index, so a clustered index (which always contains unique values), will have a type value of 3. An index with a type value of 0 is neither clustered nor unique. The flag values are defined in the innobase/include/dict0mem.h source file.

- root page: The index root page number.

- appr. key vals: The approximate index cardinality.
- leaf pages: The approximate number of leaf pages in the index.
- size pages: The approximate total number of pages in the index.
- FIELDS: The names of the fields in the index. For a clustered index that was generated automatically, the field list begins with the internal DB_ROW_ID (row ID) field. DB_TRX_ID and DB_ROLL_PTR are always added internally to the clustered index, following the fields that comprise the primary key. For a secondary index, the final fields are those from the primary key that are not part of the secondary index.

The end of the table section lists the FOREIGN KEY definitions that apply to the table. This information appears whether the table is a referencing or referenced table.

14.2.12.5 Controlling Optimizer Statistics Estimation

The MySQL query optimizer uses estimated statistics about key distributions to choose the indexes for an execution plan, based on the relative selectivity of the index. Certain operations cause InnoDB to sample random pages from each index on a table to estimate the cardinality of the index. (This technique is known as random dives.) These operations include the ANALYZE TABLE statement, the SHOW TABLE STATUS statement, and accessing the table for the first time after a restart.

To give you control over the quality of the statistics estimate (and thus better information for the query optimizer), you can now change the number of sampled pages using the parameter innodb_stats_transient_sample_pages. Previously, the number of sampled pages was always 8, which could be insufficient to produce an accurate estimate, leading to poor index choices by the query optimizer. This technique is especially important for large tables and tables used in joins. Unnecessary full table scans for such tables can be a substantial performance issue. See Section 8.2.1.20, “How to Avoid Full Table Scans” for tips on tuning such queries.

You can set the global parameter innodb_stats_transient_sample_pages, at runtime. The default value for this parameter is 8, preserving the same behavior as in past releases.

Note

The value of innodb_stats_transient_sample_pages affects the index sampling for *all* InnoDB tables and indexes. There are the following potentially significant impacts when you change the index sample size:

- Small values like 1 or 2 can result in very inaccurate estimates of cardinality.
- Increasing the innodb_stats_transient_sample_pages value might require more disk reads. Values much larger than 8 (say, 100), can cause a big slowdown in the time it takes to open a table or execute SHOW TABLE STATUS.
- The optimizer might choose very different query plans based on different estimates of index selectivity.

To disable the cardinality estimation for metadata statements such as SHOW TABLE STATUS, execute the statement SET GLOBAL innodb_stats_on_metadata=OFF (or 0). The ability to set this option dynamically is also relatively new.

All InnoDB tables are opened, and the statistics are re-estimated for all associated indexes, when the mysql client starts if the auto-rehash setting is set on (the default). To improve the start up time of the mysql client, you can turn auto-rehash off. The auto-rehash feature enables automatic name completion of database, table, and column names for interactive users.

Whatever value of innodb_stats_transient_sample_pages works best for a system, set the option and leave it at that value. Choose a value that results in reasonably accurate estimates for all tables in your database without requiring excessive I/O. Because the statistics are automatically recalculated at various times other than on execution of ANALYZE TABLE, it does not make sense to increase the index sample size, run ANALYZE TABLE, then decrease sample size again. The more accurate statistics calculated by ANALYZE running with a high value of innodb_stats_transient_sample_pages can be wiped away later.

Although it is not possible to specify the sample size on a per-table basis, smaller tables generally require fewer index samples than larger tables do. If your database has many large tables, consider using a higher value for innodb_stats_transient_sample_pages than if you have mostly smaller tables.

14.2.13 InnoDB Startup Options and System Variables

- System variables that are true or false can be enabled at server startup by naming them, or disabled by using a --skip- prefix. For example, to enable or disable the InnoDB adaptive hash index, you can use --innodb_adaptive_hash_index or --skip-innodb_adaptive_hash_index on the command line, or innodb_adaptive_hash_index or skip-innodb_adaptive_hash_index in an option file.
- System variables that take a numeric value can be specified as --*var_name*=*value* on the command line or as *var_name*=*value* in option files.
- Many system variables can be changed at runtime (see Section 5.1.5.2, “Dynamic System Variables”).
- For information about GLOBAL and SESSION variable scope modifiers, refer to the SET statement documentation.
- Certain options control the locations and layout of the InnoDB data files. Section 14.2.3, “InnoDB Configuration” explains how to use these options.
- Some options, which you might not use initially, help tune InnoDB performance characteristics based on machine capacity and your database workload. The performance-related options are explained in Section 14.2.12, “InnoDB Performance Tuning” and Section 14.2.12.2, “InnoDB Performance and Scalability Enhancements”.
- For more information on specifying options and system variables, see Section 4.2.3, “Specifying Program Options”.

Table 14.10 InnoDB Option/Variable Reference

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
daemon_memcached_enable_binlog	Yes	Yes	Yes		Global	No
daemon_memcached_engine_lib_name	Yes	Yes	Yes		Global	No
daemon_memcached_engine_lib_path	Yes	Yes	Yes		Global	No
daemon_memcached_option	Yes	Yes	Yes		Global	No
daemon_memcached_r_batch_size	Yes	Yes	Yes		Global	No
daemon_memcached_w_batch_size	Yes	Yes	Yes		Global	No
foreign_key_checks			Yes		Both	Yes
ignore-builtin-innodb	Yes	Yes			Global	No
- *Variable*: ignore_builtin_innodb			Yes		Global	No

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
innodb	Yes	Yes				
innodb_adaptive_flushing	Yes	Yes	Yes		Global	Yes
innodb_adaptive_flushing_lwm	Yes	Yes	Yes		Global	Yes
innodb_adaptive_hash_index	Yes	Yes	Yes		Global	Yes
innodb_adaptive_max_sleep_delay	Yes	Yes	Yes		Global	Yes
innodb_additional_mem_pool_size	Yes	Yes	Yes		Global	No
innodb_api_bk_commit_interval	Yes	Yes	Yes		Global	Yes
innodb_api_disable_rowlock	Yes	Yes	Yes		Global	No
innodb_api_enable_binlog	Yes	Yes	Yes		Global	No
innodb_api_enable_mdl	Yes	Yes	Yes		Global	No
innodb_api_trx_level	Yes	Yes	Yes		Global	Yes
innodb_autoextend_increment	Yes	Yes	Yes		Global	Yes
innodb_autoinc_lock_mode	Yes	Yes	Yes		Global	No
Innodb_available_undo_logs				Yes	Global	No
Innodb_buffer_pool_bytes_data				Yes	Global	No
Innodb_buffer_pool_bytes_dirty				Yes	Global	No
innodb_buffer_pool_dump_at_shutdown	Yes	Yes	Yes		Global	Yes
innodb_buffer_pool_dump_now	Yes	Yes	Yes		Global	Yes
innodb_buffer_pool_dump_pct	Yes	Yes	Yes		Global	Yes
Innodb_buffer_pool_dump_status				Yes	Global	No
innodb_buffer_pool_filename	Yes	Yes	Yes		Global	Yes
innodb_buffer_pool_instances	Yes	Yes	Yes		Global	No
innodb_buffer_pool_load_abort	Yes	Yes	Yes		Global	Yes
innodb_buffer_pool_load_at_startup	Yes	Yes	Yes		Global	No
innodb_buffer_pool_load_now	Yes	Yes	Yes		Global	Yes
Innodb_buffer_pool_load_status				Yes	Global	No
Innodb_buffer_pool_pages_data				Yes	Global	No
Innodb_buffer_pool_pages_dirty				Yes	Global	No
Innodb_buffer_pool_pages_flushed				Yes	Global	No
Innodb_buffer_pool_pages_free				Yes	Global	No
Innodb_buffer_pool_pages_latched				Yes	Global	No
Innodb_buffer_pool_pages_misc				Yes	Global	No
Innodb_buffer_pool_pages_total				Yes	Global	No
Innodb_buffer_pool_read_ahead				Yes	Global	No
Innodb_buffer_pool_read_ahead_evicted				Yes	Global	No
Innodb_buffer_pool_read_requests				Yes	Global	No
Innodb_buffer_pool_reads				Yes	Global	No
innodb_buffer_pool_size	Yes	Yes	Yes		Global	No

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
Innodb_buffer_pool_wait_free				Yes	Global	No
Innodb_buffer_pool_write_requests				Yes	Global	No
innodb_change_buffer_max_size	Yes	Yes	Yes		Global	Yes
innodb_change_buffering	Yes	Yes	Yes		Global	Yes
innodb_checksum_algorithm	Yes	Yes	Yes		Global	Yes
innodb_checksums	Yes	Yes	Yes		Global	No
innodb_cmp_per_index_enabled	Yes	Yes	Yes		Global	Yes
innodb_commit_concurrency	Yes	Yes	Yes		Global	Yes
innodb_compression_failure_threshold_pct	Yes	Yes	Yes		Global	Yes
innodb_compression_level	Yes	Yes	Yes		Global	Yes
innodb_compression_pad_pct_max	Yes	Yes	Yes		Global	Yes
innodb_concurrency_tickets	Yes	Yes	Yes		Global	Yes
innodb_data_file_path	Yes	Yes	Yes		Global	No
Innodb_data_fsyncs				Yes	Global	No
innodb_data_home_dir	Yes	Yes	Yes		Global	No
Innodb_data_pending_fsyncs				Yes	Global	No
Innodb_data_pending_reads				Yes	Global	No
Innodb_data_pending_writes				Yes	Global	No
Innodb_data_read				Yes	Global	No
Innodb_data_reads				Yes	Global	No
Innodb_data_writes				Yes	Global	No
Innodb_data_written				Yes	Global	No
Innodb_dblwr_pages_written				Yes	Global	No
Innodb_dblwr_writes				Yes	Global	No
innodb_disable_sort_file_cache	Yes	Yes	Yes		Global	Yes
innodb_doublewrite	Yes	Yes	Yes		Global	No
innodb_fast_shutdown	Yes	Yes	Yes		Global	Yes
innodb_file_format	Yes	Yes	Yes		Global	Yes
innodb_file_format_check	Yes	Yes	Yes		Global	No
innodb_file_format_max	Yes	Yes	Yes		Global	Yes
innodb_file_per_table	Yes	Yes	Yes		Global	Yes
innodb_flush_log_at_timeout			Yes		Global	Yes
innodb_flush_log_at_trx_commit	Yes	Yes	Yes		Global	Yes
innodb_flush_method	Yes	Yes	Yes		Global	No
innodb_flush_neighbors	Yes	Yes	Yes		Global	Yes
innodb_flushing_avg_loops	Yes	Yes	Yes		Global	Yes
innodb_force_load_corrupted	Yes	Yes	Yes		Global	No
innodb_force_recovery	Yes	Yes	Yes		Global	No

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
innodb_ft_aux_table	Yes	Yes	Yes		Global	Yes
innodb_ft_cache_size	Yes	Yes	Yes		Global	No
innodb_ft_enable_diag_print	Yes	Yes	Yes		Global	Yes
innodb_ft_enable_stopword	Yes	Yes	Yes		Global	Yes
innodb_ft_max_token_size	Yes	Yes	Yes		Global	No
innodb_ft_min_token_size	Yes	Yes	Yes		Global	No
innodb_ft_num_word_optimize	Yes	Yes	Yes		Global	Yes
innodb_ft_result_cache_limit	Yes	Yes	Yes		Global	Yes
innodb_ft_server_stopword_table	Yes	Yes	Yes		Global	Yes
innodb_ft_sort_pll_degree	Yes	Yes	Yes		Global	No
innodb_ft_total_cache_size	Yes	Yes	Yes		Global	No
innodb_ft_user_stopword_table	Yes	Yes	Yes		Both	Yes
Innodb_have_atomic_builtins				Yes	Global	No
innodb_io_capacity	Yes	Yes	Yes		Global	Yes
innodb_io_capacity_max	Yes	Yes	Yes		Global	Yes
innodb_large_prefix	Yes	Yes	Yes		Global	Yes
innodb_lock_wait_timeout	Yes	Yes	Yes		Both	Yes
innodb_locks_unsafe_for_binlog	Yes	Yes	Yes		Global	No
innodb_log_buffer_size	Yes	Yes	Yes		Global	No
innodb_log_compressed_pages	Yes	Yes	Yes		Global	Yes
innodb_log_file_size	Yes	Yes	Yes		Global	No
innodb_log_files_in_group	Yes	Yes	Yes		Global	No
innodb_log_group_home_dir	Yes	Yes	Yes		Global	No
Innodb_log_waits				Yes	Global	No
innodb_log_write_ahead_size	Yes	Yes	Yes		Global	Yes
Innodb_log_write_requests				Yes	Global	No
Innodb_log_writes				Yes	Global	No
innodb_lru_scan_depth	Yes	Yes	Yes		Global	Yes
innodb_max_dirty_pages_pct	Yes	Yes	Yes		Global	Yes
innodb_max_dirty_pages_pct_lwm	Yes	Yes	Yes		Global	Yes
innodb_max_purge_lag	Yes	Yes	Yes		Global	Yes
innodb_max_purge_lag_delay	Yes	Yes	Yes		Global	Yes
innodb_monitor_disable	Yes	Yes	Yes		Global	Yes
innodb_monitor_enable	Yes	Yes	Yes		Global	Yes
innodb_monitor_reset	Yes	Yes	Yes		Global	Yes
innodb_monitor_reset_all	Yes	Yes	Yes		Global	Yes
Innodb_num_open_files				Yes	Global	No
innodb_old_blocks_pct	Yes	Yes	Yes		Global	Yes

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
innodb_old_blocks_time	Yes	Yes	Yes		Global	Yes
innodb_online_alter_log_max_size	Yes	Yes	Yes		Global	Yes
innodb_open_files	Yes	Yes	Yes		Global	No
innodb_optimize_fulltext_only	Yes	Yes	Yes		Global	Yes
Innodb_os_log_fsyncs				Yes	Global	No
Innodb_os_log_pending_fsyncs				Yes	Global	No
Innodb_os_log_pending_writes				Yes	Global	No
Innodb_os_log_written				Yes	Global	No
innodb_page_cleaners	Yes	Yes	Yes		Global	No
innodb_page_size	Yes	Yes	Yes		Global	No
Innodb_page_size				Yes	Global	No
Innodb_pages_created				Yes	Global	No
Innodb_pages_read				Yes	Global	No
Innodb_pages_written				Yes	Global	No
innodb_print_all_deadlocks	Yes	Yes	Yes		Global	Yes
innodb_purge_batch_size	Yes	Yes	Yes		Global	Yes
innodb_purge_threads	Yes	Yes	Yes		Global	No
innodb_random_read_ahead	Yes	Yes	Yes		Global	Yes
innodb_read_ahead_threshold	Yes	Yes	Yes		Global	Yes
innodb_read_io_threads	Yes	Yes	Yes		Global	No
innodb_read_only	Yes	Yes	Yes		Global	No
innodb_replication_delay	Yes	Yes	Yes		Global	Yes
innodb_rollback_on_timeout	Yes	Yes	Yes		Global	No
innodb_rollback_segments	Yes	Yes	Yes		Global	Yes
Innodb_row_lock_current_waits				Yes	Global	No
Innodb_row_lock_time				Yes	Global	No
Innodb_row_lock_time_avg				Yes	Global	No
Innodb_row_lock_time_max				Yes	Global	No
Innodb_row_lock_waits				Yes	Global	No
Innodb_rows_deleted				Yes	Global	No
Innodb_rows_inserted				Yes	Global	No
Innodb_rows_read				Yes	Global	No
Innodb_rows_updated				Yes	Global	No
innodb_sort_buffer_size	Yes	Yes	Yes		Global	No
innodb_spin_wait_delay	Yes	Yes	Yes		Global	Yes
innodb_stats_auto_recalc	Yes	Yes	Yes		Global	Yes
innodb_stats_method	Yes	Yes	Yes		Global	Yes
innodb_stats_on_metadata	Yes	Yes	Yes		Global	Yes

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
innodb_stats_persistent	Yes	Yes	Yes		Global	Yes
innodb_stats_persistent_sample_pages	Yes	Yes	Yes		Global	Yes
innodb_stats_sample_pages	Yes	Yes	Yes		Global	Yes
innodb_stats_transient_sample_pages	Yes	Yes	Yes		Global	Yes
innodb-status-file	Yes	Yes				
innodb_status_output	Yes	Yes	Yes		Global	Yes
innodb_status_output_locks	Yes	Yes	Yes		Global	Yes
innodb_strict_mode	Yes	Yes	Yes		Both	Yes
innodb_support_xa	Yes	Yes	Yes		Both	Yes
innodb_sync_array_size	Yes	Yes	Yes		Global	No
innodb_sync_spin_loops	Yes	Yes	Yes		Global	Yes
innodb_table_locks	Yes	Yes	Yes		Both	Yes
innodb_temp_data_file_path	Yes	Yes	Yes		Global	No
innodb_thread_concurrency	Yes	Yes	Yes		Global	Yes
innodb_thread_sleep_delay	Yes	Yes	Yes		Global	Yes
Innodb_truncated_status_writes				Yes	Global	No
innodb_undo_directory	Yes	Yes	Yes		Global	No
innodb_undo_logs	Yes	Yes	Yes		Global	Yes
innodb_undo_tablespaces	Yes	Yes	Yes		Global	No
innodb_use_native_aio	Yes	Yes	Yes		Global	No
innodb_use_sys_malloc	Yes	Yes	Yes		Global	No
innodb_version			Yes		Global	No
innodb_write_io_threads	Yes	Yes	Yes		Global	No
timed_mutexes	Yes	Yes	Yes		Global	Yes
unique_checks			Yes		Both	Yes

InnoDB Command Options

- `--ignore-builtin-innodb`

Deprecated	5.5.22	
Command-Line Format	`--ignore-builtin-innodb`	
Option-File Format	`ignore-builtin-innodb`	
System Variable Name	`ignore_builtin_innodb`	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	`boolean`

In MySQL 5.1, this option caused the server to behave as if the built-in InnoDB were not present, which enabled InnoDB Plugin to be used instead. In MySQL 5.7, InnoDB is the default storage engine and InnoDB Plugin is not used, so this option is ignored.

- --innodb[=*value*]

  Controls loading of the InnoDB storage engine, if the server was compiled with InnoDB support. This option has a tristate format, with possible values of OFF, ON, or FORCE. See Section 5.1.8.1, "Installing and Uninstalling Plugins".

  To disable InnoDB, use --innodb=OFF or --skip-innodb. In this case, because the default storage engine is InnoDB, the server will not start unless you also use --default-storage-engine and --default-tmp-storage-engine to set the default to some other engine for both permanent and TEMPORARY tables.

- --innodb-status-file

Command-Line Format	--innodb-status-file	
Option-File Format	innodb-status-file	
	Permitted Values	
	Type	boolean
	Default	OFF

  Controls whether InnoDB creates a file named innodb_status.*pid* in the MySQL data directory. If enabled, InnoDB periodically writes the output of SHOW ENGINE INNODB STATUS to this file.

  By default, the file is not created. To create it, start mysqld with the --innodb-status-file=1 option. The file is deleted during normal shutdown.

- --skip-innodb

  Disable the InnoDB storage engine. See the description of --innodb.

InnoDB System Variables

- daemon_memcached_enable_binlog

Command-Line Format	--daemon_memcached_enable_binlog=#	
Option-File Format	daemon_memcached_enable_binlog	
System Variable Name	daemon_memcached_enable_binlog	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	boolean
	Default	false

  See Section 14.2.16, "InnoDB Integration with memcached" for usage details for this option.

- daemon_memcached_engine_lib_name

Command-Line Format	--daemon_memcached_engine_lib_name=library	
Option-File Format	daemon_memcached_engine_lib_name	
System Variable Name	daemon_memcached_engine_lib_name	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	file name
	Default	innodb_engine.so

Specifies the shared library that implements the InnoDB memcached plugin.

See Section 14.2.16, "InnoDB Integration with memcached" for usage details for this option.

- daemon_memcached_engine_lib_path

Command-Line Format	--daemon_memcached_engine_lib_path=directory	
Option-File Format	daemon_memcached_engine_lib_path	
System Variable Name	daemon_memcached_engine_lib_path	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	directory name
	Default	NULL

The path of the directory containing the shared library that implements the InnoDB memcached plugin. The default value is NULL, representing the MySQL plugin directory. You should not need to modify this parameter unless specifying a different storage engine memcached plugin that is located outside of the MySQL plugin directory.

See Section 14.2.16, "InnoDB Integration with memcached" for usage details for this option.

- daemon_memcached_option

Command-Line Format	--daemon_memcached_option=options	
Option-File Format	daemon_memcached_option	
System Variable Name	daemon_memcached_option	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	string
	Default	

Used to pass space-separated memcached options to the underlying memcached memory object caching daemon on startup. For example, you might change the port that memcached listens on, reduce the maximum number of simultaneous connections, change the maximum memory size for a key/value pair, or enable debugging messages for the error log.

See Section 14.2.16, “InnoDB Integration with memcached” for usage details for this option. For information about memcached options, refer to the memcached man page.

- daemon_memcached_r_batch_size

Command-Line Format	--daemon_memcached_r_batch_size=#	
Option-File Format	daemon_memcached_r_batch_size	
System Variable Name	daemon_memcached_r_batch_size	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	numeric
	Default	1

Specifies how many memcached read operations (get) to perform before doing a COMMIT to start a new transaction. Counterpart of daemon_memcached_w_batch_size.

This value is set to 1 by default, so that any changes made to the table through SQL statements are immediately visible to the memcached operations. You might increase it to reduce the overhead from frequent commits on a system where the underlying table is only being accessed through the memcached interface. If you set the value too large, the amount of undo or redo data could impose some storage overhead, as with any long-running transaction.

See Section 14.2.16, “InnoDB Integration with memcached” for usage details for this option.

- daemon_memcached_w_batch_size

Command-Line Format	--daemon_memcached_w_batch_size=#	
Option-File Format	daemon_memcached_w_batch_size	
System Variable Name	daemon_memcached_w_batch_size	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	numeric
	Default	1

Specifies how many memcached write operations, such as add, set, or incr, to perform before doing a COMMIT to start a new transaction. Counterpart of daemon_memcached_r_batch_size.

This value is set to 1 by default, on the assumption that any data being stored is important to preserve in case of an outage and should immediately be committed. When storing non-critical data, you might increase this value to reduce the overhead from frequent commits; but then the last *N*-1 uncommitted write operations could be lost in case of a crash.

See Section 14.2.16, “InnoDB Integration with memcached” for usage details for this option.

- ignore_builtin_innodb

Deprecated	5.5.22

Command-Line Format	--ignore-builtin-innodb	
Option-File Format	ignore-builtin-innodb	
System Variable Name	ignore_builtin_innodb	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	boolean

See the description of --ignore-builtin-innodb under "InnoDB Command Options" earlier in this section.

- innodb_adaptive_flushing

Command-Line Format	--innodb_adaptive_flushing=#	
Option-File Format	innodb_adaptive_flushing	
System Variable Name	innodb_adaptive_flushing	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	boolean
	Default	ON

Specifies whether to dynamically adjust the rate of flushing dirty pages in the InnoDB buffer pool based on the workload. Adjusting the flush rate dynamically is intended to avoid bursts of I/O activity. This setting is enabled by default. See Controlling the Flushing Rate of Dirty Pages from the InnoDB Buffer Pool for more information. For general I/O tuning advice, see Section 8.5.7, "Optimizing InnoDB Disk I/O".

- innodb_adaptive_flushing_lwm

Command-Line Format	--innodb_adaptive_flushing_lwm=#	
Option-File Format	innodb_adaptive_flushing_lwm	
System Variable Name	innodb_adaptive_flushing_lwm	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	numeric
	Default	10
	Range	0 .. 70

Low water mark representing percentage of redo log capacity at which adaptive flushing is enabled.

- innodb_adaptive_hash_index

Command-Line Format	--innodb_adaptive_hash_index=#

Option-File Format	innodb_adaptive_hash_index	
System Variable Name	innodb_adaptive_hash_index	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	boolean
	Default	ON

Whether the InnoDB adaptive hash index is enabled or disabled. As described in Controlling Adaptive Hash Indexing, it may be desirable, depending on your workload, to dynamically enable or disable adaptive hash indexing to improve query performance. Because the adaptive hash index may not be useful for all workloads, conduct benchmarks with it both enabled and disabled, using realistic workloads. See Adaptive Hash Indexes for details.

This variable is enabled by default. You can modify this parameter using the `SET GLOBAL` statement, without restarting the server. Changing the setting requires the `SUPER` privilege. You can also use `--skip-innodb_adaptive_hash_index` at server startup to disable it.

Disabling the adaptive hash index empties the hash table immediately. Normal operations can continue while the hash table is emptied, and executing queries that were using the hash table access the index B-trees directly instead. When the adaptive hash index is re-enabled, the hash table is populated again during normal operation.

- `innodb_adaptive_max_sleep_delay`

Command-Line Format	--innodb_adaptive_max_sleep_delay=#	
Option-File Format	innodb_adaptive_max_sleep_delay	
System Variable Name	innodb_adaptive_max_sleep_delay	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	numeric
	Default	150000
	Range	0 .. 1000000

Allows `InnoDB` to automatically adjust the value of `innodb_thread_sleep_delay` up or down according to the current workload. Any non-zero value enables automated, dynamic adjustment of the `innodb_thread_sleep_delay` value, up to the maximum value specified in the `innodb_adaptive_max_sleep_delay` option. The value represents the number of microseconds. This option can be useful in busy systems, with greater than 16 `InnoDB` threads. (In practice, it is most valuable for MySQL systems with hundreds or thousands of simultaneous connections.)

- `innodb_additional_mem_pool_size`

Deprecated	5.6.3
Removed	5.7.4
Command-Line Format	--innodb_additional_mem_pool_size=#
Option-File Format	innodb_additional_mem_pool_size

System Variable Name	innodb_additional_mem_pool_size	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	numeric
	Default	8388608
	Range	2097152 .. 4294967295

The size in bytes of a memory pool InnoDB uses to store data dictionary information and other internal data structures. The more tables you have in your application, the more memory you allocate here. If InnoDB runs out of memory in this pool, it starts to allocate memory from the operating system and writes warning messages to the MySQL error log. The default value is 8MB.

This variable relates to the InnoDB internal memory allocator, which is unused if innodb_use_sys_malloc is enabled.

innodb_additional_mem_pool_size was deprecated in MySQL 5.6.3 and removed MySQL 5.7.4.

- innodb_api_bk_commit_interval

Command-Line Format	--innodb_api_bk_commit_interval=#	
Option-File Format	innodb_api_bk_commit_interval	
System Variable Name	innodb_api_bk_commit_interval	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	numeric
	Default	5
	Range	1 .. 1073741824

How often to auto-commit idle connections that use the InnoDB memcached interface, in seconds. See Section 14.2.16, “InnoDB Integration with memcached” for usage details for this option.

- innodb_api_disable_rowlock

Command-Line Format	--innodb_api_disable_rowlock=#	
Option-File Format	innodb_api_disable_rowlock	
System Variable Name	innodb_api_disable_rowlock	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	boolean
	Default	OFF

Use this variable to disable row locks when InnoDB memcached performs DML operations. By default, innodb_api_disable_rowlock is set to OFF which means that memcached requests row locks for get and set operations. When innodb_api_disable_rowlock is set to ON, memcached requests a table lock instead of row locks.

The innodb_api_disable_rowlock option is not dynamic. It must be specified on the mysqld command line or entered in the MySQL configuration file. Configuration takes effect when the plugin is installed, which you do each time the MySQL server is started.

- innodb_api_enable_binlog

Command-Line Format	--innodb_api_enable_binlog=#	
Option-File Format	innodb_api_enable_binlog	
System Variable Name	innodb_api_enable_binlog	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	boolean
	Default	OFF

Lets you use the InnoDB memcached plugin with the MySQL binary log. See Section 14.2.16, “InnoDB Integration with memcached” for usage details for this option.

- innodb_api_enable_mdl

Command-Line Format	--innodb_api_enable_mdl=#	
Option-File Format	innodb_api_enable_mdl	
System Variable Name	innodb_api_enable_mdl	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	boolean
	Default	OFF

Locks the table used by the InnoDB memcached plugin, so that it cannot be dropped or altered by DDL through the SQL interface. See Section 14.2.16, “InnoDB Integration with memcached” for usage details for this option.

- innodb_api_trx_level

Command-Line Format	--innodb_api_trx_level=#	
Option-File Format	innodb_api_trx_level	
System Variable Name	innodb_api_trx_level	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	numeric
	Default	0

Lets you control the transaction isolation level on queries processed by the memcached interface. See Section 14.2.16, “InnoDB Integration with memcached” for usage details for this option. The constants corresponding to the familiar names are:

- 0 = READ UNCOMMITTED
- 1 = READ COMMITTED
- 2 = REPEATABLE READ
- 3 = SERIALIZABLE

- innodb_autoextend_increment

Command-Line Format	--innodb_autoextend_increment=#	
Option-File Format	innodb_autoextend_increment	
System Variable Name	innodb_autoextend_increment	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	numeric
	Default	64
	Range	1 .. 1000

The increment size (in MB) for extending the size of an auto-extend InnoDB system tablespace file when it becomes full. The default value is 64. This variable does not affect the per-table tablespace files that are created if you use innodb_file_per_table=1. Those files are auto-extending regardless of the value of innodb_autoextend_increment. The initial extensions are by small amounts, after which extensions occur in increments of 4MB.

- innodb_autoinc_lock_mode

Command-Line Format	--innodb_autoinc_lock_mode=#	
Option-File Format	innodb_autoinc_lock_mode	
System Variable Name	innodb_autoinc_lock_mode	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	numeric
	Default	1
	Valid Values	0
		1
		2

The lock mode to use for generating auto-increment values. The permissible values are 0, 1, or 2, for “traditional”, “consecutive”, or “interleaved” lock mode, respectively. Section 14.2.6.5, “AUTO_INCREMENT Handling in InnoDB”, describes the characteristics of these modes.

This variable has a default of 1 (“consecutive” lock mode).

- `innodb_buffer_pool_dump_at_shutdown`

Command-Line Format	`--innodb_buffer_pool_dump_at_shutdown=#`	
Option-File Format	`innodb_buffer_pool_dump_at_shutdown`	
System Variable Name	`innodb_buffer_pool_dump_at_shutdown`	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	`boolean`
	Default	`OFF`

Specifies whether to record the pages cached in the InnoDB buffer pool when the MySQL server is shut down, to shorten the warmup process at the next restart. Typically used in combination with `innodb_buffer_pool_load_at_startup`.

- `innodb_buffer_pool_dump_now`

Command-Line Format	`--innodb_buffer_pool_dump_now=#`	
Option-File Format	`innodb_buffer_pool_dump_now`	
System Variable Name	`innodb_buffer_pool_dump_now`	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	`boolean`
	Default	`OFF`

Immediately records the pages cached in the InnoDB buffer pool. Typically used in combination with `innodb_buffer_pool_load_now`.

- `innodb_buffer_pool_dump_pct`

Introduced	5.7.2	
Command-Line Format	`--innodb_buffer_pool_dump_pct=#`	
Option-File Format	`innodb_buffer_pool_dump_pct`	
System Variable Name	`innodb_buffer_pool_dump_pct`	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	`numeric`
	Default	`100`
	Range	`1 .. 100`

Specifies the percentage of the most recently used pages for each buffer pool to read out and dump. The range is 1 to 100 with a default value of 100 (dump all pages). For example, if there are 4 buffer pools with 100 pages each, and innodb_buffer_pool_dump_pct is set to 40, the 40 most recently used pages from each buffer pool will be dumped.

- innodb_buffer_pool_filename

Command-Line Format	--innodb_buffer_pool_filename=file	
Option-File Format	innodb_buffer_pool_filename	
System Variable Name	innodb_buffer_pool_filename	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	file name
	Default	ib_buffer_pool

Specifies the file that holds the list of page numbers produced by innodb_buffer_pool_dump_at_shutdown or innodb_buffer_pool_dump_now.

- innodb_buffer_pool_instances

Command-Line Format	--innodb_buffer_pool_instances=#	
Option-File Format	innodb_buffer_pool_instances	
System Variable Name	innodb_buffer_pool_instances	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	numeric
	Default	-1 (autosized)
	Range	1 .. 64

The number of regions that the InnoDB buffer pool is divided into. For systems with buffer pools in the multi-gigabyte range, dividing the buffer pool into separate instances can improve concurrency, by reducing contention as different threads read and write to cached pages. Each page that is stored in or read from the buffer pool is assigned to one of the buffer pool instances randomly, using a hashing function. Each buffer pool manages its own free lists, flush lists, LRUs, and all other data structures connected to a buffer pool, and is protected by its own buffer pool mutex.

This option takes effect only when you set the innodb_buffer_pool_size to a size of 1 gigabyte or more. The total size you specify is divided among all the buffer pools. For best efficiency, specify a combination of innodb_buffer_pool_instances and innodb_buffer_pool_size so that each buffer pool instance is at least 1 gigabyte.

The default is 8, except on 32-bit Windows systems, where the default depends on the value of innodb_buffer_pool_size:

- If innodb_buffer_pool_size is greater than 1.3GB, the default for innodb_buffer_pool_instances is innodb_buffer_pool_size/128MB, with individual memory allocation requests for each chunk. 1.3GB was chosen as the boundary at which there is significant risk for 32-bit Windows to be unable to allocate the contiguous address space needed for a single buffer pool.

  - Otherwise, the default is 1.

- innodb_buffer_pool_load_abort

Command-Line Format	--innodb_buffer_pool_load_abort=#	
Option-File Format	innodb_buffer_pool_load_abort	
System Variable Name	innodb_buffer_pool_load_abort	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	boolean
	Default	OFF

Interrupts the process of restoring InnoDB buffer pool contents triggered by innodb_buffer_pool_load_at_startup or innodb_buffer_pool_load_now.

- innodb_buffer_pool_load_at_startup

Command-Line Format	--innodb_buffer_pool_load_at_startup=#	
Option-File Format	innodb_buffer_pool_load_at_startup	
System Variable Name	innodb_buffer_pool_load_at_startup	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	boolean
	Default	OFF

Specifies that, on MySQL server startup, the InnoDB buffer pool is automatically warmed up by loading the same pages it held at an earlier time. Typically used in combination with innodb_buffer_pool_dump_at_shutdown.

- innodb_buffer_pool_load_now

Command-Line Format	--innodb_buffer_pool_load_now=#	
Option-File Format	innodb_buffer_pool_load_now	
System Variable Name	innodb_buffer_pool_load_now	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	boolean
	Default	OFF

Immediately warms up the InnoDB buffer pool by loading a set of data pages, without waiting for a server restart. Can be useful to bring cache memory back to a known state during benchmarking, or to ready the MySQL server to resume its normal workload after running queries for reports or maintenance.

- innodb_buffer_pool_size

Command-Line Format	--innodb_buffer_pool_size=#	
Option-File Format	innodb_buffer_pool_size	
System Variable Name	innodb_buffer_pool_size	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Platform Bit Size	32
	Type	numeric
	Default	134217728
	Range	5242880 .. 2**32-1
	Permitted Values	
	Platform Bit Size	64
	Type	numeric
	Default	134217728
	Range	5242880 .. 2**64-1

The size in bytes of the buffer pool, the memory area where InnoDB caches table and index data. The default value is 128MB. The maximum value depends on the CPU architecture; the maximum is 4294967295 (2^{32}-1) on 32-bit systems and 18446744073709551615 (2^{64}-1) on 64-bit systems. On 32-bit systems, the CPU architecture and operating system may impose a lower practical maximum size than the stated maximum. When the size of the buffer pool is greater than 1GB, setting innodb_buffer_pool_instances to a value greater than 1 can improve the scalability on a busy server.

The larger you set this value, the less disk I/O is needed to access the same data in tables more than once. On a dedicated database server, you might set this to up to 80% of the machine physical memory size. Be prepared to scale back this value if these other issues occur:

- Competition for physical memory might cause paging in the operating system.

- InnoDB reserves additional memory for buffers and control structures, so that the total allocated space is approximately 10% greater than the specified size.

- The address space must be contiguous, which can be an issue on Windows systems with DLLs that load at specific addresses.

- The time to initialize the buffer pool is roughly proportional to its size. On large installations, this initialization time might be significant. For example, on a modern Linux x86_64 server, initialization of a 10GB buffer pool takes approximately 6 seconds. See Section 8.9.1, “The InnoDB Buffer Pool”.

- innodb_change_buffer_max_size

Command-Line Format	`--innodb_change_buffer_max_size=#`	
Option-File Format	`innodb_change_buffer_max_size`	
System Variable Name	`innodb_change_buffer_max_size`	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	`numeric`
	Default	`25`
	Range	`0 .. 50`

Maximum size for the InnoDB change buffer, as a percentage of the total size of the buffer pool. You might increase this value for a MySQL server with heavy insert, update, and delete activity, or decrease it for a MySQL server with unchanging data used for reporting. For general I/O tuning advice, see Section 8.5.7, “Optimizing `InnoDB` Disk I/O”.

- `innodb_change_buffering`

Command-Line Format	`--innodb_change_buffering=#`	
Option-File Format	`innodb_change_buffering`	
System Variable Name	`innodb_change_buffering`	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	`enumeration`
	Default	`all`
	Valid Values	`inserts`
		`deletes`
		`purges`
		`changes`
		`all`
		`none`

Whether `InnoDB` performs change buffering, an optimization that delays write operations to secondary indexes so that the I/O operations can be performed sequentially. The permitted values are `inserts` (buffer insert operations), `deletes` (buffer delete operations; strictly speaking, the writes that mark index records for later deletion during a purge operation), `changes` (buffer insert and delete-marking operations), `purges` (buffer purge operations, the writes when deleted index entries are finally garbage-collected), `all` (buffer insert, delete-marking, and purge operations) and `none` (do not buffer any operations). The default is `all`. For details, see Controlling InnoDB Change Buffering. For general I/O tuning advice, see Section 8.5.7, “Optimizing `InnoDB` Disk I/O”.

- `innodb_checksum_algorithm`

Command-Line Format	`--innodb_checksum_algorithm=#`
Option-File Format	`innodb_checksum_algorithm`

System Variable Name	innodb_checksum_algorithm	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	enumeration
	Default	innodb
	Valid Values	innodb
		crc32
		none
		strict_innodb
		strict_crc32
		strict_none

Specifies how to generate and verify the checksum stored in each disk block of each InnoDB tablespace. Replaces the innodb_checksums option.

The value innodb is backward-compatible with all versions of MySQL. The value crc32 uses an algorithm that is faster to compute the checksum for every modified block, and to check the checksums for each disk read. The value none writes a constant value in the checksum field rather than computing a value based on the block data. The blocks in a tablespace can use a mix of old, new, and no checksum values, being updated gradually as the data is modified; once any blocks in a tablespace are modified to use the crc32 algorithm, the associated tables cannot be read by earlier versions of MySQL.

The strict_* forms work the same as innodb, crc32, and none, except that InnoDB halts if it encounters a mix of checksum values in the same tablespace. You can only use these options in a completely new instance, to set up all tablespaces for the first time. The strict_* settings are somewhat faster, because they do not need to compute both new and old checksum values to accept both during disk reads.

For usage information, including a matrix of valid combinations of checksum values during read and write operations, see Fast CRC32 Checksum Algorithm.

- innodb_checksums

Deprecated	5.6.3	
Command-Line Format	--innodb_checksums	
Option-File Format	innodb_checksums	
System Variable Name	innodb_checksums	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	boolean
	Default	ON

InnoDB can use checksum validation on all tablespace pages read from the disk to ensure extra fault tolerance against hardware faults or corrupted data files. This validation is enabled by default. Under specialized circumstances (such as when running benchmarks) this extra safety feature can be disabled with --skip-innodb-checksums. You can specify the method of calculating the checksum with innodb_checksum_algorithm.

In MySQL 5.6.3 and higher, this option is deprecated, replaced by innodb_checksum_algorithm. innodb_checksum_algorithm=innodb is the same as innodb_checksums=ON (the default). innodb_checksum_algorithm=none is the same as innodb_checksums=OFF. Remove any innodb_checksums options from your configuration files and startup scripts, to avoid conflicts with innodb_checksum_algorithm: innodb_checksums=OFF would automatically set innodb_checksum_algorithm=none; innodb_checksums=ON would be ignored and overridden by any other setting for innodb_checksum_algorithm.

- innodb_cmp_per_index_enabled

Command-Line Format	--innodb_cmp_per_index_enabled=#	
Option-File Format	innodb_cmp_per_index_enabled	
System Variable Name	innodb_cmp_per_index_enabled	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	boolean
	Default	OFF
	Valid Values	OFF
		ON

Enables per-index compression-related statistics in the INFORMATION_SCHEMA.INNODB_CMP_PER_INDEX table. Because these statistics can be expensive to gather, only enable this option on development, test, or slave instances during performance tuning related to InnoDB compressed tables.

- innodb_commit_concurrency

Command-Line Format	--innodb_commit_concurrency=#	
Option-File Format	innodb_commit_concurrency	
System Variable Name	innodb_commit_concurrency	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	numeric
	Default	0
	Range	0 .. 1000

The number of threads that can commit at the same time. A value of 0 (the default) permits any number of transactions to commit simultaneously.

The value of innodb_commit_concurrency cannot be changed at runtime from zero to nonzero or vice versa. The value can be changed from one nonzero value to another.

- innodb_compression_failure_threshold_pct

Command-Line Format	--innodb_compression_failure_threshold_pct=#	
Option-File Format	innodb_compression_failure_threshold_pct	
System Variable Name	innodb_compression_failure_threshold_pct	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	numeric
	Default	5
	Range	0 .. 100

Sets the cutoff point at which MySQL begins adding padding within compressed pages to avoid expensive compression failures. A value of zero disables the mechanism that monitors compression efficiency and dynamically adjusts the padding amount.

- innodb_compression_level

Command-Line Format	--innodb_compression_level=#	
Option-File Format	innodb_compression_level	
System Variable Name	innodb_compression_level	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	numeric
	Default	6
	Range	0 .. 9

Specifies the level of zlib compression to use for InnoDB compressed tables and indexes.

- innodb_compression_pad_pct_max

Command-Line Format	--innodb_compression_pad_pct_max=#	
Option-File Format	innodb_compression_pad_pct_max	
System Variable Name	innodb_compression_pad_pct_max	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	numeric
	Default	50
	Range	0 .. 75

Specifies the maximum percentage that can be reserved as free space within each compressed page, allowing room to reorganize the data and modification log within the page when a compressed table or index is updated and the data might be recompressed. Only applies when innodb_compression_failure_threshold_pct is set to a non-zero value, and the rate of compression failures passes the cutoff point.

- innodb_concurrency_tickets

Command-Line Format	--innodb_concurrency_tickets=#	
Option-File Format	innodb_concurrency_tickets	
System Variable Name	innodb_concurrency_tickets	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	numeric
	Default	5000
	Range	1 .. 4294967295

Determines the number of threads that can enter InnoDB concurrently. A thread is placed in a queue when it tries to enter InnoDB if the number of threads has already reached the concurrency limit. When a thread is permitted to enter InnoDB, it is given a number of “free tickets” equal to the value of innodb_concurrency_tickets, and the thread can enter and leave InnoDB freely until it has used up its tickets. After that point, the thread again becomes subject to the concurrency check (and possible queuing) the next time it tries to enter InnoDB. The default value is 5000.

- innodb_data_file_path

Command-Line Format	--innodb_data_file_path=name	
Option-File Format	innodb_data_file_path	
System Variable Name	innodb_data_file_path	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	string
	Default	ibdata1:12M:autoextend

The paths to individual InnoDB data files and their sizes. The full directory path to each data file is formed by concatenating innodb_data_home_dir to each path specified here. The file sizes are specified KB, MB or GB (1024MB) by appending K, M or G to the size value. If specifying data file size in kilobytes (KB), do so in multiples of 1024. Otherwise, KB values are rounded off to nearest megabyte (MB) boundary. The sum of the sizes of the files must be at least slightly larger than 10MB. If you do not specify innodb_data_file_path, the default behavior is to create a single auto-extending data file, slightly larger than 12MB, named ibdata1. The size limit of individual files is determined by your operating system. You can set the file size to more than 4GB on those operating systems that support big files. You can also use raw disk partitions as data files. For detailed information on configuring InnoDB tablespace files, see Section 14.2.3, “InnoDB Configuration”.

- innodb_data_home_dir

Command-Line Format	--innodb_data_home_dir=path
Option-File Format	innodb_data_home_dir

System Variable Name	innodb_data_home_dir	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	directory name

The common part of the directory path for all InnoDB data files in the system tablespace. This setting does not affect the location of file-per-table tablespaces when innodb_file_per_table is enabled. The default value is the MySQL data directory. If you specify the value as an empty string, you can use absolute file paths in innodb_data_file_path.

- innodb_disable_sort_file_cache

Command-Line Format	--innodb_disable_sort_file_cache=#	
Option-File Format	innodb_disable_sort_file_cache	
System Variable Name	innodb_disable_sort_file_cache	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	boolean
	Default	OFF

If enabled, this variable disables the operating system file system cache for merge-sort temporary files. The effect is to open such files with the equivalent of O_DIRECT.

- innodb_doublewrite

Command-Line Format	--innodb-doublewrite	
Option-File Format	innodb_doublewrite	
System Variable Name	innodb_doublewrite	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	boolean

If this variable is enabled (the default), InnoDB stores all data twice, first to the doublewrite buffer, then to the actual data files. This variable can be turned off with --skip-innodb_doublewrite for benchmarks or cases when top performance is needed rather than concern for data integrity or possible failures.

- innodb_fast_shutdown

Command-Line Format	--innodb_fast_shutdown[=#]
Option-File Format	innodb_fast_shutdown
System Variable Name	innodb_fast_shutdown
Variable Scope	Global

Dynamic Variable	Yes	
	Permitted Values	
	Type	numeric
	Default	1
	Valid Values	0
		1
		2

The InnoDB shutdown mode. If the value is 0, InnoDB does a slow shutdown, a full purge and an insert buffer merge before shutting down. If the value is 1 (the default), InnoDB skips these operations at shutdown, a process known as a fast shutdown. If the value is 2, InnoDB flushes its logs and shuts down cold, as if MySQL had crashed; no committed transactions are lost, but the crash recovery operation makes the next startup take longer.

The slow shutdown can take minutes, or even hours in extreme cases where substantial amounts of data are still buffered. Use the slow shutdown technique before upgrading or downgrading between MySQL major releases, so that all data files are fully prepared in case the upgrade process updates the file format.

Use innodb_fast_shutdown=2 in emergency or troubleshooting situations, to get the absolute fastest shutdown if data is at risk of corruption.

- innodb_file_format

Command-Line Format	--innodb_file_format=#	
Option-File Format	innodb_file_format	
System Variable Name	innodb_file_format	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	string
	Default	Antelope
	Valid Values	Antelope
		Barracuda

The file format to use for new InnoDB tables. Currently, Antelope and Barracuda are supported. This applies only for tables that have their own tablespace, so for it to have an effect, innodb_file_per_table must be enabled. The Barracuda file format is required for certain InnoDB features such as table compression.

Be aware that ALTER TABLE operations that recreate InnoDB tables (ALGORITHM=COPY) will use the current innodb_file_format setting (the conditions outlined above still apply).

- innodb_file_format_check

Command-Line Format	--innodb_file_format_check=#
Option-File Format	innodb_file_format_check
System Variable Name	innodb_file_format_check

Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	boolean
	Default	ON

This variable can be set to 1 or 0 at server startup to enable or disable whether InnoDB checks the file format tag in the system tablespace (for example, Antelope or Barracuda). If the tag is checked and is higher than that supported by the current version of InnoDB, an error occurs and InnoDB does not start. If the tag is not higher, InnoDB sets the value of innodb_file_format_max to the file format tag.

> **Note**
>
> Despite the default value sometimes being displayed as ON or OFF, always use the numeric values 1 or 0 to turn this option on or off in your configuration file or command line.

- innodb_file_format_max

Command-Line Format	--innodb_file_format_max=#	
Option-File Format	innodb_file_format_max	
System Variable Name	innodb_file_format_max	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	string
	Default	Antelope
	Valid Values	Antelope
		Barracuda

At server startup, InnoDB sets the value of this variable to the file format tag in the system tablespace (for example, Antelope or Barracuda). If the server creates or opens a table with a “higher” file format, it sets the value of innodb_file_format_max to that format.

- innodb_file_per_table

Command-Line Format	--innodb_file_per_table	
Option-File Format	innodb_file_per_table	
System Variable Name	innodb_file_per_table	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	boolean
	Default	ON

When innodb_file_per_table is enabled (the default in 5.6.6 and higher), InnoDB stores the data and indexes for each newly created table in a separate .ibd file, rather than in the system tablespace. The storage for these InnoDB tables is reclaimed when the tables are dropped or truncated. This setting enables several other InnoDB features, such as table compression. See Section 14.2.5.2, “InnoDB File-Per-Table Mode” for details about such features as well as advantages and disadvantages of using per-table tablespaces.

Be aware that enabling innodb_file_per_table also means that an ALTER TABLE operation will move InnoDB table from the system tablespace to an individual .ibd file in cases where ALTER TABLE recreates the table (ALGORITHM=COPY).

When innodb_file_per_table is disabled, InnoDB stores the data for all tables and indexes in the ibdata files that make up the system tablespace. This setting reduces the performance overhead of filesystem operations for operations such as DROP TABLE or TRUNCATE TABLE. It is most appropriate for a server environment where entire storage devices are devoted to MySQL data. Because the system tablespace never shrinks, and is shared across all databases in an instance, avoid loading huge amounts of temporary data on a space-constrained system when innodb_file_per_table=OFF. Set up a separate instance in such cases, so that you can drop the entire instance to reclaim the space.

By default, innodb_file_per_table is enabled as of MySQL 5.6.6, disabled before that. Consider disabling it if backward compatibility with MySQL 5.5 or 5.1 is a concern. This will prevent ALTER TABLE from moving InnoDB tables from the system tablespace to individual .ibd files.

innodb_file_per_table is dynamic and can be set ON or OFF using SET GLOBAL. You can also set this parameter in the MySQL configuration file (my.cnf or my.ini) but this requires shutting down and restarting the server.

Dynamically changing the value of this parameter requires the SUPER privilege and immediately affects the operation of all connections.

- innodb_flush_log_at_timeout

System Variable Name	innodb_flush_log_at_timeout	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	numeric
	Default	1
	Range	0 .. 2700

Write and flush the logs every *N* seconds. innodb_flush_log_at_timeout was introduced in MySQL 5.6.6. It allows the timeout period between flushes to be increased in order to reduce flushing and avoid impacting performance of binary log group commit. Prior to MySQL 5.6.6, flushing frequency was once per second. The default setting for innodb_flush_log_at_timeout is also once per second.

- innodb_flush_log_at_trx_commit

Command-Line Format	--innodb_flush_log_at_trx_commit[=#]
Option-File Format	innodb_flush_log_at_trx_commit
System Variable Name	innodb_flush_log_at_trx_commit

Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	enumeration
	Default	1
	Valid Values	0
		1
		2

Controls the balance between strict ACID compliance for commit operations, and higher performance that is possible when commit-related I/O operations are rearranged and done in batches. You can achieve better performance by changing the default value, but then you can lose up to one second worth of transactions in a crash.

- The default value of 1 is required for full ACID compliance. With this value, the log buffer is written out to the log file at each transaction commit and the flush to disk operation is performed on the log file.
- With a value of 0, any mysqld process crash can erase up to a second of transactions. The log buffer is written out to the log file once per second and the flush to disk operation is performed on the log file. No writes from the log buffer to the log file are performed at transaction commit. Once-per-second flushing is not 100% guaranteed to happen every second, due to process scheduling issues.
- With a value of 2, any mysqld process crash can erase up to a second of transactions. The log buffer is written out to the log file at each commit. The flush to disk operation is performed on the log file once per second. Once-per-second flushing is not 100% guaranteed to happen every second, due to process scheduling issues.
- As of MySQL 5.6.6, InnoDB log flushing frequency is controlled by innodb_flush_log_at_timeout, which allows you to set log flushing frequency to *N* seconds (where *N* is 1 ... 2700, with a default value of 1). However, any mysqld process crash can erase up to *N* seconds of transactions.
- DDL changes and other internal InnoDB activities flush the InnoDB log independent of the innodb_flush_log_at_trx_commit setting.
- InnoDB's crash recovery works regardless of the innodb_flush_log_at_trx_commit setting. Transactions are either applied entirely or erased entirely.

For durability and consistency in a replication setup that uses InnoDB with transactions:

- If binary logging is enabled, set sync_binlog=1.
- Always set innodb_flush_log_at_trx_commit=1.

Caution

Many operating systems and some disk hardware fool the flush-to-disk operation. They may tell mysqld that the flush has taken place, even though it has not. Then the durability of transactions is not guaranteed even with the setting 1, and in the worst case a power outage can even corrupt InnoDB data. Using a battery-backed disk cache in the SCSI disk controller or in the disk itself speeds up file flushes, and makes the operation safer. You can also try using the Unix command hdparm to disable the caching of disk writes in hardware caches, or use some other command specific to the hardware vendor.

- innodb_flush_method

Command-Line Format	--innodb_flush_method=name	
Option-File Format	innodb_flush_method	
System Variable Name	innodb_flush_method	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type (Linux)	string
	Default	fdatasync
	Valid Values	fdatasync
		O_DSYNC
		O_DIRECT
		O_DIRECT_NO_FSYNC
	Permitted Values	
	Type (Solaris)	string
	Default	fdatasync
	Valid Values	fdatasync
		O_DSYNC
		O_DIRECT
		O_DIRECT_NO_FSYNC
	Permitted Values	
	Type (HP-UX)	string
	Default	fdatasync
	Valid Values	fdatasync
		O_DSYNC
		O_DIRECT
		O_DIRECT_NO_FSYNC

Controls the system calls used to flush data to the InnoDB data files and log files, which can influence I/O throughput. This variable is relevant only for Unix and Linux systems. On Windows systems, the flush method is always async_unbuffered and cannot be changed.

By default, InnoDB uses the fsync() system call to flush both the data and log files. If innodb_flush_method option is set to O_DSYNC, InnoDB uses O_SYNC to open and flush the log files, and fsync() to flush the data files. If O_DIRECT is specified (available on some GNU/Linux versions, FreeBSD, and Solaris), InnoDB uses O_DIRECT (or directio() on Solaris) to open the data files, and uses fsync() to flush both the data and log files. Note that InnoDB uses fsync() instead of fdatasync(), and it does not use O_DSYNC by default because there have been problems with it on many varieties of Unix.

An alternative setting is O_DIRECT_NO_FSYNC: it uses the O_DIRECT flag during flushing I/O, but skips the fsync() system call afterwards. This setting is suitable for some types of filesystems but not others. For example, it is not suitable for XFS. If you are not sure whether the filesystem you use requires an fsync(), for example to preserve all file metadata, use O_DIRECT instead.

Depending on hardware configuration, setting innodb_flush_method to O_DIRECT or O_DIRECT_NO_FSYNC can have either a positive or negative effect on performance. Benchmark your particular configuration to decide which setting to use, or whether to keep the default. Examine the Innodb_data_fsyncs status variable to see the overall number of fsync() calls done with each setting. The mix of read and write operations in your workload can also affect which setting performs better for you. For example, on a system with a hardware RAID controller and battery-backed write cache, O_DIRECT can help to avoid double buffering between the InnoDB buffer pool and the operating system's filesystem cache. On some systems where InnoDB data and log files are located on a SAN, the default value or O_DSYNC might be faster for a read-heavy workload with mostly SELECT statements. Always test this parameter with the same type of hardware and workload that reflects your production environment. For general I/O tuning advice, see Section 8.5.7, “Optimizing InnoDB Disk I/O”.

Formerly, a value of fdatasync also specified the default behavior. This value was removed, due to confusion that a value of fdatasync caused fsync() system calls rather than fdatasync() for flushing. To obtain the default value now, do not set any value for innodb_flush_method at startup.

- innodb_flush_neighbors

Command-Line Format	--innodb_flush_neighbors	
Option-File Format	innodb_flush_neighbors	
System Variable Name	innodb_flush_neighbors	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	enumeration
	Default	1
	Valid Values	0
		1
		2

Specifies whether flushing a page from the InnoDB buffer pool also flushes other dirty pages in the same extent.

- The default value of 1 flushes contiguous dirty pages in the same extent from the buffer pool.
- A setting of 0 turns innodb_flush_neighbors off and no other dirty pages are flushed from the buffer pool.
- A setting of 2 flushes dirty pages in the same extent from the buffer pool.

When the table data is stored on a traditional HDD storage device, flushing such neighbor pages in one operation reduces I/O overhead (primarily for disk seek operations) compared to flushing individual pages at different times. For table data stored on SSD, seek time is not a significant factor and you can turn this setting off to spread out the write operations. For general I/O tuning advice, see Section 8.5.7, “Optimizing InnoDB Disk I/O”.

- innodb_flushing_avg_loops

Command-Line Format	--innodb_flushing_avg_loops=#	
Option-File Format	innodb_flushing_avg_loops	
System Variable Name	innodb_flushing_avg_loops	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	numeric
	Default	30
	Range	1 .. 1000

Number of iterations for which InnoDB keeps the previously calculated snapshot of the flushing state, controlling how quickly adaptive flushing responds to changing workloads. Increasing the value makes the rate of flush operations change smoothly and gradually as the workload changes. Decreasing the value makes adaptive flushing adjust quickly to workload changes, which can cause spikes in flushing activity if the workload increases and decreases suddenly.

- innodb_force_load_corrupted

Command-Line Format	--innodb_force_load_corrupted	
Option-File Format	innodb_force_load_corrupted	
System Variable Name	innodb_force_load_corrupted	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	boolean
	Default	OFF

Lets InnoDB load tables at startup that are marked as corrupted. Use only during troubleshooting, to recover data that is otherwise inaccessible. When troubleshooting is complete, turn this setting back off and restart the server.

- innodb_force_recovery

Command-Line Format	--innodb_force_recovery=#	
Option-File Format	innodb_force_recovery	
System Variable Name	innodb_force_recovery	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	numeric
	Default	0

	Range	`0 .. 6`

The crash recovery mode, typically only changed in serious troubleshooting situations. Possible values are from 0 to 6. The meanings of these values are described in Section 14.2.17.2, “Starting InnoDB on a Corrupted Database”.

Warning

Only set this variable greater than 0 in an emergency situation, to dump your tables from a corrupt database. As a safety measure, InnoDB prevents any changes to its data when this variable is greater than 0.

This restriction prohibits some queries that use WHERE or ORDER BY clauses, because high values can prevent queries from using indexes, to guard against possible corrupt index data.

The restriction may also cause replication administration commands to fail with an error, as replication options such as --relay-log-info-repository=TABLE and --master-info-repository=TABLE store information in tables in InnoDB.

- innodb_ft_aux_table

Command-Line Format	--innodb_ft_aux_table=#	from 5.7.2
Option-File Format	innodb_ft_aux_table	from 5.7.2
System Variable Name	innodb_ft_aux_table	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	string

Specifies the qualified name of an InnoDB table containing a FULLTEXT index. This variable is intended for diagnostic purposes.

After you set this variable to a name in the format *db_name/table_name*, the INFORMATION_SCHEMA tables INNODB_FT_INDEX_TABLE, INNODB_FT_INDEX_CACHE, INNODB_FT_CONFIG, INNODB_FT_DELETED, and INNODB_FT_BEING_DELETED will show information about the search index for the specified table.

- innodb_ft_cache_size

Command-Line Format	--innodb_ft_cache_size=#	
Option-File Format	innodb_ft_cache_size	
System Variable Name	innodb_ft_cache_size	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	numeric
	Default	8000000

	Range	1600000 .. 80000000

The memory allocated for the InnoDB FULLTEXT search index cache, which holds a parsed document in memory while creating an InnoDB FULLTEXT index. Index inserts and updates are only committed to disk when the innodb_ft_cache_size size limit is reached. innodb_ft_cache_size defines the cache size on a per table basis. To set a global limit for all tables, see innodb_ft_total_cache_size.

- innodb_ft_enable_diag_print

Command-Line Format	--innodb_ft_enable_diag_print=#	
Option-File Format	innodb_ft_enable_diag_print	
System Variable Name	innodb_ft_enable_diag_print	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	boolean
	Default	OFF

Whether to enable additional full-text search (FTS) diagnostic output. This option is primarily intended for advanced FTS debugging and will not be of interest to most users. Output is printed to the error log and includes information such as:

- FTS index sync progress (when the FTS cache limit is reached). For example:

```
FTS SYNC for table test, deleted count: 100 size: 10000 bytes
SYNC words: 100
```

- FTS optimize progress. For example:

```
FTS start optimize test
FTS_OPTIMIZE: optimize "mysql"
FTS_OPTIMIZE: processed "mysql"
```

- FTS index build progress. For example:

```
Number of doc processed: 1000
```

- For FTS queries, the query parsing tree, word weight, query processing time, and memory usage are printed. For example:

```
FTS Search Processing time: 1 secs: 100 millisec: row(s) 10000
Full Search Memory: 245666 (bytes),  Row: 10000
```

- innodb_ft_enable_stopword

Command-Line Format	--innodb_ft_enable_stopword=#
Option-File Format	innodb_ft_enable_stopword
System Variable Name	innodb_ft_enable_stopword
Variable Scope	Global

Dynamic Variable	Yes	
	Permitted Values	
	Type	boolean
	Default	ON

Specifies that a set of stopwords is associated with an InnoDB FULLTEXT index at the time the index is created. If the innodb_ft_user_stopword_table option is set, the stopwords are taken from that table. Else, if the innodb_ft_server_stopword_table option is set, the stopwords are taken from that table. Otherwise, a built-in set of default stopwords is used.

- innodb_ft_max_token_size

Command-Line Format	--innodb_ft_max_token_size=#	
Option-File Format	innodb_ft_max_token_size	
System Variable Name	innodb_ft_max_token_size	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	numeric
	Default	84
	Range	10 .. 252

Maximum length of words that are stored in an InnoDB FULLTEXT index. Setting a limit on this value reduces the size of the index, thus speeding up queries, by omitting long keywords or arbitrary collections of letters that are not real words and are not likely to be search terms.

- innodb_ft_min_token_size

Command-Line Format	--innodb_ft_min_token_size=#	
Option-File Format	innodb_ft_min_token_size	
System Variable Name	innodb_ft_min_token_size	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	numeric
	Default	3
	Range	0 .. 16

Minimum length of words that are stored in an InnoDB FULLTEXT index. Increasing this value reduces the size of the index, thus speeding up queries, by omitting common word that are unlikely to be significant in a search context, such as the English words “a” and “to”. For content using a CJK (Chinese, Japanese, Korean) character set, specify a value of 1.

- innodb_ft_num_word_optimize

Command-Line Format	--innodb_ft_num_word_optimize=#
Option-File Format	innodb_ft_num_word_optimize

System Variable Name	innodb_ft_num_word_optimize	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	numeric
	Default	2000

Number of words to process during each OPTIMIZE TABLE operation on an InnoDB FULLTEXT index. Because a bulk insert or update operation to a table containing a full-text search index could require substantial index maintenance to incorporate all changes, you might do a series of OPTIMIZE TABLE statements, each picking up where the last left off.

- innodb_ft_result_cache_limit

Introduced	5.7.2	
Command-Line Format	--innodb_ft_result_cache_limit=#	
Option-File Format	innodb_ft_result_cache_limit	
System Variable Name	innodb_ft_result_cache_limit	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values (>= 5.7.2)	
	Type (Windows)	numeric
	Default	2000000000
	Range	1000000 .. 2**32-1
	Permitted Values (>= 5.7.2)	
	Platform Bit Size	32
	Type (Unix)	numeric
	Default	2000000000
	Range	1000000 .. 2**32-1
	Permitted Values (>= 5.7.2, <= 5.7.3)	
	Platform Bit Size	64
	Type (Unix)	numeric
	Default	2000000000
	Range	1000000 .. 2**64-1
	Permitted Values (>= 5.7.4)	
	Type	numeric
	Default	2000000000
	Range	1000000 .. 2**32-1

The InnoDB FULLTEXT search (FTS) query result cache limit (defined in bytes) per FTS query or per thread. Intermediate and final InnoDB FTS query results are handled in memory. Use innodb_ft_result_cache_limit to place a size limit on the InnoDB FTS query result cache to avoid excessive memory consumption in case of very large InnoDB FTS query results (millions or hundreds of millions of rows, for example). Memory is allocated as required when an FTS query is processed. If the result cache size limit is reached, an error is returned indicating that the query exceeds the maximum allowed memory.

As of MySQL 5.7.4, the maximum value of innodb_ft_result_cache_limit for all platform types and platform bit sizes is 2**32-1. Bug #71554.

- innodb_ft_server_stopword_table

Command-Line Format	--innodb_ft_server_stopword_table=db_name/table_name	
Option-File Format	innodb_ft_server_stopword_table	
System Variable Name	innodb_ft_server_stopword_table	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	string
	Default	NULL

Name of the table containing a list of words to ignore when creating an InnoDB FULLTEXT index, in the format *db_name/table_name*.

Note

The stopword table must be an InnoDB table, containing a single VARCHAR column named VALUE. The stopword table must exist before you specify its name in the configuration option value.

- innodb_ft_sort_pll_degree

Command-Line Format	--innodb_ft_sort_pll_degree=#	
Option-File Format	innodb_ft_sort_pll_degree	
System Variable Name	innodb_ft_sort_pll_degree	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	numeric
	Default	2
	Range	1 .. 32

Number of threads used in parallel to index and tokenize text in an InnoDB FULLTEXT index, when building a search index for a large table. See innodb_sort_buffer_size for additional usage information.

- innodb_ft_total_cache_size

Introduced	5.7.2	
Command-Line Format	`--innodb_ft_total_cache_size=#`	
Option-File Format	`innodb_ft_total_cache_size`	
System Variable Name	`innodb_ft_total_cache_size`	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	`numeric`
	Default	`640000000`
	Range	`32000000 .. 1600000000`

The total memory allocated for the `InnoDB` FULLTEXT search index cache for all tables. Creating numerous tables, each with a full-text search index, could consume a significant portion of available memory. `innodb_ft_total_cache_size`, defines a global memory limit for all full-text search indexes to help avoid excessive memory consumption. If the global limit is reached by an index operation, a force sync is triggered.

- `innodb_ft_user_stopword_table`

Command-Line Format	`--innodb_ft_user_stopword_table=db_name/table_name`	
Option-File Format	`innodb_ft_user_stopword_table`	
System Variable Name	`innodb_ft_user_stopword_table`	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	`string`
	Default	`NULL`

Name of the table containing a list of words to ignore when creating an InnoDB `FULLTEXT` index, in the format *`db_name/table_name`*.

Note

The stopword table must be an `InnoDB` table, containing a single `VARCHAR` column named `VALUE`. The stopword table must exist before you specify its name in the configuration option value.

- `innodb_io_capacity`

Command-Line Format	`--innodb_io_capacity=#`	
Option-File Format	`innodb_io_capacity`	
System Variable Name	`innodb_io_capacity`	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	

	Platform Bit Size	32
	Type	numeric
	Default	200
	Range	100 .. 2**32-1
	Permitted Values	
	Platform Bit Size	64
	Type	numeric
	Default	200
	Range	100 .. 2**64-1

The innodb_io_capacity parameter sets an upper limit on the I/O activity performed by the InnoDB background tasks, such as flushing pages from the buffer pool and merging data from the insert buffer. The default value is 200. For busy systems capable of higher I/O rates, you can set a higher value at server startup, to help the server handle the background maintenance work associated with a high rate of row changes.

The innodb_io_capacity limit is a total limit for all buffer pool instances. When dirty pages are flushed, the innodb_io_capacity limit is divided equally among buffer pool instances.

As of MySQL 5.7.2, the innodb_io_capacity setting is also used to limit the number of buffer pool load operations per second when there is other I/O activity being performed by InnoDB background tasks.

For systems with individual 5400 RPM or 7200 RPM drives, you might lower the value to the former default of 100.

This parameter should be set to approximately the number of I/O operations that the system can perform per second. Ideally, keep this setting as low as practical, but not so low that these background activities fall behind. If the value is too high, data is removed from the buffer pool and insert buffer too quickly to provide significant benefit from the caching.

The value represents an estimated proportion of the I/O operations per second (IOPS) available to older-generation disk drives that could perform about 100 IOPS. The current default of 200 reflects that modern storage devices are capable of much higher I/O rates.

In general, you can increase the value as a function of the number of drives used for InnoDB I/O, particularly fast drives capable of high numbers of IOPS. For example, systems that use multiple disks or solid-state disks for InnoDB are likely to benefit from the ability to control this parameter.

Although you can specify a very high number, in practice such large values have little if any benefit; for example, a value of one million would be considered very high.

You can set the innodb_io_capacity value to any number 100 or greater to a maximum defined by innodb_io_capacity_max. The default value is 200. You can set the value of this parameter in the MySQL option file (my.cnf or my.ini) or change it dynamically with the SET GLOBAL command, which requires the SUPER privilege.

See Controlling the InnoDB Master Thread I/O Rate for more guidelines about this option. For general information about InnoDB I/O performance, see Section 8.5.7, “Optimizing InnoDB Disk I/O”.

- innodb_io_capacity_max

Command-Line Format	--innodb_io_capacity_max=#	
Option-File Format	innodb_io_capacity_max	
System Variable Name	innodb_io_capacity_max	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Platform Bit Size	32
	Type	numeric
	Default	see description
	Range	2000 .. 2**32-1
	Permitted Values	
	Platform Bit Size	64
	Type (Windows)	numeric
	Default	2000
	Range	2000 .. 2**32-1
	Permitted Values	
	Platform Bit Size	64
	Type (Unix)	numeric
	Default	see description
	Range	2000 .. 2**64-1

The limit up to which InnoDB is allowed to extend the innodb_io_capacity setting in case of emergency. If you specify an innodb_io_capacity setting at startup and do not specify a value for innodb_io_capacity_max, the innodb_io_capacity_max value defaults to twice the value of innodb_io_capacity, with a lower limit of 2000. 2000 is also the initial default innodb_io_capacity_max configuration value.

The innodb_io_capacity_max setting is a total limit for all buffer pool instances.

For a brief period during MySQL 5.6 development, this variable was known as innodb_max_io_capacity.

- innodb_large_prefix

Command-Line Format	--innodb_large_prefix
Option-File Format	innodb_large_prefix
System Variable Name	innodb_large_prefix
Variable Scope	Global

Dynamic Variable	Yes	
	Permitted Values	
	Type	boolean
	Default	OFF

Enable this option to allow index key prefixes longer than 767 bytes (up to 3072 bytes), for InnoDB tables that use the DYNAMIC and COMPRESSED row formats. (Creating such tables also requires the option values innodb_file_format=barracuda and innodb_file_per_table=true.) See Section 14.2.6.7, “Limits on InnoDB Tables” for the relevant maximums associated with index key prefixes under various settings.

For tables using the REDUNDANT and COMPACT row formats, this option does not affect the allowed key prefix length. It does introduce a new error possibility. When this setting is enabled, attempting to create an index prefix with a key length greater than 3072 for a REDUNDANT or COMPACT table causes an ER_INDEX_COLUMN_TOO_LONG error.

- innodb_lock_wait_timeout

Command-Line Format	--innodb_lock_wait_timeout=#	
Option-File Format	innodb_lock_wait_timeout	
System Variable Name	innodb_lock_wait_timeout	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	numeric
	Default	50
	Range	1 .. 1073741824

The length of time in seconds an InnoDB transaction waits for a row lock before giving up. The default value is 50 seconds. A transaction that tries to access a row that is locked by another InnoDB transaction waits at most this many seconds for write access to the row before issuing the following error:

```
ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction
```

When a lock wait timeout occurs, the current statement is rolled back (not the entire transaction). To have the entire transaction roll back, start the server with the --innodb_rollback_on_timeout option. See also Section 14.2.17.4, “InnoDB Error Handling”.

You might decrease this value for highly interactive applications or OLTP systems, to display user feedback quickly or put the update into a queue for processing later. You might increase this value for long-running back-end operations, such as a transform step in a data warehouse that waits for other large insert or update operations to finish.

innodb_lock_wait_timeout applies to InnoDB row locks only. A MySQL table lock does not happen inside InnoDB and this timeout does not apply to waits for table locks.

The lock wait timeout value does not apply to deadlocks, because InnoDB detects them immediately and rolls back one of the deadlocked transactions.

innodb_lock_wait_timeout can be set at runtime with the SET GLOBAL or SET SESSION statement. Changing the GLOBAL setting requires the SUPER privilege and affects the operation of all clients that subsequently connect. Any client can change the SESSION setting for innodb_lock_wait_timeout, which affects only that client.

- innodb_locks_unsafe_for_binlog

Deprecated	5.6.3	
Command-Line Format	--innodb_locks_unsafe_for_binlog	
Option-File Format	innodb_locks_unsafe_for_binlog	
System Variable Name	innodb_locks_unsafe_for_binlog	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	boolean
	Default	OFF

This variable affects how InnoDB uses gap locking for searches and index scans. As of MySQL 5.6.3, innodb_locks_unsafe_for_binlog is deprecated and will be removed in a future MySQL release.

Normally, InnoDB uses an algorithm called next-key locking that combines index-row locking with gap locking. InnoDB performs row-level locking in such a way that when it searches or scans a table index, it sets shared or exclusive locks on the index records it encounters. Thus, the row-level locks are actually index-record locks. In addition, a next-key lock on an index record also affects the “gap” before that index record. That is, a next-key lock is an index-record lock plus a gap lock on the gap preceding the index record. If one session has a shared or exclusive lock on record R in an index, another session cannot insert a new index record in the gap immediately before R in the index order. See Section 14.2.2.6, “InnoDB Record, Gap, and Next-Key Locks”.

By default, the value of innodb_locks_unsafe_for_binlog is 0 (disabled), which means that gap locking is enabled: InnoDB uses next-key locks for searches and index scans. To enable the variable, set it to 1. This causes gap locking to be disabled: InnoDB uses only index-record locks for searches and index scans.

Enabling innodb_locks_unsafe_for_binlog does not disable the use of gap locking for foreign-key constraint checking or duplicate-key checking.

The effect of enabling innodb_locks_unsafe_for_binlog is similar to but not identical to setting the transaction isolation level to READ COMMITTED:

- Enabling innodb_locks_unsafe_for_binlog is a global setting and affects all sessions, whereas the isolation level can be set globally for all sessions, or individually per session.

- innodb_locks_unsafe_for_binlog can be set only at server startup, whereas the isolation level can be set at startup or changed at runtime.

READ COMMITTED therefore offers finer and more flexible control than innodb_locks_unsafe_for_binlog. For additional details about the effect of isolation level on gap locking, see Section 13.3.6, “SET TRANSACTION Syntax”.

Enabling innodb_locks_unsafe_for_binlog may cause phantom problems because other sessions can insert new rows into the gaps when gap locking is disabled. Suppose that there is an index on the id column of the child table and that you want to read and lock all rows from the table having an identifier value larger than 100, with the intention of updating some column in the selected rows later:

```
SELECT * FROM child WHERE id > 100 FOR UPDATE;
```

The query scans the index starting from the first record where id is greater than 100. If the locks set on the index records in that range do not lock out inserts made in the gaps, another session can insert a new row into the table. Consequently, if you were to execute the same SELECT again within the same transaction, you would see a new row in the result set returned by the query. This also means that if new items are added to the database, InnoDB does not guarantee serializability. Therefore, if innodb_locks_unsafe_for_binlog is enabled, InnoDB guarantees at most an isolation level of READ COMMITTED. (Conflict serializability is still guaranteed.) For additional information about phantoms, see Section 14.2.2.7, “Avoiding the Phantom Problem Using Next-Key Locking”.

Enabling innodb_locks_unsafe_for_binlog has additional effects:

- For UPDATE or DELETE statements, InnoDB holds locks only for rows that it updates or deletes. Record locks for nonmatching rows are released after MySQL has evaluated the WHERE condition. This greatly reduces the probability of deadlocks, but they can still happen.
- For UPDATE statements, if a row is already locked, InnoDB performs a “semi-consistent” read, returning the latest committed version to MySQL so that MySQL can determine whether the row matches the WHERE condition of the UPDATE. If the row matches (must be updated), MySQL reads the row again and this time InnoDB either locks it or waits for a lock on it.

Consider the following example, beginning with this table:

```
CREATE TABLE t (a INT NOT NULL, b INT) ENGINE = InnoDB;
INSERT INTO t VALUES (1,2),(2,3),(3,2),(4,3),(5,2);
COMMIT;
```

In this case, table has no indexes, so searches and index scans use the hidden clustered index for record locking (see Clustered and Secondary Indexes).

Suppose that one client performs an UPDATE using these statements:

```
SET autocommit = 0;
UPDATE t SET b = 5 WHERE b = 3;
```

Suppose also that a second client performs an UPDATE by executing these statements following those of the first client:

```
SET autocommit = 0;
UPDATE t SET b = 4 WHERE b = 2;
```

As InnoDB executes each UPDATE, it first acquires an exclusive lock for each row, and then determines whether to modify it. If InnoDB does not modify the row and innodb_locks_unsafe_for_binlog is enabled, it releases the lock. Otherwise, InnoDB retains the lock until the end of the transaction. This affects transaction processing as follows.

If innodb_locks_unsafe_for_binlog is disabled, the first UPDATE acquires x-locks and does not release any of them:

```
x-lock(1,2); retain x-lock
x-lock(2,3); update(2,3) to (2,5); retain x-lock
x-lock(3,2); retain x-lock
x-lock(4,3); update(4,3) to (4,5); retain x-lock
x-lock(5,2); retain x-lock
```

The second UPDATE blocks as soon as it tries to acquire any locks (because first update has retained locks on all rows), and does not proceed until the first UPDATE commits or rolls back:

```
x-lock(1,2); block and wait for first UPDATE to commit or roll back
```

If innodb_locks_unsafe_for_binlog is enabled, the first UPDATE acquires x-locks and releases those for rows that it does not modify:

```
x-lock(1,2); unlock(1,2)
x-lock(2,3); update(2,3) to (2,5); retain x-lock
x-lock(3,2); unlock(3,2)
x-lock(4,3); update(4,3) to (4,5); retain x-lock
x-lock(5,2); unlock(5,2)
```

For the second UPDATE, InnoDB does a “semi-consistent” read, returning the latest committed version of each row to MySQL so that MySQL can determine whether the row matches the WHERE condition of the UPDATE:

```
x-lock(1,2); update(1,2) to (1,4); retain x-lock
x-lock(2,3); unlock(2,3)
x-lock(3,2); update(3,2) to (3,4); retain x-lock
x-lock(4,3); unlock(4,3)
x-lock(5,2); update(5,2) to (5,4); retain x-lock
```

- innodb_log_buffer_size

Command-Line Format	--innodb_log_buffer_size=#	
Option-File Format	innodb_log_buffer_size	
System Variable Name	innodb_log_buffer_size	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	numeric
	Default	8388608
	Range	262144 .. 4294967295

The size in bytes of the buffer that InnoDB uses to write to the log files on disk. The default value is 8MB. A large log buffer enables large transactions to run without a need to write the log to disk before the transactions commit. Thus, if you have transactions that update, insert, or delete many rows, making the log buffer larger saves disk I/O. For general I/O tuning advice, see Section 8.5.7, “Optimizing InnoDB Disk I/O”.

- innodb_log_compressed_pages

Command-Line Format	--innodb_log_compressed_pages=#	
Option-File Format	innodb_log_compressed_pages	
System Variable Name	innodb_log_compressed_pages	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	boolean
	Default	ON

Specifies whether images of re-compressed pages are stored in InnoDB redo logs.

- innodb_log_file_size

Command-Line Format	--innodb_log_file_size=#	
Option-File Format	innodb_log_file_size	
System Variable Name	innodb_log_file_size	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	numeric
	Default	50331648
	Range	1048576 .. 512GB / innodb_log_files_in_group

The size in bytes of each log file in a log group. The combined size of log files (innodb_log_file_size * innodb_log_files_in_group) cannot exceed a maximum value that is slightly less than 512GB. A pair of 255 GB log files, for example, would allow you to approach the limit but not exceed it. The default value is 48MB. Sensible values range from 1MB to 1/*N*-th of the size of the buffer pool, where *N* is the number of log files in the group. The larger the value, the less checkpoint flush activity is needed in the buffer pool, saving disk I/O. Larger log files also make crash recovery slower, although improvements to recovery performance in MySQL 5.5 and higher make the log file size less of a consideration. For general I/O tuning advice, see Section 8.5.7, "Optimizing InnoDB Disk I/O".

- innodb_log_files_in_group

Command-Line Format	--innodb_log_files_in_group=#	
Option-File Format	innodb_log_files_in_group	
System Variable Name	innodb_log_files_in_group	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	numeric
	Default	2
	Range	2 .. 100

The number of log files in the log group. InnoDB writes to the files in a circular fashion. The default (and recommended) value is 2. The location of these files is specified by innodb_log_group_home_dir. The combined size of log files (innodb_log_file_size * innodb_log_files_in_group) can be up to 512GB.

- innodb_log_group_home_dir

Command-Line Format	--innodb_log_group_home_dir=path	
Option-File Format	innodb_log_group_home_dir	
System Variable Name	innodb_log_group_home_dir	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	directory name

The directory path to the InnoDB redo log files, whose number is specified by innodb_log_files_in_group. If you do not specify any InnoDB log variables, the default is to create two files named ib_logfile0 and ib_logfile1 in the MySQL data directory. Their size is given by the size of the innodb_log_file_size system variable.

- innodb_log_write_ahead_size

Introduced	5.7.4	
Command-Line Format	--innodb_log_write_ahead_size=#	
Option-File Format	innodb_log_write_ahead_size	
System Variable Name	innodb_log_write_ahead_size	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	numeric
	Default	8192
	Range	512 (log file block size) .. Equal to innodb_page_size

The write-ahead block size for the redo log, in bytes. To avoid “read-on-write”, set innodb_log_write_ahead_size to match the operating system or file system cache block size. Read-on-write occurs when redo log blocks are not entirely cached to the operating system or file system due to a mismatch between write-ahead block size for redo logs and operating system or file system cache block size.

Valid values for innodb_log_write_ahead_size are multiples of the InnoDB log file block size (2^n). The minimum value is the InnoDB log file block size (512). Write-ahead does not occur when the minimum value is specified. The maximum value is equal to innodb_page_size. If you specify a value for innodb_log_write_ahead_size that is larger than the innodb_page_size value, the innodb_log_write_ahead_size value is truncated to the innodb_page_size value.

Setting the innodb_log_write_ahead_size value too low in relation to the operating system or file system cache block size will result in “read-on-write”. Setting the value too high may have a slight impact on fsync performance for log file writes due to several blocks being written at once.

- innodb_lru_scan_depth

Command-Line Format	--innodb_lru_scan_depth=#	
Option-File Format	innodb_lru_scan_depth	
System Variable Name	innodb_lru_scan_depth	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Platform Bit Size	32
	Type	numeric
	Default	1024
	Range	100 .. 2**32-1
	Permitted Values	
	Platform Bit Size	64
	Type	numeric
	Default	1024
	Range	100 .. 2**64-1

A parameter that influences the algorithms and heuristics for the flush operation for the InnoDB buffer pool. Primarily of interest to performance experts tuning I/O-intensive workloads. It specifies, per buffer pool instance, how far down the buffer pool LRU list the page_cleaner thread scans looking for dirty pages to flush. This is a background operation performed once a second. If you have spare I/O capacity under a typical workload, increase the value. If a write-intensive workload saturates your I/O capacity, decrease the value, especially if you have a large buffer pool. For general I/O tuning advice, see Section 8.5.7, “Optimizing InnoDB Disk I/O”.

- innodb_max_dirty_pages_pct

Command-Line Format	--innodb_max_dirty_pages_pct=#	
Option-File Format	innodb_max_dirty_pages_pct	
System Variable Name	innodb_max_dirty_pages_pct	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values (<= 5.7.4)	
	Type	numeric
	Default	75
	Range	0 .. 99
	Permitted Values (>= 5.7.5)	

	Type	numeric
	Default	75
	Range	0 .. 99.99

InnoDB tries to flush data from the buffer pool so that the percentage of dirty pages does not exceed this value. The default value is 75.

The innodb_max_dirty_pages_pct setting establishes a target for flushing activity. It does not affect the rate of flushing. For information about managing the rate of flushing, see Controlling the Flushing Rate of Dirty Pages from the InnoDB Buffer Pool.

For additional information about this variable, see Improvements to Buffer Pool Flushing. For general I/O tuning advice, see Section 8.5.7, “Optimizing InnoDB Disk I/O”.

- innodb_max_dirty_pages_pct_lwm

Command-Line Format	--innodb_max_dirty_pages_pct_lwm=#	
Option-File Format	innodb_max_dirty_pages_pct_lwm	
System Variable Name	innodb_max_dirty_pages_pct_lwm	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values (<= 5.7.4)	
	Type	numeric
	Default	0
	Range	0 .. 99
	Permitted Values (>= 5.7.5)	
	Type	numeric
	Default	0
	Range	0 .. 99.99

Low water mark representing percentage of dirty pages where preflushing is enabled to control the dirty page ratio. The default of 0 disables the pre-flushing behavior entirely. For additional information about this variable, see Improvements to Buffer Pool Flushing.

- innodb_max_purge_lag

Command-Line Format	--innodb_max_purge_lag=#	
Option-File Format	innodb_max_purge_lag	
System Variable Name	innodb_max_purge_lag	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	numeric
	Default	0
	Range	0 .. 4294967295

This variable controls how to delay INSERT, UPDATE, and DELETE operations when purge operations are lagging (see Section 14.2.2.12, "InnoDB Multi-Versioning"). The default value is 0 (no delays).

The InnoDB transaction system maintains a list of transactions that have index records delete-marked by UPDATE or DELETE operations. The length of this list represents the *purge_lag* value. When *purge_lag* exceeds innodb_max_purge_lag, each INSERT, UPDATE, and DELETE operation is delayed.

To prevent excessive delays in extreme situations where *purge_lag* becomes huge, you can put a cap on the amount of delay by setting the innodb_max_purge_lag_delay configuration option. The delay is computed at the beginning of a purge batch.

A typical setting for a problematic workload might be 1 million, assuming that transactions are small, only 100 bytes in size, and it is permissible to have 100MB of unpurged InnoDB table rows.

The lag value is displayed as the history list length in the TRANSACTIONS section of InnoDB Monitor output. For example, if the output includes the following lines, the lag value is 20:

```
------------
TRANSACTIONS
------------
Trx id counter 0 290328385
Purge done for trx's n:o < 0 290315608 undo n:o < 0 17
History list length 20
```

For general I/O tuning advice, see Section 8.5.7, "Optimizing InnoDB Disk I/O".

- innodb_max_purge_lag_delay

Command-Line Format	--innodb_max_purge_lag_delay=#	
Option-File Format	innodb_max_purge_lag_delay	
System Variable Name	innodb_max_purge_lag_delay	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	numeric
	Default	0
	Min Value	0

Specifies the maximum delay in milliseconds for the delay imposed by the innodb_max_purge_lag configuration option. Any non-zero value represents an upper limit on the delay period computed from the formula based on the value of innodb_max_purge_lag. The default of zero means that there is no upper limit imposed on the delay interval.

For general I/O tuning advice, see Section 8.5.7, "Optimizing InnoDB Disk I/O".

- innodb_monitor_disable

Command-Line Format	--innodb_monitor_disable=[counter|module|pattern|all]
Option-File Format	innodb_monitor_disable

System Variable Name	innodb_monitor_disable	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	string

Turns off one or more counters in the INFORMATION_SCHEMA.INNODB_METRICS table. For usage information, see Section 19.30.19, “The INFORMATION_SCHEMA INNODB_METRICS Table”.

- innodb_monitor_enable

Command-Line Format	--innodb_monitor_enable=[counter|module|pattern|all]	
Option-File Format	innodb_monitor_enable	
System Variable Name	innodb_monitor_enable	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	string

Turns on one or more counters in the INFORMATION_SCHEMA.INNODB_METRICS table. For usage information, see Section 19.30.19, “The INFORMATION_SCHEMA INNODB_METRICS Table”.

- innodb_monitor_reset

Command-Line Format	--innodb_monitor_reset=[counter|module|pattern|all]	
Option-File Format	innodb_monitor_reset	
System Variable Name	innodb_monitor_reset	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	string

Resets to zero the count value for one or more counters in the INFORMATION_SCHEMA.INNODB_METRICS table. For usage information, see Section 19.30.19, “The INFORMATION_SCHEMA INNODB_METRICS Table”.

- innodb_monitor_reset_all

Command-Line Format	--innodb_monitor_reset_all=[counter|module|pattern|all]	
Option-File Format	innodb_monitor_reset_all	
System Variable Name	innodb_monitor_reset_all	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	string

Resets all values (minimum, maximum, and so on) for one or more counters in the INFORMATION_SCHEMA.INNODB_METRICS table. For usage information, see Section 19.30.19, “The INFORMATION_SCHEMA INNODB_METRICS Table”.

- innodb_old_blocks_pct

Command-Line Format	--innodb_old_blocks_pct=#	
Option-File Format	innodb_old_blocks_pct	
System Variable Name	innodb_old_blocks_pct	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	numeric
	Default	37
	Range	5 .. 95

Specifies the approximate percentage of the InnoDB buffer pool used for the old block sublist. The range of values is 5 to 95. The default value is 37 (that is, 3/8 of the pool). Often used in combination with innodb_old_blocks_time. See Making the Buffer Pool Scan Resistant for more information. See Section 8.9.1, “The InnoDB Buffer Pool” for information about buffer pool management, such as the LRU algorithm and eviction policies.

- innodb_old_blocks_time

Command-Line Format	--innodb_old_blocks_time=#	
Option-File Format	innodb_old_blocks_time	
System Variable Name	innodb_old_blocks_time	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	numeric
	Default	1000
	Range	0 .. 2**32-1

Non-zero values protect against the buffer pool being filled up by data that is referenced only for a brief period, such as during a full table scan. Increasing this value offers more protection against full table scans interfering with data cached in the buffer pool.

Specifies how long in milliseconds (ms) a block inserted into the old sublist must stay there after its first access before it can be moved to the new sublist. If the value is 0, a block inserted into the old sublist moves immediately to the new sublist the first time it is accessed, no matter how soon after insertion the access occurs. If the value is greater than 0, blocks remain in the old sublist until an access occurs at least that many ms after the first access. For example, a value of 1000 causes blocks to stay in the old sublist for 1 second after the first access before they become eligible to move to the new sublist.

The default value is 1000.

This variable is often used in combination with innodb_old_blocks_pct. See Making the Buffer Pool Scan Resistant for more information. See Section 8.9.1, "The InnoDB Buffer Pool" for information about buffer pool management, such as the LRU algorithm and eviction policies.

- innodb_online_alter_log_max_size

Command-Line Format	--innodb_online_alter_log_max_size=#	
Option-File Format	innodb_online_alter_log_max_size	
System Variable Name	innodb_online_alter_log_max_size	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	numeric
	Default	134217728
	Range	65536 .. 2**64-1

Specifies an upper limit on the size of the temporary log files used during online DDL operations for InnoDB tables. There is one such log file for each index being created or table being altered. This log file stores data inserted, updated, or deleted in the table during the DDL operation. The temporary log file is extended when needed by the value of innodb_sort_buffer_size, up to the maximum specified by innodb_online_alter_log_max_size. If any temporary log file exceeds the upper size limit, the ALTER TABLE operation fails and all uncommitted concurrent DML operations are rolled back. Thus, a large value for this option allows more DML to happen during an online DDL operation, but also causes a longer period at the end of the DDL operation when the table is locked to apply the data from the log.

- innodb_open_files

Command-Line Format	--innodb_open_files=#	
Option-File Format	innodb_open_files	
System Variable Name	innodb_open_files	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	numeric
	Default	-1 (autosized)
	Range	10 .. 4294967295

This variable is relevant only if you use multiple InnoDB tablespaces. It specifies the maximum number of .ibd files that MySQL can keep open at one time. The minimum value is 10. The default value is 300 if innodb_file_per_table is not enabled, and the higher of 300 and table_open_cache otherwise.

The file descriptors used for .ibd files are for InnoDB tables only. They are independent of those specified by the --open-files-limit server option, and do not affect the operation of the table cache. For general I/O tuning advice, see Section 8.5.7, "Optimizing InnoDB Disk I/O".

- innodb_optimize_fulltext_only

Command-Line Format	--innodb_optimize_fulltext_only=#	
Option-File Format	innodb_optimize_fulltext_only	
System Variable Name	innodb_optimize_fulltext_only	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	boolean
	Default	OFF

Changes the way the OPTIMIZE TABLE statement operates on InnoDB tables. Intended to be enabled temporarily, during maintenance operations for InnoDB tables with FULLTEXT indexes.

By default, OPTIMIZE TABLE reorganizes the data in the clustered index of the table. When this option is enabled, OPTIMIZE TABLE skips this reorganization of the table data, and instead processes the newly added, deleted, and updated token data for a FULLTEXT index, See FULLTEXT Indexes for more information about FULLTEXT indexes for InnoDB tables.

- innodb_page_cleaners

Introduced	5.7.4	
Command-Line Format	--innodb_page_cleaners=#	
Option-File Format	innodb_page_cleaners	
System Variable Name	innodb_page_cleaners	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	numeric
	Default	1
	Range	1 .. 64

The number of page_cleaner threads that flush dirty pages from buffer pool instances. The page_cleaner threads perform flush list and LRU flushing. A single page_cleaner thread was introduced in MySQL 5.6.2 to offload buffer pool flushing work from the InnoDB master thread. As of MySQL 5.7.4, InnoDB provides support for multiple page_cleaner threads. The default value of 1 maintains the pre-MySQL 5.7.4 configuration in which there is a single page_cleaner thread. When there are multiple page_cleaner threads, buffer pool flushing tasks for each buffer pool instance are dispatched to idle page_cleaner threads.

If your workload is write-IO bound (when flushing dirty pages from buffer pool instances to data files) and if your system hardware has available capacity, increasing the number of page_cleaner threads may help improve write-IO throughput.

- innodb_page_size

Command-Line Format	--innodb_page_size=#k
Option-File Format	innodb_page_size

System Variable Name	innodb_page_size	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	enumeration
	Default	16384
	Valid Values	4k
		8k
		16k
		4096
		8192
		16384

Specifies the page size for all InnoDB tablespaces in a MySQL instance. This value is set when the instance is created and remains constant afterwards. You can specify page size using the values 16k (the default), 8k, or 4k.

The default, with the largest page size, is appropriate for a wide range of workloads, particularly for queries involving table scans and DML operations involving bulk updates. Smaller page sizes might be more efficient for OLTP workloads involving many small writes, where contention can be an issue when a single page contains many rows. Smaller pages might also be efficient with SSD storage devices, which typically use small block sizes. Keeping the InnoDB page size close to the storage device block size minimizes the amount of unchanged data that is rewritten to disk. For general I/O tuning advice, see Section 8.5.7, “Optimizing InnoDB Disk I/O”.

- innodb_print_all_deadlocks

Command-Line Format	--innodb_print_all_deadlocks=#	
Option-File Format	innodb_print_all_deadlocks	
System Variable Name	innodb_print_all_deadlocks	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	boolean
	Default	OFF

When this option is enabled, information about all deadlocks in InnoDB user transactions is recorded in the mysqld error log. Otherwise, you see information about only the last deadlock, using the SHOW ENGINE INNODB STATUS command. An occasional InnoDB deadlock is not necessarily an issue, because InnoDB detects the condition immediately, and rolls back one of the transactions automatically. You might use this option to troubleshoot why deadlocks are happening if an application does not have appropriate error-handling logic to detect the rollback and retry its operation. A large number of deadlocks might indicate the need to restructure transactions that issue DML or SELECT ... FOR UPDATE statements for multiple tables, so that each transaction accesses the tables in the same order, thus avoiding the deadlock condition.

- innodb_purge_batch_size

Command-Line Format	`--innodb_purge_batch_size=#`	
Option-File Format	`innodb_purge_batch_size`	
System Variable Name	`innodb_purge_batch_size`	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	`numeric`
	Default	`300`
	Range	`1 .. 5000`

The granularity of changes, expressed in units of redo log records, that trigger a purge operation, flushing the changed buffer pool blocks to disk. This option is intended for tuning performance in combination with the setting `innodb_purge_threads=`*`n`*, and typical users do not need to modify it.

- `innodb_purge_threads`

Command-Line Format	`--innodb_purge_threads=#`	
Option-File Format	`innodb_purge_threads`	
System Variable Name	`innodb_purge_threads`	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	`numeric`
	Default	`1`
	Range	`1 .. 32`

The number of background threads devoted to the InnoDB purge operation. The default and minimum value of 1 signifies that the purge operation is always performed by background threads, never as part of the master thread. Non-zero values runs the purge operation in one or more background threads, which can reduce internal contention within InnoDB, improving scalability. Increasing the value to greater than 1 creates that many separate purge threads, which can improve efficiency on systems where DML operations are performed on multiple tables. The maximum is 32.

- `innodb_random_read_ahead`

Command-Line Format	`--innodb_random_read_ahead=#`	
Option-File Format	`innodb_random_read_ahead`	
System Variable Name	`innodb_random_read_ahead`	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	`boolean`
	Default	`OFF`

Enables the random read-ahead technique for optimizing InnoDB I/O. This is a setting that was originally on by default, then was removed in MySQL 5.5, and now is available but turned off by default. See Changes in the Read-Ahead Algorithm for details about the performance considerations for the different types of read-ahead requests. For general I/O tuning advice, see Section 8.5.7, “Optimizing InnoDB Disk I/O”.

- innodb_read_ahead_threshold

Command-Line Format	--innodb_read_ahead_threshold=#	
Option-File Format	innodb_read_ahead_threshold	
System Variable Name	innodb_read_ahead_threshold	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	numeric
	Default	56
	Range	0 .. 64

Controls the sensitivity of linear read-ahead that InnoDB uses to prefetch pages into the buffer pool. If InnoDB reads at least innodb_read_ahead_threshold pages sequentially from an extent (64 pages), it initiates an asynchronous read for the entire following extent. The permissible range of values is 0 to 64. A value of 0 disables read-ahead. For the default of 56, InnoDB must read at least 56 pages sequentially from an extent to initiate an asynchronous read for the following extent.

Knowing how many pages are read through this read-ahead mechanism, and how many of them are evicted from the buffer pool without ever being accessed, can be useful to help fine-tune the innodb_read_ahead_threshold parameter. As of MySQL 5.5, SHOW ENGINE INNODB STATUS output displays counter information from the Innodb_buffer_pool_read_ahead and Innodb_buffer_pool_read_ahead_evicted global status variables. These variables indicate the number of pages brought into the buffer pool by read-ahead requests, and the number of such pages evicted from the buffer pool without ever being accessed respectively. These counters provide global values since the last server restart.

SHOW ENGINE INNODB STATUS also shows the rate at which the read-ahead pages are read in and the rate at which such pages are evicted without being accessed. The per-second averages are based on the statistics collected since the last invocation of SHOW ENGINE INNODB STATUS and are displayed in the BUFFER POOL AND MEMORY section of the output.

See Changes in the Read-Ahead Algorithm for more information. For general I/O tuning advice, see Section 8.5.7, “Optimizing InnoDB Disk I/O”.

- innodb_read_io_threads

Command-Line Format	--innodb_read_io_threads=#
Option-File Format	innodb_read_io_threads
System Variable Name	innodb_read_io_threads
Variable Scope	Global
Dynamic Variable	No
	Permitted Values

	Type	numeric
	Default	4
	Range	1 .. 64

The number of I/O threads for read operations in InnoDB. The default value is 4. Its counterpart for write threads is innodb_write_io_threads. See Multiple Background InnoDB I/O Threads for more information. For general I/O tuning advice, see Section 8.5.7, "Optimizing InnoDB Disk I/O".

> **Note**
>
> On Linux systems, running multiple MySQL servers (typically more than 12) with default settings for innodb_read_io_threads, innodb_write_io_threads, and the Linux aio-max-nr setting can exceed system limits. Ideally, increase the aio-max-nr setting; as a workaround, you might reduce the settings for one or both of the MySQL configuration options.

- innodb_read_only

Command-Line Format	--innodb_read_only=#	
Option-File Format	innodb_read_only	
System Variable Name	innodb_read_only	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	boolean
	Default	OFF

Starts the server in read-only mode. For distributing database applications or data sets on read-only media. Can also be used in data warehouses to share the same data directory between multiple instances. See Section 14.2.3.1, "Configuring InnoDB for Read-Only Operation" for usage instructions.

- innodb_replication_delay

Command-Line Format	--innodb_replication_delay=#	
Option-File Format	innodb_replication_delay	
System Variable Name	innodb_replication_delay	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	numeric
	Default	0
	Range	0 .. 4294967295

The replication thread delay (in ms) on a slave server if innodb_thread_concurrency is reached.

- innodb_rollback_on_timeout

Command-Line Format	--innodb_rollback_on_timeout

Option-File Format	innodb_rollback_on_timeout	
System Variable Name	innodb_rollback_on_timeout	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	boolean
	Default	OFF

In MySQL 5.7, InnoDB rolls back only the last statement on a transaction timeout by default. If --innodb_rollback_on_timeout is specified, a transaction timeout causes InnoDB to abort and roll back the entire transaction (the same behavior as in MySQL 4.1).

- innodb_rollback_segments

Command-Line Format	--innodb_rollback_segments=#	
Option-File Format	innodb_rollback_segments	
System Variable Name	innodb_rollback_segments	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	numeric
	Default	128
	Range	1 .. 128

Defines how many of the rollback segments in the system tablespace that InnoDB uses within a transaction. This setting, while still valid, is replaced by innodb_undo_logs.

- innodb_sort_buffer_size

Command-Line Format	--innodb_sort_buffer_size=#	
Option-File Format	innodb_sort_buffer_size	
System Variable Name	innodb_sort_buffer_size	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	numeric
	Default	1048576
	Range	65536 .. 67108864

Specifies the size of sort buffers used for sorting data during creation of an InnoDB index. The size specified defines the amount of data filled in memory for an internal sort and written out to disk, which can be referred to as a “run”. During the merge phase, pairs of buffers of the specified size are “read in” and merged. The larger the setting, the fewer “runs” and merges there are, which is important to understand from a tuning perspective.

This sort area is only used for merge sorts during index creation, not during later index maintenance operations. Buffers are deallocated when index creation completes.

The value of this option also controls the amount by which the temporary log file is extended, to record concurrent DML during online DDL operations.

Before this setting was made configurable, the size was hardcoded to 1048576 bytes (1MB), and that value remains the default.

During an ALTER TABLE or CREATE TABLE statement that creates an index, 3 buffers are allocated, each with a size defined by this option. Additionally, auxiliary pointers are allocated to rows in the sort buffer so that the sort can run on pointers (as opposed to moving rows during the sort operation).

For a typical sort operation, a formula such as this can be used to estimate memory consumption:

```
(6 /*FTS_NUM_AUX_INDEX*/ *
(3*@@global.innodb_sort_buffer_size) + 2 * (
@@global.innodb_sort_buffer_size/dict_index_get_min_size(index)*/)
* 8 /*64-bit sizeof *buf->tuples*/")
```

“@@global.innodb_sort_buffer_size/dict_index_get_min_size(index)” indicates the maximum tuples held. “2 * (@@global.innodb_sort_buffer_size/ *dict_index_get_min_size(index)*/) * 8 /*64-bit size of *buf->tuples*/ indicates auxiliary pointers allocated.”.

Note

For 32-bit, multiply by 4 instead of 8.

For parallel sorts on an index, multiply by the innodb_ft_sort_pll_degree setting:

```
(6 /*FTS_NUM_AUX_INDEX*/ @@global.innodb_ft_sort_pll_degree)
```

- innodb_spin_wait_delay

Command-Line Format	--innodb_spin_wait_delay=#	
Option-File Format	innodb_spin_wait_delay	
System Variable Name	innodb_spin_wait_delay	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	numeric
	Default	6
	Range	0 .. 4294967295

The maximum delay between polls for a spin lock. The low-level implementation of this mechanism varies depending on the combination of hardware and operating system, so the delay does not correspond to a fixed time interval. The default value is 6. See Control of Spin Lock Polling for more information.

- innodb_stats_auto_recalc

Command-Line Format	--innodb_stats_auto_recalc=#	
Option-File Format	innodb_stats_auto_recalc	
System Variable Name	innodb_stats_auto_recalc	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	boolean
	Default	ON

Causes InnoDB to automatically recalculate persistent statistics after the data in a table is changed substantially. The threshold value is currently 10% of the rows in the table. This setting applies to tables created when the innodb_stats_persistent option is enabled, or where the clause STATS_PERSISTENT=1 is enabled by a CREATE TABLE or ALTER TABLE statement. The amount of data sampled to produce the statistics is controlled by the innodb_stats_persistent_sample_pages configuration option.

For additional information about innodb_stats_auto_recalc, see Persistent Optimizer Statistics for InnoDB Tables.

- innodb_stats_method

Command-Line Format	--innodb_stats_method=name	
Option-File Format	innodb_stats_method	
System Variable Name	innodb_stats_method	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	enumeration
	Default	nulls_equal
	Valid Values	nulls_equal
		nulls_unequal
		nulls_ignored

How the server treats NULL values when collecting statistics about the distribution of index values for InnoDB tables. This variable has three possible values, nulls_equal, nulls_unequal, and nulls_ignored. For nulls_equal, all NULL index values are considered equal and form a single value group that has a size equal to the number of NULL values. For nulls_unequal, NULL values are considered unequal, and each NULL forms a distinct value group of size 1. For nulls_ignored, NULL values are ignored.

The method that is used for generating table statistics influences how the optimizer chooses indexes for query execution, as described in Section 8.3.7, “InnoDB and MyISAM Index Statistics Collection”.

- innodb_stats_on_metadata

Command-Line Format	--innodb_stats_on_metadata

Option-File Format	innodb_stats_on_metadata	
System Variable Name	innodb_stats_on_metadata	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	boolean
	Default	OFF

When this variable is enabled, InnoDB updates statistics when metadata statements such as SHOW TABLE STATUS or SHOW INDEX are run, or when accessing the INFORMATION_SCHEMA tables TABLES or STATISTICS. (These updates are similar to what happens for ANALYZE TABLE.) When disabled, InnoDB does not update statistics during these operations. Leaving this setting disabled can improve access speed for schemas that have a large number of tables or indexes. It can also improve the stability of execution plans for queries that involve InnoDB tables.

To change the setting, issue the statement SET GLOBAL innodb_stats_on_metadata=*mode*, where *mode* is either ON or OFF (or 1 or 0). Changing this setting requires the SUPER privilege and immediately affects the operation of all connections.

This variable is disabled by default.

- innodb_stats_persistent

Command-Line Format	--innodb_stats_persistent=setting	
Option-File Format	innodb_stats_persistent	
System Variable Name	innodb_stats_persistent	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	boolean
	Default	ON
	Valid Values	OFF
		ON
		0
		1

Specifies whether the InnoDB index statistics produced by the ANALYZE TABLE command are stored on disk, remaining consistent until a subsequent ANALYZE TABLE. Otherwise, the statistics are recalculated more frequently, such as at each server restart, which can lead to variations in query execution plans. This setting is stored with each table when the table is created. You can specify or change it through SQL with the STATS_PERSISTENT clause of the CREATE TABLE and ALTER TABLE commands.

- innodb_stats_persistent_sample_pages

Command-Line Format	--innodb_stats_persistent_sample_pages=#
Option-File Format	innodb_stats_persistent_sample_pages

System Variable Name	innodb_stats_persistent_sample_pages	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	numeric
	Default	20

The number of index pages to sample when estimating cardinality and other statistics for an indexed column, such as those calculated by ANALYZE TABLE. Increasing the value improves the accuracy of index statistics, which can improve the query execution plan, at the expense of increased I/O during the execution of ANALYZE TABLE for an InnoDB table.

This option only applies when the innodb_stats_persistent setting is turned on for a table; when that option is turned off for a table, the innodb_stats_transient_sample_pages setting applies instead.

- innodb_stats_sample_pages

Deprecated	5.6.3	
Command-Line Format	--innodb_stats_sample_pages=#	
Option-File Format	innodb_stats_sample_pages	
System Variable Name	innodb_stats_sample_pages	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	numeric
	Default	8
	Range	1 .. 2**64-1

Deprecated, use innodb_stats_transient_sample_pages instead.

- innodb_stats_transient_sample_pages

Command-Line Format	--innodb_stats_transient_sample_pages=#	
Option-File Format	innodb_stats_transient_sample_pages	
System Variable Name	innodb_stats_transient_sample_pages	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	numeric
	Default	8

The number of index pages to sample when estimating cardinality and other statistics for an indexed column, such as those calculated by ANALYZE TABLE. The default value is 8. Increasing the value improves the accuracy of index statistics, which can improve the query execution plan, at the expense of increased I/O when opening an InnoDB table or recalculating statistics.

This option only applies when the innodb_stats_persistent setting is turned off for a table; when this option is turned on for a table, the innodb_stats_persistent_sample_pages setting applies instead. Takes the place of the innodb_stats_sample_pages option. See Section 14.2.12.5, “Controlling Optimizer Statistics Estimation” for more information.

- innodb_status_output

Introduced	5.7.4	
Command-Line Format	--innodb_status_output	
Option-File Format	innodb_status_output	
System Variable Name	innodb_status_output	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	boolean
	Default	OFF

Used to enable or disable periodic output for the standard InnoDB Monitor. Also used in combination with innodb_status_output_locks to enable or disable periodic output for the InnoDB Lock Monitor. See Section 14.2.12.4, “InnoDB Monitors” for additional information.

- innodb_status_output_locks

Introduced	5.7.4	
Command-Line Format	--innodb_status_output_locks	
Option-File Format	innodb_status_output_locks	
System Variable Name	innodb_status_output_locks	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	boolean
	Default	OFF

Used to enable or disable periodic output for the InnoDB Lock Monitor. Must be used in combination with innodb_status_output. See Section 14.2.12.4, “InnoDB Monitors” for additional information.

- innodb_strict_mode

Command-Line Format	--innodb_strict_mode=#	
Option-File Format	innodb_strict_mode	
System Variable Name	innodb_strict_mode	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	boolean

	Default	OFF

When innodb_strict_mode is ON, InnoDB returns errors rather than warnings for certain conditions. The default value is OFF.

Strict mode helps guard against ignored typos and syntax errors in SQL, or other unintended consequences of various combinations of operational modes and SQL statements. When innodb_strict_mode is ON, InnoDB raises error conditions in certain cases, rather than issuing a warning and processing the specified statement (perhaps with unintended behavior). This is analogous to sql_mode in MySQL, which controls what SQL syntax MySQL accepts, and determines whether it silently ignores errors, or validates input syntax and data values.

The innodb_strict_mode setting affects the handling of syntax errors for CREATE TABLE, ALTER TABLE and CREATE INDEX statements. innodb_strict_mode also enables a record size check, so that an INSERT or UPDATE never fails due to the record being too large for the selected page size.

Oracle recommends enabling innodb_strict_mode when using ROW_FORMAT and KEY_BLOCK_SIZE clauses on CREATE TABLE, ALTER TABLE, and CREATE INDEX statements. When innodb_strict_mode is OFF, InnoDB ignores conflicting clauses and creates the table or index, with only a warning in the message log. The resulting table might have different behavior than you intended, such as having no compression when you tried to create a compressed table. When innodb_strict_mode is ON, such problems generate an immediate error and the table or index is not created, avoiding a troubleshooting session later.

You can turn innodb_strict_mode ON or OFF on the command line when you start mysqld, or in the configuration file my.cnf or my.ini. You can also enable or disable innodb_strict_mode at runtime with the statement SET [GLOBAL|SESSION] innodb_strict_mode=*mode*, where *mode* is either ON or OFF. Changing the GLOBAL setting requires the SUPER privilege and affects the operation of all clients that subsequently connect. Any client can change the SESSION setting for innodb_strict_mode, and the setting affects only that client.

- innodb_support_xa

Command-Line Format	--innodb_support_xa	
Option-File Format	innodb_support_xa	
System Variable Name	innodb_support_xa	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	boolean
	Default	TRUE

Enables InnoDB support for two-phase commit in XA transactions, causing an extra disk flush for transaction preparation. This setting is the default. The XA mechanism is used internally and is essential for any server that has its binary log turned on and is accepting changes to its data from more than one thread. If you turn it off, transactions can be written to the binary log in a different order from the one in which the live database is committing them. This can produce different data when the binary log is replayed in disaster recovery or on a replication slave. Do not turn it off on a replication master server unless you have an unusual setup where only one thread is able to change data.

For a server that is accepting data changes from only one thread, it is safe and recommended to turn off this option to improve performance for InnoDB tables. For example, you can turn it off on replication slaves where only the replication SQL thread is changing data.

You can also turn off this option if you do not need it for safe binary logging or replication, and you also do not use an external XA transaction manager.

- innodb_sync_array_size

Command-Line Format	--innodb_sync_array_size=#	
Option-File Format	innodb_sync_array_size	
System Variable Name	innodb_sync_array_size	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	numeric
	Default	1
	Range	1 .. 1024

Splits an internal data structure used to coordinate threads, for higher concurrency in workloads with large numbers of waiting threads. This setting must be configured when the MySQL instance is starting up, and cannot be changed afterward. Increasing this option value is recommended for workloads that frequently produce a large number of waiting threads, typically greater than 768.

- innodb_sync_spin_loops

Command-Line Format	--innodb_sync_spin_loops=#	
Option-File Format	innodb_sync_spin_loops	
System Variable Name	innodb_sync_spin_loops	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	numeric
	Default	30
	Range	0 .. 4294967295

The number of times a thread waits for an InnoDB mutex to be freed before the thread is suspended. The default value is 30.

- innodb_table_locks

Command-Line Format	--innodb_table_locks
Option-File Format	innodb_table_locks
System Variable Name	innodb_table_locks
Variable Scope	Global, Session
Dynamic Variable	Yes

	Permitted Values	
	Type	boolean
	Default	TRUE

If autocommit = 0, InnoDB honors LOCK TABLES; MySQL does not return from LOCK TABLES ... WRITE until all other threads have released all their locks to the table. The default value of innodb_table_locks is 1, which means that LOCK TABLES causes InnoDB to lock a table internally if autocommit = 0.

In MySQL 5.7, innodb_table_locks = 0 has no effect for tables locked explicitly with LOCK TABLES ... WRITE. It does have an effect for tables locked for read or write by LOCK TABLES ... WRITE implicitly (for example, through triggers) or by LOCK TABLES ... READ.

- innodb_temp_data_file_path

Introduced	5.7.1	
Command-Line Format	--innodb_temp_data_file_path=file	
Option-File Format	innodb_temp_data_file_path	
System Variable Name	innodb_temp_data_file_path	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	string
	Default	ibtmp1:12M:autoextend

The paths to individual InnoDB temporary tablespace data files and their sizes. The full directory path to each data file is formed by concatenating innodb_data_home_dir to each path specified here. The file sizes are specified in KB, MB, or GB (1024MB) by appending K, M, or G to the size value. The sum of the sizes of the files must be at least slightly larger than 12MB. If you do not specify innodb_temp_data_file_path, the default behavior is to create a single auto-extending temporary tablespace data file, slightly larger than 12MB, named ibtmp1. The size limit of individual files is determined by your operating system. You can set the file size to more than 4GB on those operating systems that support big files. Use of raw disk partitions as temporary data files is not supported.

The name of a InnoDB temporary tablespace data file cannot be the same as the name of a InnoDB data file. Any inability or error creating a temporary tablespace data file is treated as fatal and server startup will be refused. The temporary tablespace has a dynamically generated space-id, which can change on each server restart.

- innodb_thread_concurrency

Command-Line Format	--innodb_thread_concurrency=#	
Option-File Format	innodb_thread_concurrency	
System Variable Name	innodb_thread_concurrency	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	numeric

	Default	0
	Range	0 .. 1000

InnoDB tries to keep the number of operating system threads concurrently inside InnoDB less than or equal to the limit given by this variable. Once the number of threads reaches this limit, additional threads are placed into a wait state within a FIFO queue for execution. Threads waiting for locks are not counted in the number of concurrently executing threads.

The correct value for this variable is dependent on environment and workload. Try a range of different values to determine what value works for your applications. A recommended value is 2 times the number of CPUs plus the number of disks.

The range of this variable is 0 to 1000. A value of 0 (the default) is interpreted as infinite concurrency (no concurrency checking). Disabling thread concurrency checking enables InnoDB to create as many threads as it needs. A value of 0 also disables the queries inside InnoDB and queries in queue counters in the ROW OPERATIONS section of SHOW ENGINE INNODB STATUS output.

- innodb_thread_sleep_delay

Command-Line Format	--innodb_thread_sleep_delay=#	
Option-File Format	innodb_thread_sleep_delay	
System Variable Name	innodb_thread_sleep_delay	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values (<= 5.7.3)	
	Platform Bit Size	32
	Type	numeric
	Default	10000
	Range	0 .. 4294967295
	Permitted Values (<= 5.7.3)	
	Platform Bit Size	64
	Type	numeric
	Default	10000
	Range	0 .. 18446744073709551615
	Permitted Values (>= 5.7.4)	
	Type	numeric
	Default	10000
	Range	0 .. 1000000

How long InnoDB threads sleep before joining the InnoDB queue, in microseconds. The default value is 10,000. A value of 0 disables sleep. In MySQL 5.6.3 and higher, you can set the configuration option innodb_adaptive_max_sleep_delay to the highest value you would allow for innodb_thread_sleep_delay, and InnoDB automatically adjusts innodb_thread_sleep_delay up or down depending on the current thread-scheduling activity. This dynamic adjustment helps the thread scheduling mechanism to work smoothly during times when the system is lightly loaded and when it is operating near full capacity.

- innodb_undo_directory

Command-Line Format	--innodb_undo_directory=name	
Option-File Format	innodb_undo_directory	
System Variable Name	innodb_undo_directory	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	directory name
	Default	.

The relative or absolute directory path where InnoDB creates separate tablespaces for the undo logs. Typically used to place those logs on a different storage device. Used in conjunction with innodb_undo_logs and innodb_undo_tablespaces, which determine the disk layout of the undo logs outside the system tablespace. Its default value of . represents the same directory where InnoDB creates its other log files by default.

- innodb_undo_logs

Command-Line Format	--innodb_undo_logs=#	
Option-File Format	innodb_undo_logs	
System Variable Name	innodb_undo_logs	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	numeric
	Default	128
	Range	0 .. 128

Defines how many of the rollback segments in the system tablespace that InnoDB uses within a transaction. This setting is appropriate for tuning performance if you observe mutex contention related to the undo logs. Replaces the innodb_rollback_segments setting. For the total number of available undo logs, rather than the number of active ones, see the Innodb_available_undo_logs status variable.

Although you can increase or decrease how many rollback segments are used within a transaction, the number of rollback segments physically present in the system never decreases. Thus you might start with a low value for this parameter and gradually increase it, to avoid allocating rollback segments that are not needed later. If innodb_undo_logs is not set, it defaults to the maximum value of 128. For information about managing rollback segments, see Section 14.2.2.12, “InnoDB Multi-Versioning”.

- innodb_undo_tablespaces

Command-Line Format	--innodb_undo_tablespaces=#
Option-File Format	innodb_undo_tablespaces

System Variable Name	innodb_undo_tablespaces	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	numeric
	Default	0
	Range	0 .. 126

The number of tablespace files that the undo logs are divided between, when you use a non-zero innodb_undo_logs setting. By default, all the undo logs are part of the system tablespace and the system tablespace will always contain one undo tablespace in addition to those configured by innodb_undo_tablespaces. Because the undo logs can become large during long-running transactions, splitting the undo logs between multiple tablespaces reduces the maximum size of any one tablespace. The tablespace files are created in the location defined by innodb_undo_directory, with names of the form undo*N*, where *N* is a sequential series of integers, including leading zeros. The default size of undo tablespaces files is 10M. The number of innodb_undo_tablespaces must be set prior to initializing InnoDB. Attempting to restart InnoDB after changing the number of innodb_undo_tablespaces will result in a failed start with an error stating that InnoDB did not find the expected number of undo tablespaces.

- innodb_use_native_aio

Command-Line Format	--innodb_use_native_aio=#	
Option-File Format	innodb_use_native_aio	
System Variable Name	innodb_use_native_aio	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	boolean
	Default	ON

Specifies whether to use the Linux asynchronous I/O subsystem. This variable applies to Linux systems only, and cannot be changed while the server is running.

Normally, you do not need to touch this option, because it is enabled by default. If a problem with the asynchronous I/O subsystem in the OS prevents InnoDB from starting, start the server with this variable disabled (use innodb_use_native_aio=0 in the option file). This option could also be turned off automatically during startup, if InnoDB detects a potential problem such as a combination of tmpdir location, tmpfs filesystem, and Linux kernel that that does not support AIO on tmpfs.

- innodb_use_sys_malloc

Deprecated	5.6.3
Removed	5.7.4
Command-Line Format	--innodb_use_sys_malloc=#
Option-File Format	innodb_use_sys_malloc
System Variable Name	innodb_use_sys_malloc

Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	boolean
	Default	ON

Whether InnoDB uses the operating system memory allocator (ON) or its own (OFF). The default value is ON. See Using Operating System Memory Allocators for more information.

innodb_use_sys_malloc was deprecated in MySQL 5.6.3 and removed in MySQL 5.7.4.

- innodb_version

  The InnoDB version number. In 5.7, the separate numbering for InnoDB does not apply and this value is the same as for the version variable.

- innodb_write_io_threads

Command-Line Format	--innodb_write_io_threads=#	
Option-File Format	innodb_write_io_threads	
System Variable Name	innodb_write_io_threads	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	numeric
	Default	4
	Range	1 .. 64

The number of I/O threads for write operations in InnoDB. The default value is 4. Its counterpart for read threads is innodb_read_io_threads. See Multiple Background InnoDB I/O Threads for more information. For general I/O tuning advice, see Section 8.5.7, “Optimizing InnoDB Disk I/O”.

Note

On Linux systems, running multiple MySQL servers (typically more than 12) with default settings for innodb_read_io_threads, innodb_write_io_threads, and the Linux aio-max-nr setting can exceed system limits. Ideally, increase the aio-max-nr setting; as a workaround, you might reduce the settings for one or both of the MySQL configuration options.

You should also take into consideration the value of sync_binlog, which controls synchronization of the binary log to disk.

For general I/O tuning advice, see Section 8.5.7, “Optimizing InnoDB Disk I/O”.

14.2.13.1 Changes to InnoDB Startup Options and System Variables

New Parameters

The following InnoDB configuration parameters were added in 5.7. See Section 14.2.13, “InnoDB Startup Options and System Variables” for parameter descriptions and the specific release in which parameters were added.

- innodb_buffer_pool_dump_pct
- innodb_ft_result_cache_limit
- innodb_ft_total_cache_size
- innodb_log_write_ahead_size
- innodb_page_cleaners
- innodb_status_output
- innodb_status_output_locks
- innodb_temp_data_file_path

Removed Parameters

The following InnoDB configuration parameters were removed in 5.7. See Section 14.2.13, “InnoDB Startup Options and System Variables” for parameter descriptions and the specific release in which parameters were removed.

- innodb_additional_mem_pool_size
- innodb_use_sys_malloc

14.2.14 InnoDB Backup and Recovery

The key to safe database management is making regular backups. Depending on your data volume, number of MySQL servers, and database workload, you can use these techniques, alone or in combination: hot backup with MySQL Enterprise Backup; cold backup by copying files while the MySQL server is shut down; physical backup for fast operation (especially for restore); logical backup with mysqldump for smaller data volumes or to record the structure of schema objects.

Hot Backups

The mysqlbackup command, part of the MySQL Enterprise Backup component, lets you back up a running MySQL instance, including InnoDB and MyISAM tables, with minimal disruption to operations while producing a consistent snapshot of the database. When mysqlbackup is copying InnoDB tables, reads and writes to both InnoDB and MyISAM tables can continue. During the copying of MyISAM tables, reads (but not writes) to those tables are permitted. MySQL Enterprise Backup can also create compressed backup files, and back up subsets of tables and databases. In conjunction with MySQL’s binary log, users can perform point-in-time recovery. MySQL Enterprise Backup is part of the MySQL Enterprise subscription. For more details, see Section 23.2, “MySQL Enterprise Backup”.

Cold Backups

If you can shut down your MySQL server, you can make a binary backup that consists of all files used by InnoDB to manage its tables. Use the following procedure:

1. Do a slow shutdown of the MySQL server and make sure that it stops without errors.
2. Copy all InnoDB data files (ibdata files and .ibd files) into a safe place.

3. Copy all the .frm files for InnoDB tables to a safe place.

4. Copy all InnoDB log files (ib_logfile files) to a safe place.

5. Copy your my.cnf configuration file or files to a safe place.

Alternative Backup Types

In addition to making binary backups as just described, regularly make dumps of your tables with mysqldump. A binary file might be corrupted without you noticing it. Dumped tables are stored into text files that are human-readable, so spotting table corruption becomes easier. Also, because the format is simpler, the chance for serious data corruption is smaller. mysqldump also has a --single-transaction option for making a consistent snapshot without locking out other clients. See Section 7.3.1, “Establishing a Backup Policy”.

Replication works with InnoDB tables, so you can use MySQL replication capabilities to keep a copy of your database at database sites requiring high availability.

Performing Recovery

To recover your InnoDB database to the present from the time at which the binary backup was made, you must run your MySQL server with binary logging turned on, even before taking the backup. To achieve point-in-time recovery after restoring a backup, you can apply changes from the binary log that occurred after the backup was made. See Section 7.5, “Point-in-Time (Incremental) Recovery Using the Binary Log”.

To recover from a crash of your MySQL server, the only requirement is to restart it. InnoDB automatically checks the logs and performs a roll-forward of the database to the present. InnoDB automatically rolls back uncommitted transactions that were present at the time of the crash. During recovery, mysqld displays output something like this:

```
InnoDB: Database was not shut down normally.
InnoDB: Starting recovery from log files...
InnoDB: Starting log scan based on checkpoint at
InnoDB: log sequence number 0 13674004
InnoDB: Doing recovery: scanned up to log sequence number 0 13739520
InnoDB: Doing recovery: scanned up to log sequence number 0 13805056
InnoDB: Doing recovery: scanned up to log sequence number 0 13870592
InnoDB: Doing recovery: scanned up to log sequence number 0 13936128
...
InnoDB: Doing recovery: scanned up to log sequence number 0 20555264
InnoDB: Doing recovery: scanned up to log sequence number 0 20620800
InnoDB: Doing recovery: scanned up to log sequence number 0 20664692
InnoDB: 1 uncommitted transaction(s) which must be rolled back
InnoDB: Starting rollback of uncommitted transactions
InnoDB: Rolling back trx no 16745
InnoDB: Rolling back of trx no 16745 completed
InnoDB: Rollback of uncommitted transactions completed
InnoDB: Starting an apply batch of log records to the database...
InnoDB: Apply batch completed
InnoDB: Started
mysqld: ready for connections
```

If your database becomes corrupted or disk failure occurs, you must perform the recovery using a backup. In the case of corruption, first find a backup that is not corrupted. After restoring the base backup, do a point-in-time recovery from the binary log files using mysqlbinlog and mysql to restore the changes that occurred after the backup was made.

In some cases of database corruption, it is enough just to dump, drop, and re-create one or a few corrupt tables. You can use the CHECK TABLE SQL statement to check whether a table is corrupt, although

CHECK TABLE naturally cannot detect every possible kind of corruption. You can use the Tablespace Monitor to check the integrity of the file space management inside the tablespace files.

In some cases, apparent database page corruption is actually due to the operating system corrupting its own file cache, and the data on disk may be okay. It is best first to try restarting your computer. Doing so may eliminate errors that appeared to be database page corruption. If MySQL still has trouble starting because of InnoDB consistency problems, see Section 14.2.17.2, "Starting InnoDB on a Corrupted Database" for steps to start the instance in a diagnostic mode where you can dump the data.

14.2.14.1 The InnoDB Recovery Process

InnoDB crash recovery consists of several steps:

The first step, applying the redo log, is performed during initialization, before accepting any connections. If all changes were flushed from the buffer pool to the tablespaces (ibdata* and *.ibd files) at the time of the shutdown or crash, the redo log application can be skipped. If the redo log files are missing at startup, InnoDB skips the redo log application.

Removing redo logs to speed up the recovery process is not recommended, even if some data loss is acceptable. Removing redo logs should only ever be considered an option after a clean shutdown is performed, with innodb_fast_shutdown set to 0 or 1.

The remaining steps after redo log application do not depend on the redo log (other than for logging the writes) and are performed in parallel with normal processing. These include:

- Rolling back incomplete transactions: Any transactions that were active at the time of crash or fast shutdown. The time it takes to roll back an incomplete transaction can be three or four times the amount of time a transaction is active before it is interrupted, depending on server load.

  You cannot cancel transactions that are in the process of being rolled back. In extreme cases, when rolling back transactions is expected to take a long time, it may be faster to start InnoDB with an innodb_force_recovery setting of 3 or greater.

- Insert buffer merge: Applying changes from the insert buffer (part of the system tablespace) to leaf pages of secondary indexes, as the index pages are read to the buffer pool.

- Purge: Deleting delete-marked records that are no longer visible for any active transaction.

Of these, only rollback of incomplete transactions is special to crash recovery. The insert buffer merge and the purge are performed during normal processing.

In most situations, even if the MySQL server was killed unexpectedly in the middle of heavy activity, the recovery process happens automatically and no action is needed from the DBA. If a hardware failure or severe system error corrupted InnoDB data, MySQL might refuse to start. In that case, see Section 14.2.17.2, "Starting InnoDB on a Corrupted Database" for the steps to troubleshoot such an issue.

14.2.15 InnoDB and MySQL Replication

MySQL replication works for InnoDB tables as it does for MyISAM tables. It is also possible to use replication in a way where the storage engine on the slave is not the same as the original storage engine on the master. For example, you can replicate modifications to an InnoDB table on the master to a MyISAM table on the slave.

To set up a new slave for a master, make a copy of the InnoDB tablespace and the log files, as well as the .frm files of the InnoDB tables, and move the copies to the slave. If the innodb_file_per_table option is enabled, copy the .ibd files as well. For the proper procedure to do this, see Section 14.2.14, "InnoDB Backup and Recovery".

To make a new slave without taking down the master or an existing slave, use the MySQL Enterprise Backup product. If you can shut down the master or an existing slave, take a cold backup of the InnoDB tablespaces and log files and use that to set up a slave.

Transactions that fail on the master do not affect replication at all. MySQL replication is based on the binary log where MySQL writes SQL statements that modify data. A transaction that fails (for example, because of a foreign key violation, or because it is rolled back) is not written to the binary log, so it is not sent to slaves. See Section 13.3.1, "START TRANSACTION, COMMIT, and ROLLBACK Syntax".

Replication and CASCADE. Cascading actions for InnoDB tables on the master are replicated on the slave *only* if the tables sharing the foreign key relation use InnoDB on both the master and slave. This is true whether you are using statement-based or row-based replication. Suppose that you have started replication, and then create two tables on the master using the following CREATE TABLE statements:

```
CREATE TABLE fc1 (
    i INT PRIMARY KEY,
    j INT
) ENGINE = InnoDB;

CREATE TABLE fc2 (
    m INT PRIMARY KEY,
    n INT,
    FOREIGN KEY ni (n) REFERENCES fc1 (i)
        ON DELETE CASCADE
) ENGINE = InnoDB;
```

Suppose that the slave does not have InnoDB support enabled. If this is the case, then the tables on the slave are created, but they use the MyISAM storage engine, and the FOREIGN KEY option is ignored. Now we insert some rows into the tables on the master:

```
master> INSERT INTO fc1 VALUES (1, 1), (2, 2);
Query OK, 2 rows affected (0.09 sec)
Records: 2  Duplicates: 0  Warnings: 0

master> INSERT INTO fc2 VALUES (1, 1), (2, 2), (3, 1);
Query OK, 3 rows affected (0.19 sec)
Records: 3  Duplicates: 0  Warnings: 0
```

At this point, on both the master and the slave, table fc1 contains 2 rows, and table fc2 contains 3 rows, as shown here:

```
master> SELECT * FROM fc1;
+---+------+
| i | j    |
+---+------+
| 1 |    1 |
| 2 |    2 |
+---+------+
2 rows in set (0.00 sec)

master> SELECT * FROM fc2;
+---+------+
| m | n    |
+---+------+
| 1 |    1 |
| 2 |    2 |
| 3 |    1 |
+---+------+
3 rows in set (0.00 sec)

slave> SELECT * FROM fc1;
```

```
+---+------+
| i | j    |
+---+------+
| 1 |    1 |
| 2 |    2 |
+---+------+
2 rows in set (0.00 sec)

slave> SELECT * FROM fc2;
+---+------+
| m | n    |
+---+------+
| 1 |    1 |
| 2 |    2 |
| 3 |    1 |
+---+------+
3 rows in set (0.00 sec)
```

Now suppose that you perform the following DELETE statement on the master:

```
master> DELETE FROM fc1 WHERE i=1;
Query OK, 1 row affected (0.09 sec)
```

Due to the cascade, table fc2 on the master now contains only 1 row:

```
master> SELECT * FROM fc2;
+---+---+
| m | n |
+---+---+
| 2 | 2 |
+---+---+
1 row in set (0.00 sec)
```

However, the cascade does not propagate on the slave because on the slave the DELETE for fc1 deletes no rows from fc2. The slave's copy of fc2 still contains all of the rows that were originally inserted:

```
slave> SELECT * FROM fc2;
+---+---+
| m | n |
+---+---+
| 1 | 1 |
| 3 | 1 |
| 2 | 2 |
+---+---+
3 rows in set (0.00 sec)
```

This difference is due to the fact that the cascading deletes are handled internally by the InnoDB storage engine, which means that none of the changes are logged.

14.2.16 InnoDB Integration with memcached

The InnoDB memcached plugin delivers an integrated memcached daemon that automatically stores and retrieves data from InnoDB tables, turning the MySQL server into a fast “key-value store” for single-row insert, update, or delete operations. You can access the same InnoDB tables through SQL for convenience, complex queries, bulk operations, application compatibility, and other strengths of traditional database software.

This “NoSQL-style” interface uses the familiar memcached API to speed up database operations, letting InnoDB handle memory caching using its buffer pool mechanism. Data modified through memcached operations such as ADD, SET, INCR are stored to disk, using the familiar InnoDB mechanisms such as change buffering, the doublewrite buffer, and crash recovery. The combination of memcached simplicity and InnoDB durability provides users with the best of both worlds, as explained in Section 14.2.16.1, "Benefits of the InnoDB / memcached Combination". For architectural details about how the components fit together, see Section 14.2.16.2, "Architecture of InnoDB and memcached Integration".

14.2.16.1 Benefits of the InnoDB / memcached Combination

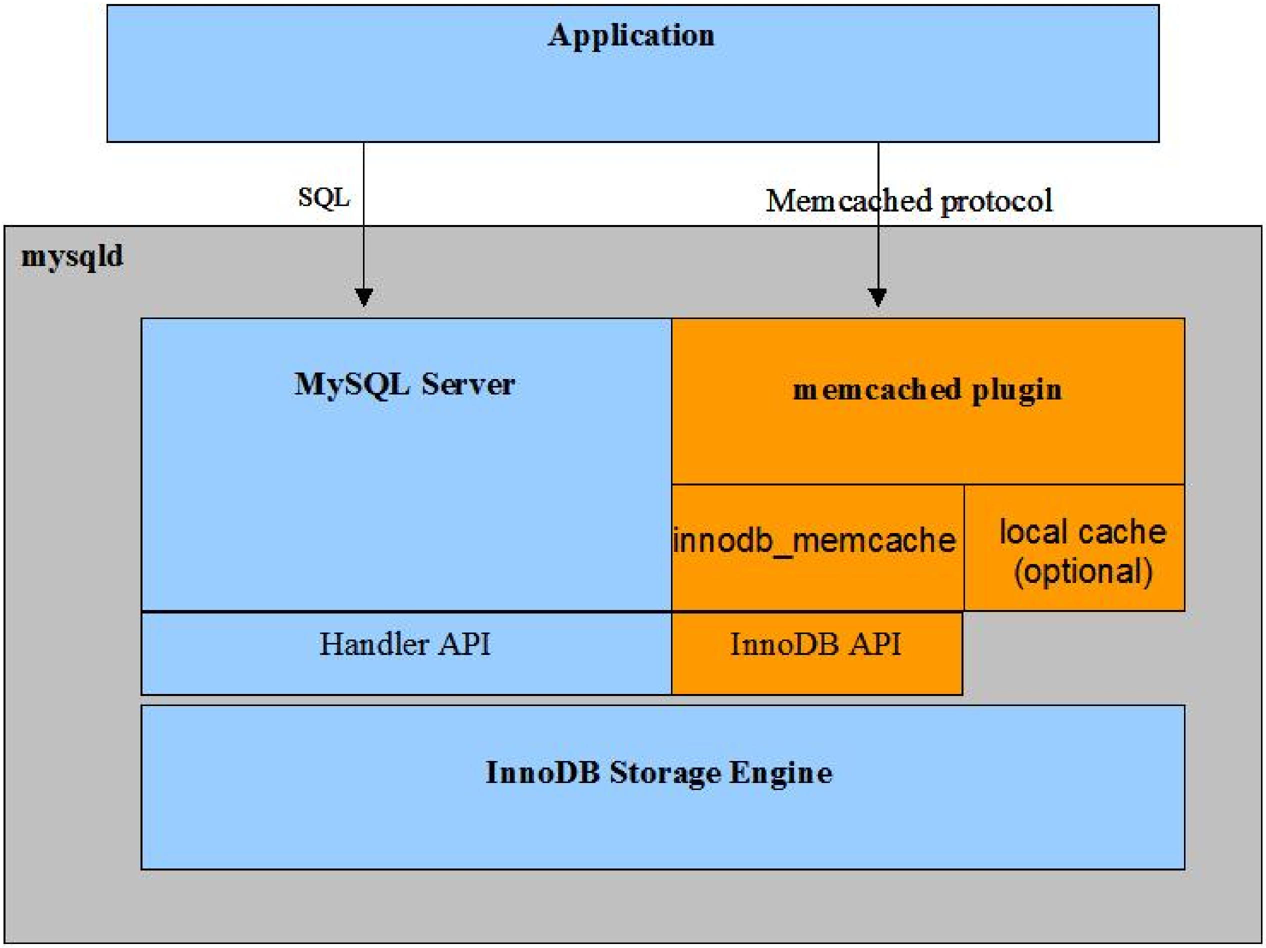
This section outlines advantages and usage scenarios for the memcached interface to InnoDB tables introduced in Section 14.2.16, "InnoDB Integration with memcached". The combination of InnoDB tables and memcached offers advantages over using either by themselves:

- Raw performance for simple lookups. Direct access to the InnoDB storage engine avoids the parsing and planning overhead of SQL. Running memcached in the same process space as the MySQL server avoids the network overhead of passing requests back and forth.
- Data is stored in a MySQL database to protect against crashes, outages, and corruption.
- The transfer between memory and disk is handled automatically, simplifying application logic.
- Data can be unstructured or structured, depending on the type of application. You can make an all-new table for the data, or map the NoSQL-style processing to one or more existing tables.
- You can still access the underlying table through SQL, for reporting, analysis, ad hoc queries, bulk loading, set operations such as union and intersection, and other operations well suited to the expressiveness and flexibility of SQL.
- You can ensure high availability of the NoSQL data by using this feature on a master server in combination with MySQL replication.
- The integration of memcached with MySQL provides a painless way to make the in-memory data persistent, so you can use it for more significant kinds of data. You can put more add, incr, and similar write operations into your application, without worrying that the data could disappear at any moment. You can stop and start the memcached server without losing updates made to the cached data. To guard against unexpected outages, you can take advantage of InnoDB crash recovery, replication, and backup procedures.
- The way InnoDB does fast primary key lookups is a natural fit for memcached single-item queries. The direct, low-level database access path used by the memcached plugin is much more efficient for key-value lookups than equivalent SQL queries.
- The serialization features of memcached, which can turn complex data structures, binary files, or even code blocks into storeable strings, offer a simple way to get such objects into a database.
- Because you can access the underlying data through SQL, you can produce reports, search or update across multiple keys, and call functions such as AVG() and MAX() on the memcached data. All of these operations are expensive or complicated with the standalone memcached.
- You do not need to manually load data into memcached at startup. As particular keys are requested by an application, the values are retrieved from the database automatically, and cached in memory using the InnoDB buffer pool.
- Because memcached consumes relatively little CPU, and its memory footprint is easy to control, it can run comfortably alongside a MySQL instance on the same system.
- Because data consistency is enforced through the usual mechanism as with regular InnoDB tables, you do not have to worry about stale memcached data or fallback logic to query the database in the case of a missing key.

14.2.16.2 Architecture of InnoDB and memcached Integration

This section describes how the memcached daemon is integrated into the MySQL Server, to help understand how this approach compares with other techniques that combine NoSQL components or interfaces with a MySQL back-end.

When integrated with MySQL Server, memcached is implemented as a MySQL plugin daemon, accessing the InnoDB storage engine directly and bypassing the SQL layer:



Features provided in the current release:

- memcached as a daemon plugin of mysqld: both mysqld and memcached run in the same process space, with very low latency access to data.
- Direct access to InnoDB tables, bypassing the SQL parser, the optimizer, and even the Handler API layer.
- Standard memcached protocols, both the text-based protocol and the binary protocol. The InnoDB + memcached combination passes all 55 compatibility tests from the memcapable command.
- Multi-column support: you can map multiple columns into the “value” part of the key/value store, with column values delimited by a user-specified separator character.
- By default, you use the memcached protocol to read and write data directly to InnoDB, and let MySQL manage the in-memory caching through the InnoDB buffer pool. The default settings represent the combination of high reliability with the fewest surprises for database applications. For example, the default settings avoid uncommitted data on the database side, or stale data returned for memcached get requests.

- Advanced users can configure the system as a traditional memcached server, with all data cached only in the memcached default engine (memory), or use a combination of the "memcached default engine" (memory caching) and the InnoDB memcached engine (InnoDB as backend persistent storage).
- You can control how often data is passed back and forth between InnoDB and memcached operations through the innodb_api_bk_commit_interval, daemon_memcached_r_batch_size, and daemon_memcached_w_batch_size configuration options. Both the batch size options default to a value of 1 for maximum reliability.
- You can specify any memcached configuration options through the MySQL configuration variable daemon_memcached_option. For example, you might change the port that memcached listens on, reduce the maximum number of simultaneous connections, change the maximum memory size for a key/value pair, or enable debugging messages for the error log.
- A configuration option innodb_api_trx_level lets you control the transaction isolation level on queries processed by the memcached interface. Although memcached has no concept of transactions, you might use this property to control how soon memcached sees changes caused by SQL statements, if you issue DML statements on the same table that memcached interfaces with. By default, it is set to READ UNCOMMITTED.
- Another configuration option is innodb_api_enable_mdl. "MDL" stands for "metadata locking". This basically locks the table from the MySQL level, so that the mapped table cannot be dropped or altered by DDL through the SQL interface. Without the lock, the table can be dropped from MySQL layer, but will be kept in the InnoDB storage until memcached or any other user stops using it.

Differences Between Using memcached Standalone or with InnoDB

MySQL users might already be familiar with using memcached along with MySQL, as described in Section 15.6, "Using MySQL with memcached". This section describes the similarities and differences between the information in that section, and when using the InnoDB integration features of the memcached that is built into MySQL. The link at the start of each item goes to the associated information about the traditional memcached server.

- Installation: Because the memcached library comes with the MySQL server, installation and setup are straightforward. You run a SQL script to set up a table for memcached to use, issue a one-time install plugin statement to enable memcached, and add to the MySQL configuration file or startup script any desired memcached options, for example to use a different port. You might still install the regular memcached distribution to get the additional utilities such as memcp, memcat, and memcapable.
- Deployment: It is typical to run large numbers of low-capacity memcached servers. Because the InnoDB + memcached combination has a 1:1 ratio between database and memcached servers, the typical deployment involves a smaller number of moderate or high-powered servers, machines that were already running MySQL. The benefit of this server configuration is more for improving the efficiency of each individual database server than in tapping into unused memory or distributing lookups across large numbers of servers. In the default configuration, very little memory is used for memcached, and the in-memory lookups are served from the InnoDB buffer pool, which automatically caches the most recently used and most frequently used data. As in a traditional MySQL server instance, keep the value of the innodb_buffer_pool_size configuration option as high as practical (without causing paging at the OS level), so that as much of the workload as possible is done in memory.
- Expiry: By default (that is, with the caching policy innodb_only), the latest data from the InnoDB table is always returned, so the expiry options have no practical effect. If you change the caching policy to caching or cache-only, the expiry options work as usual, but requested data might be stale if it was updated in the underlying table before it expires from the memory cache.
- Namespaces: memcached is like a single giant directory, where to keep files from conflicting with each other you might give them elaborate names with prefixes and suffixes. The integrated InnoDB

/ memcached server lets you use these same naming conventions for keys, with one addition. Key names of the format *@@table_id.key.table_id* are decoded to reference a specific a table, using mapping data from the innodb_memcache.containers table. The *key* is looked up in or written to the specified table.

The @@ notation only works for individual calls to the get, add, and set functions, not the others such as incr or delete. To designate the default table for all subsequent memcached operations within a session, perform a get request using the @@ notation and a table ID, but without the key portion. For example:

```
get @@table_x
```

Subsequent get, set, incr, delete and other operations use the table designated by table_x in the innodb_memcache.containers.name column.

- Hashing and distribution: The default configuration, with the caching policy innodb_only, is suitable for the traditional deployment configuration where all data is available on all servers, such as a set of replication slave servers.

  If you physically divide the data, as in a sharded configuration, you can split the data across several machines running the InnoDB and memcached combined server, and use the traditional memcached hashing mechanism to route requests to a particular machine. On the MySQL side, typically you would let all the data be inserted by add requests to memcached so the appropriate values were stored in the database on the appropriate server.

  These types of deployment best practices are still being codified.

- Memory usage: By default (with the caching policy innodb_only), the memcached protocol passes information back and forth with InnoDB tables, and the fixed-size InnoDB buffer pool handles the in-memory lookups rather than memcached memory usage growing and shrinking. Relatively little memory is used on the memcached side.

  If you switch the caching policy to caching or cache-only, the normal rules of memcached memory usage apply. Memory for the memcached data values is allocated in terms of “slabs”. You can control the slab size and maximum memory used for memcached.

  Either way, you can monitor and troubleshoot the integrated memcached daemon using the familiar statistics system, accessed through the standard protocol, for example over a telnet session. Because extra utilities are not included with the integrated daemon, to use the memcached-tool script, install a full memcached distribution.

- Thread usage: MySQL threads and memcached threads must co-exist on the same server, so any limits imposed on threads by the operating system apply to this total number.

- Log usage: Because the memcached daemon is run alongside the MySQL server and writes to stderr, the -v, -vv, and -vvv options for logging write their output to the MySQL error log.

- memcached operations: All the familiar operations such as get, set, add, and delete are available. Serialization (that is, the exact string format to represent complex data structures) depends on the language interface.

- Using memcached as a MySQL front end: That is what the InnoDB integration with memcached is all about. Putting these components together improves the performance of your application. Making InnoDB handle data transfers between memory and disk simplifies the logic of your application.

- Utilities: The MySQL server includes the libmemcached library but not the additional command-line utilities. To get the commands such as memcp, memcat, and memcapable commands, install a full memcached distribution. When memrm and memflush remove items from the cache, they are also removed from the underlying InnoDB table.
- Programming interfaces: You can access the MySQL server through the InnoDB and memcached combination using the same language as always: C and C++, Java, Perl, Python, PHP, and Ruby. Specify the server hostname and port as with any other memcached server. By default, the integrated memcached server listens on the same port as usual, 11211. You can use both the text and binary protocols. You can customize the behavior of the memcached functions at runtime. Serialization (that is, the exact string format to represent complex data structures) depends on the language interface.
- Frequently asked questions: MySQL has had an extensive memcached FAQ for several releases. In MySQL 5.7, the answers are largely the same, except that using InnoDB tables as a storage medium for memcached data means that you can use this combination for more write-intensive applications than before, rather than as a read-only cache.

For a more detailed look at the workings of this feature, see Section 14.2.16.7, “Internals of the InnoDB memcached Plugin”.

14.2.16.3 Getting Started with InnoDB Memcached Plugin

This section describes the steps to activate the InnoDB / memcached integration on a MySQL Server. Because the memcached daemon is tightly integrated with the MySQL Server to avoid network traffic and minimize latency, you perform this process on each MySQL instance that uses this feature.

Note

Before setting up the memcached interface for any data, consult Section 14.2.16.4, “Security Considerations for the InnoDB memcached Plugin” to understand the security procedures needed to prevent unauthorized access.

Prerequisites for the InnoDB memcached Plugin

Before you set up the plugin and the internal tables, verify that your server has the required prerequisite software.

Platform Support

Currently, the memcached Daemon Plugin is only supported on Linux, Solaris, and OS X platforms.

Software Prerequisites

You must have libevent installed, since it is required by memcached. The way to get this library is different if you use the MySQL installer or build from source, as described in the following sections.

Using a MySQL Installation Package

When you use a MySQL installer, the libevent library is not included. Use the particular method for your operating system to download and install libevent 1.4.3 or later: for example, depending on the operating system, you might use the command apt-get, yum, or port install. For example, on Ubuntu Linux:

```
sudo apt-get install libevent-dev
```

The libraries for memcached and the InnoDB plugin for memcached are put into the right place by the MySQL installer. For typical operation, the files lib/plugin/libmemcached.so and lib/plugin/innodb_engine.so are used.

Building from Source

For a brief introduction on the setup steps, see the file README-innodb_memcached in the source distribution in plugin/innodb_memcached. This is a more detailed explanation of that procedure.

If you have the source code release, libevent 1.4.3 is bundled with the package and is located at the top level of the MySQL source code directory. The bundled version of libevent is used unless you direct the build to use a local system version of libevent by setting -DWITH_LIBEVENT to system or yes.

When you build MySQL server, build with -DWITH_INNODB_MEMCACHED=ON. This will generate two shared libraries in the MySQL plugin directory that are required to run InnoDB memcached:

- libmemcached.so: the memcached daemon plugin to MySQL.
- innodb_engine.so: an InnoDB API plugin to memcached.

Setting Operating System Limits

The memcached daemon can sometimes cause the MySQL server to exceed the OS limit on the number of open files. You might need to run the ulimit command to increase the limit, and then start the MySQL server from that same shell. See Section 14.2.16.8, “Troubleshooting the InnoDB memcached Plugin” for the steps to resolve this issue.

Installing and Configuring the InnoDB memcached Plugin

Setting Up Required Tables

To configure the memcached plugin so it can interact with InnoDB tables, run the configuration script scripts/innodb_memcached_config.sql to install the necessary tables used behind the scenes:

```
mysql: source MYSQL_HOME/share/innodb_memcached_config.sql
```

This is a one-time operation. The tables remain in place if you later disable and re-enable the memcached support. For information about the layout and purpose of these tables, see Section 14.2.16.7, “Internals of the InnoDB memcached Plugin”.

Installing the Daemon Plugin

To activate the daemon plugin, use the install plugin statement, just as when installing any other MySQL plugin:

```
mysql> install plugin daemon_memcached soname "libmemcached.so";
```

Once the plugin is installed this way, it is automatically activated each time the MySQL server is booted or restarted.

Disabling the Daemon Plugin

When making major changes to the plugin configuration, you might need to turn off the plugin. To do so, issue the following statement:

```
mysql> uninstall plugin daemon_memcached;
```

To re-enable it, issue the preceding install plugin statement again. All the previous configuration settings, internal tables, and data are preserved when the plugin is restarted this way.

For additional information about enabling and disabling plugins, see Section 5.1.8.1, “Installing and Uninstalling Plugins”.

Specifying memcached Configuration Options

If you have any memcached specific configuration parameters, specify them on the mysqld command line or enter them in the MySQL configuration file, encoded in the argument to the daemon_memcached_option MySQL configuration option. The memcached configuration options take effect when the plugin is installed, which you do each time the MySQL server is started.

For example, to make memcached listen on port 11222 instead of the default port 11211, add -p11222 to the MySQL configuration option daemon_memcached_option:

```
mysqld .... --daemon_memcached_option="-p11222"
```

You can add other memcached command line options to the daemon_memcached_option string. The other configuration options are:

- daemon_memcached_engine_lib_name (default innodb_engine.so)
- daemon_memcached_engine_lib_path (default NULL, representing the plugin directory).
- daemon_memcached_r_batch_size, batch commit size for read operations (get). It specifies after how many memcached read operations the system automatically does a commit. By default, this is set to 1 so that every get request can access the very latest committed data in the InnoDB table, whether the data was updated through memcached or by SQL. When its value is greater than 1, the counter for read operations is incremented once for every get call. The flush_all call resets both the read and write counters.
- daemon_memcached_w_batch_size, batch commit for any write operations (set, replace, append, prepend, incr, decr, and so on) By default, this is set as 1, so that no uncommitted data is lost in case of an outage, and any SQL queries on the underlying table can access the very latest data. When its value is greater than 1, the counter for write operations is incremented once for every add, set, incr, decr, and delete call. The flush_all call resets both the read and write counters.

By default, you do not need to change anything with the first two configuration options. Those options allow you to load any other storage engine for memcached (such as the NDB memcached engine).

Again, please note that you will have these configuration parameters in your MySQL configuration file or MySQL boot command line. They take effect when you load the memcached plugin.

Summary

Now you have everything set up. You can directly interact with InnoDB tables through the memcached interface. To verify that the feature is working properly, see Verifying the InnoDB and memcached Setup.

Verifying the InnoDB and memcached Setup

Now that everything is set up, you can experiment with the InnoDB and memcached combination:

Here is an example using the Unix, Linux, or OS X command shell:

```
# Point memcached-related commands at the memcached attached to the mysqld process.
```

```
export MEMCACHED_SERVERS=127.0.0.1:11211
# Store the contents of a modestly sized text file in memcached, with the data passed
# to MySQL and stored in a table. The key is the basename of the file, 'mime.types'.
memcp /etc/apache2/mime.types
# Retrieve the data we just stored, from the memory cache.
memcat mime.types
```

Here is an example using telnet to send memcached commands and receive results through the ASCII protocol:

```
telnet 127.0.0.1 11211
set a11 10 0 9
123456789
STORED
get a11
VALUE a11 0 9
123456789
END
quit
```

To prove that all the same data has been stored in MySQL, connect to the MySQL server and issue:

```
mysql> select * from test.demo_test;
```

Now, shut down the MySQL server, which also shuts off the integrated memcached server. Further attempts to access the memcached data now fail with a connection error. Normally, the memcached data would disappear at this point, and you would write application logic to load the data back into memory when memcached was restarted. But the MySQL / memcached integration automates this process:

- Restart the MySQL server.
- Run the install plugin statement to start the daemon_memcached plugin again.
- Now any memcat commands or get operations once again return the key/value pairs you stored in the earlier memcached session. When a key is requested and the associated value is not already in the memory cache, it is automatically queried from the MySQL table, by default test.demo_test.

14.2.16.4 Security Considerations for the InnoDB memcached Plugin

Caution

Consult this section before deploying the InnoDB memcached plugin on any production servers, or even test servers if the MySQL instance contains any sensitive information.

Because memcached does not use an authentication mechanism by default, and the optional SASL authentication is not as strong as traditional DBMS security measures, make sure to keep only non-sensitive data in the MySQL instance using the InnoDB memcached plugin, and wall off any servers using this configuration from potential intruders. Do not allow memcached access to such servers from the Internet, only from within a firewalled intranet, ideally from a subnet whose membership you can restrict.

Password-Protecting the memcached Interface through SASL

SASL support gives you the capability to protect your MySQL database from unauthenticated access through memcached clients. This section explains the steps to enable this option. The steps to enable such support are almost identical to those you would do to enable SASL for a traditional memcached server.

Background Info:

SASL stands for “Simple Authentication and Security Layer”, a standard for adding authentication support to connection-based protocols. memcached added SASL support starting in its 1.4.3 release.

SASL authentication is only supported with the binary protocol.

For the InnoDB + memcached combination, the table that stores the memcached data must be registered in the container system table. And memcached clients can only access such a registered table. Even though the DBA can add access restrictions on a table that is registered with the memcached plugin, they have no control over who can access it through memcached applications. This is why we provide a means (through SASL) to control who can access InnoDB tables associated with the memcached plugin.

The following section shows how to build, enable, and test an SASL-enabled InnoDB memcached plugin.

Steps to Build and Enable SASL in InnoDB Memcached Plugin:

By default, SASL-enabled InnoDB memcached is not included in the release package, since it relies on building memcached with SASL libraries. To enable this feature, download the MySQL source and rebuild the InnoDB memcached plugin after downloading the SASL libraries:

1. First, get the SASL development and utility libraries. For example, on Ubuntu, you can get these libraries through:

   ```
   sudo apt-get -f install libsasl2-2 sasl2-bin libsasl2-2 libsasl2-dev libsasl2-modules
   ```

2. Then build the InnoDB memcached plugin (shared libraries) with SASL capability, by adding ENABLE_MEMCACHED_SASL=1 to the cmake options. In addition, memcached provides a simple plaintext password support, which is easier to use for testing. To enable this, set the option ENABLE_MEMCACHED_SASL_PWDB=1.

   Overall, you will add following three options to the cmake:

   ```
   cmake ... -DWITH_INNODB_MEMCACHED=1
     -DENABLE_MEMCACHED_SASL=1 -DENABLE_MEMCACHED_SASL_PWDB=1
   ```

3. The third step is to install the InnoDB memcached plugin as before, as explained in Section 14.2.16.3, “Getting Started with InnoDB Memcached Plugin”.

4. As previously mentioned, memcached provides a simple plaintext password support through SASL, which will be used for this demo.

   a. Create a user named testname and its password as testpasswd in a file:

      ```
      echo "testname:testpasswd:::::::" >/home/jy/memcached-sasl-db
      ```

   b. Let memcached know about it by setting the environment variable MEMCACHED_SASL_PWDB:

      ```
      export MEMCACHED_SASL_PWDB=/home/jy/memcached-sasl-db
      ```

   c. Also tell memcached that it is a plaintext password:

```
echo "mech_list: plain" > /home/jy/work2/msasl/clients/memcached.conf
export SASL_CONF_PATH=/home/jy/work2/msasl/clients/memcached.conf
```

5. Then reboot the server, and add a daemon_memcached_option option -S to enable SASL:

   ```
   mysqld ... --daemon_memcached_option="-S"
   ```

6. Now the setup is complete. To test it, you might need an SASL-enabled client, such as this SASL-enabled libmemcached.

   ```
   memcp --servers=localhost:11211 --binary  --username=testname
     --password=testpasswd myfile.txt

   memcat --servers=localhost:11211 --binary --username=testname
     --password=testpasswd myfile.txt
   ```

   Without appropriate user name or password, the above operation is rejected with the error message memcache error AUTHENTICATION FAILURE. Otherwise, the operation succeed. You can also examine the plaintext password set in the memcached-sasl-db file to verify it.

There are other methods to test SASL authentication with memcached. But the one described above is the most straightforward.

14.2.16.5 Writing Applications for the InnoDB memcached Interface

Typically, writing an application for the InnoDB memcached interface involves some degree of rewriting or adapting existing code that uses MySQL or the memcached API:

- Instead of many memcached servers running on low-powered machines, you have the same number of memcached servers as MySQL servers, running on relatively high-powered machines with substantial disk storage and memory. You might reuse some existing code that works with the memcached API, but some adaptation is likely needed due to the different server configuration.

- The data stored through this interface all goes into VARCHAR, TEXT, or BLOB columns, and must be converted to do numeric operations. You can do the conversion on the application side, or by using the CAST() function in queries.

- Coming from a database background, you might be used to general-purpose SQL tables with many columns. The tables accessed by the memcached code likely have only a few or even just a single column holding data values.

- You might adapt parts of your application that do single-row queries, inserts, updates, or deletes, to squeeze more performance out of critical sections of code. Both queries (read) and DML (write) operations can be substantially faster when performed through the memcached interface. The speedup for writes is typically greater than the speedup for reads, so you might focus on adapting the code that performs logging or records interactive choices on a web site.

The following sections explore these aspects in more detail.

Adapting an Existing MySQL Schema for a memcached Application

Consider these aspects of memcached applications when adapting an existing MySQL schema or application to use the memcached interface:

- memcached keys cannot contain spaces or newlines, because those characters are used as separators in the ASCII protocol. If you are using lookup values that contain spaces, transform or hash them into values without spaces before using them as keys in calls to add(), set(), get() and so on. Although theoretically those characters are allowed in keys in programs that use the binary protocol, you should always restrict the characters used in keys to ensure compatibility with a broad range of clients.

- If you have a short numeric primary key column in an InnoDB table, you can use that as the unique lookup key for memcached by converting the integer to a string value. If the memcached server is being used for more than one application, or with more than one InnoDB table, consider modifying the name to make sure it is unique. For example, you might prepend the table name, or the database name and the table name, before the numeric value.

  Note

  As of MySQL 5.7.3, the InnoDB memcached plugin supports inserts and reads on mapped InnoDB tables that have an INTEGER defined as the primary key.

- You cannot use a partitioned table for data queried or stored through the memcached interface.

- The memcached protocol passes numeric values around as strings. To store numeric values in the underlying InnoDB table, for example to implement counters that can be used in SQL functions such as SUM() or AVG():

  - Use VARCHAR columns with enough characters to hold all the digits of the largest expected number (and additional characters if appropriate for the negative sign, decimal point, or both).

  - In any query that performs arithmetic using the column values, use the CAST() function to convert from string to integer or other numeric type. For example:

    ```
    -- Alphabetic entries are returned as zero.
    select cast(c2 as unsigned integer) from demo_test;
    -- Since there could be numeric values of 0, can't disqualify them.
    -- Test the string values to find the ones that are integers, and average only those.
    select avg(cast(c2 as unsigned integer)) from demo_test
      where c2 between '0' and '9999999999';
    -- Views let you hide the complexity of queries. The results are already converted;
    -- no need to repeat conversion functions and WHERE clauses each time.
    create view numbers as select c1 key, cast(c2 as unsigned integer) val
      from demo_test where c2 between '0' and '9999999999';
    select sum(val) from numbers;
    ```

    Note that any alphabetic values in the result set are turned into 0 by the call to CAST(). When using functions such as AVG() that depend on the number of rows in the result set, include WHERE clauses to filter out any non-numeric values.

- If the InnoDB column you use as a key can be longer than 250 bytes, hash it to a value that is less than 250 bytes.

- To use an existing table with the memcached interface, define an entry for it in the innodb_memcache.containers table. To make that the table the default for all requests relayed through memcached, specify the value default in the name column, then restart the MySQL server to make that change take effect. If you are using multiple tables for different classes of memcached data, set up multiple entries in the innodb_memcache.containers table with name values of your choosing, then issue a memcached request of the form get @@*name* or set @@*name* within the application to switch the table used for subsequent requests through the memcached API.

  For an example of using a table other than the predefined test.demo_test table, see Example 14.14, “Specifying the Table and Column Mapping for an InnoDB + memcached Application”. For the required layout and meaning of the columns in such a table, see Section 14.2.16.7, “Internals of the InnoDB memcached Plugin”.

- To use multiple MySQL column values with memcached key/value pairs, in the innodb_memcache.containers entry associated with the MySQL table, specify in the value_columns field several column names separated by comma, semicolon, space, or pipe characters; for example, col1,col2,col3 or col1|col2|col3.

  Concatenate the column values into a single string using the pipe character as a separator, before passing that string to memcached add or set calls. The string is unpacked automatically into the various columns. Each get call returns a single string with the column values, also delimited by the pipe separator character. you unpack those values using the appropriate syntax depending on your application language.

Example 14.14 Specifying the Table and Column Mapping for an InnoDB + memcached Application

Here is an example showing how to use your own table for a MySQL application going through the InnoDB memcached plugin for data manipulation.

First, we set up a table to hold some country data: the population, area in metric units, and 'R' or 'L' indicating if people drive on the right or on the left.

```
use test;

CREATE TABLE `multicol` (
  `country` varchar(128) NOT NULL DEFAULT '',
  `population` varchar(10) DEFAULT NULL,
  `area_sq_km` varchar(9) DEFAULT NULL,
  `drive_side` varchar(1) DEFAULT NULL,
  `c3` int(11) DEFAULT NULL,
  `c4` bigint(20) unsigned DEFAULT NULL,
  `c5` int(11) DEFAULT NULL,
  PRIMARY KEY (`country`),
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Now we make a descriptor for this table so that the InnoDB memcached plugin knows how to access it:

- The sample entry in the CONTAINERS table has a name column 'aaa'; we set up another identifier 'bbb'. If we made a single master table for all memcached applications to use, we would make the ID 'default' and skip the @@ requests to switch tables.

- We specify the test.multicol table. The schema name is stored in one column and the table name is stored in another column.

- The key column will be our unique country value. That column was specified as the primary key when we created the table above, so we also specify the index name 'PRIMARY' here.

- Rather than a single column to hold a composite data value, we will divide the data among three table columns, so we specify a comma-separated list of those columns that will be used when storing or retrieving values.

- And for the flags, expire, and CAS values, we specify corresponding columns based on the settings from the sample table demo.test. These values are typically not significant in applications using the InnoDB memcached plugin, because MySQL keeps the data synchronized and there is no need to worry about data expiring or being stale.

```
insert into innodb_memcache.containers
  (name,db_schema,db_table,key_columns,value_columns,flags,cas_column,
  expire_time_column,unique_idx_name_on_key)
values
```

```
  ('bbb','test','multicol','country','population,area_sq_km,drive_side',
  'c3','c4','c5','PRIMARY');

commit;
```

Here is a sample Python program showing how we would access this table from a program:

- No database authorization is needed, since all data manipulation is done through the memcached interface. All we need to know is the port number the memcached daemon is listening to on the local system.
- We load sample values for a few arbitrary countries. (Area and population figures from Wikipedia.)
- To make the program use the multicol table, we call the switch_table() function that does a dummy GET or SET request using @@ notation. The name in the request is bbb, which is the value we stored in innodb_memcache.containers.name. (In a real application, we would use a more descriptive name. This example just illustrates that you specify a table identifier, not the table name, with the GET @@... request.
- The utility functions to insert and query the data demonstrate how we might turn a Python data structure into pipe-separated values for sending to MySQL with ADD or SET requests, and unpack the pipe-separated values returned by GET requests. This extra processing is only required when mapping the single memcached value to multiple MySQL table columns.

```
import sys, os
import memcache

def connect_to_memcached():
  memc = memcache.Client(['127.0.0.1:11211'], debug=0);
  print "Connected to memcached."
  return memc

def banner(message):
  print
  print "=" * len(message)
  print message
  print "=" * len(message)

country_data = [
("Canada","34820000","9984670","R"),
("USA","314242000","9826675","R"),
("Ireland","6399152","84421","L"),
("UK","62262000","243610","L"),
("Mexico","113910608","1972550","R"),
("Denmark","5543453","43094","R"),
("Norway","5002942","385252","R"),
("UAE","8264070","83600","R"),
("India","1210193422","3287263","L"),
("China","1347350000","9640821","R"),
]

def switch_table(memc,table):
  key = "@@" + table
  print "Switching default table to '" + table + "' by issuing GET for '" + key + "'."
  result = memc.get(key)

def insert_country_data(memc):
  banner("Inserting initial data via memcached interface")
  for item in country_data:
    country = item[0]
    population = item[1]
    area = item[2]
    drive_side = item[3]
    key = country
    value = "|".join([population,area,drive_side])
    print "Key = " + key
    print "Value = " + value

    if memc.add(key,value):
      print "Added new key, value pair."
    else:
      print "Updating value for existing key."
      memc.set(key,value)

def query_country_data(memc):
  banner("Retrieving data for all keys (country names)")
  for item in country_data:
    key = item[0]
    result = memc.get(key)
    print "Here is the result retrieved from the database for key " + key + ":"
    print result
    (m_population, m_area, m_drive_side) = result.split("|")
    print "Unpacked population value: " + m_population
    print "Unpacked area value      : " + m_area
    print "Unpacked drive side value: " + m_drive_side

if __name__ == '__main__':

  memc = connect_to_memcached()
  switch_table(memc,"bbb")
  insert_country_data(memc)
  query_country_data(memc)

  sys.exit(0)
```

Here are some SQL queries to illustrate the state of the MySQL data after the script is run, and show how you could access the same data directly through SQL, or from an application written in any language using the appropriate MySQL Connector or API.

The table descriptor `'bbb'` is in place, allowing us to switch to the `multicol` table by issuing a `memcached` request `GET @bbb`:

```
mysql: use innodb_memcache;
Database changed

mysql: select * from containers;
+------+-----------+-----------+-------------+----------------------------------+-------+------------+----
| name | db_schema | db_table  | key_columns | value_columns                    | flags | cas_column | exp
+------+-----------+-----------+-------------+----------------------------------+-------+------------+----
| aaa  | test      | demo_test | c1          | c2                               | c3    | c4         | c5
| bbb  | test      | multicol  | country     | population,area_sq_km,drive_side | c3    | c4         | c5
+------+-----------+-----------+-------------+----------------------------------+-------+------------+----
2 rows in set (0.01 sec)
```

After running the script, the data is in the `multicol` table, available for traditional MySQL queries or DML statements:

```
mysql: use test;
Database changed

mysql: select * from multicol;
+---------+------------+------------+------------+------+------+------+
| country | population | area_sq_km | drive_side | c3   | c4   | c5   |
+---------+------------+------------+------------+------+------+------+
| Canada  | 34820000   | 9984670    | R          |    0 |   11 |    0 |
| China   | 1347350000 | 9640821    | R          |    0 |   20 |    0 |
| Denmark | 5543453    | 43094      | R          |    0 |   16 |    0 |
| India   | 1210193422 | 3287263    | L          |    0 |   19 |    0 |
| Ireland | 6399152    | 84421      | L          |    0 |   13 |    0 |
| Mexico  | 113910608  | 1972550    | R          |    0 |   15 |    0 |
| Norway  | 5002942    | 385252     | R          |    0 |   17 |    0 |
| UAE     | 8264070    | 83600      | R          |    0 |   18 |    0 |
| UK      | 62262000   | 243610     | L          |    0 |   14 |    0 |
| USA     | 314242000  | 9826675    | R          |    0 |   12 |    0 |
+---------+------------+------------+------------+------+------+------+
10 rows in set (0.00 sec)

mysql: desc multicol;
+------------+---------------------+------+-----+---------+-------+
| Field      | Type                | Null | Key | Default | Extra |
+------------+---------------------+------+-----+---------+-------+
| country    | varchar(128)        | NO   | PRI |         |       |
| population | varchar(10)         | YES  |     | NULL    |       |
| area_sq_km | varchar(9)          | YES  |     | NULL    |       |
| drive_side | varchar(1)          | YES  |     | NULL    |       |
| c3         | int(11)             | YES  |     | NULL    |       |
| c4         | bigint(20) unsigned | YES  |     | NULL    |       |
| c5         | int(11)             | YES  |     | NULL    |       |
+------------+---------------------+------+-----+---------+-------+
7 rows in set (0.01 sec)
```

Allow sufficient size to hold all necessary digits, decimal points, sign characters, leading zeros, and so on when defining the length for columns that will be treated as numbers. Too-long values in a string column such as a VARCHAR are truncated by removing some characters, which might produce a nonsensical numeric value.

We can produce reports through SQL queries, doing calculations and tests across any columns, not just the country key column. (Because these examples use data from only a few countries, the numbers are for illustration purposes only.) Here, we find the average population of countries where people drive on the right, and the average size of countries whose names start with “U”:

```

mysql: select avg(population) from multicol where drive_side = 'R';
+-------------------+
| avg(population)   |
+-------------------+
| 261304724.7142857 |
+-------------------+
1 row in set (0.00 sec)

mysql: select sum(area_sq_km) from multicol where country like 'U%';
+-----------------+
| sum(area_sq_km) |
+-----------------+
|        10153885 |
+-----------------+
1 row in set (0.00 sec)
```

Because the population and area_sq_km columns store character data rather than strongly typed numeric data, functions such as avg() and sum() work by converting each value to a number first. This approach *does not work* for operators such as < or >: for example, when comparing character-based values, 9 > 1000, which is not you expect from a clause such as ORDER BY population DESC. For the most accurate type treatment, perform queries against views that cast numeric columns to the appropriate types. This technique lets you issue very simple SELECT * queries from your database applications, while ensuring that all casting, filtering, and ordering is correct. Here, we make a view that can be queried to find the top 3 countries in descending order of population, with the results always reflecting the latest data from the multicol table, and with the population and area figures always treated as numbers:

```
mysql: create view populous_countries as
  select
    country,
    cast(population as unsigned integer) population,
    cast(area_sq_km as unsigned integer) area_sq_km,
    drive_side from multicol
  order by cast(population as unsigned integer) desc
  limit 3;
Query OK, 0 rows affected (0.01 sec)

mysql: select * from populous_countries;
+---------+------------+------------+------------+
| country | population | area_sq_km | drive_side |
+---------+------------+------------+------------+
| China   | 1347350000 |    9640821 | R          |
| India   | 1210193422 |    3287263 | L          |
| USA     |  314242000 |    9826675 | R          |
+---------+------------+------------+------------+
3 rows in set (0.00 sec)

mysql: desc populous_countries;
+------------+---------------------+------+-----+---------+-------+
| Field      | Type                | Null | Key | Default | Extra |
+------------+---------------------+------+-----+---------+-------+
| country    | varchar(128)        | NO   |     |         |       |
| population | bigint(10) unsigned | YES  |     | NULL    |       |
| area_sq_km | int(9) unsigned     | YES  |     | NULL    |       |
| drive_side | varchar(1)          | YES  |     | NULL    |       |
+------------+---------------------+------+-----+---------+-------+
4 rows in set (0.02 sec)
```

Adapting an Existing `memcached` Application for the Integrated `memcached` Daemon

Consider these aspects of MySQL and `InnoDB` tables when adapting an existing `memcached` application to use the MySQL integration:

- If you have key values longer than a few bytes, you might find it more efficient to use a numeric auto-increment column for the primary key in the `InnoDB` table, and create a unique secondary index on the column holding the `memcached` key values. This is because `InnoDB` performs best for large-scale insertions if the primary key values are added in sorted order (as they are with auto-increment values), and the primary key values are duplicated in each secondary index, which can take up unnecessary space when the primary key is a long string value.

- If you store several different classes of information in `memcached`, you might set up a separate `InnoDB` table for each kind of data. Define additional table identifiers in the `innodb_memcache.containers` table, and use the notation `@@`*`table_id.key`* to store or retrieve items from different tables. Physically dividing the items lets you tune the characteristics of each table for best space utilization, performance, and reliability. For example, you might enable compression for a table that holds blog posts, but not for one that holds thumbnail images. You might back up one table more frequently than another because it holds critical data. You might create additional secondary indexes on tables that are frequently used to generate reports through SQL.

- Preferably, set up a stable set of table definitions for use with the `memcached` interface and leave them in place permanently. Changes to the `containers` table take effect the next time that table is queried. The entries in that table are processed at startup, and are consulted whenever an unrecognized table ID is requested by the `@@` notation. Thus, new entries are visible as soon as you try to use the associated table ID, but changes to existing entries require a server restart before they take effect.

- When you use the default caching policy `innodb_only`, your calls to `add()`, `set()`, `incr()`, and so on can succeed but still trigger debugging messages such as `while expecting 'STORED', got` `unexpected response 'NOT_STORED`. This is because in the `innodb_only` configuration, new and updated values are sent directly to the `InnoDB` table without being saved in the memory cache.

Tuning Performance of the InnoDB memcached Plugin

Because using `InnoDB` in combination with `memcached` involves writing all data to disk, whether immediately or sometime later, understand that raw performance is expected to be somewhat lower than using `memcached` by itself. Focus your tuning goals for the `InnoDB` `memcached` plugin on achieving higher performance than equivalent SQL operations.

Benchmarks suggest that both queries and DML operations (inserts, updates, and deletes) are faster going through the `memcached` interface than with traditional SQL. DML operations typically see a larger speedup. Thus, the types of applications you might adapt to use the `memcached` interface first are those that are write-intensive. You might also use MySQL as a data store for types of write-intensive applications that formerly used some fast, lightweight mechanism where reliability was not a priority.

Adapting SQL Queries

The types of queries that are most suited to the simple `GET` request style are those with a single clause, or a set of `AND` conditions, in the `WHERE` clause:

```
SQL:
select col from tbl where key = 'key_value';

memcached:
GET key_value

SQL:
select col from tbl where col1 = val1 and col2 = val2 and col3 = val3;

memcached:
# Since you must always know these 3 values to look up the key,
# combine them into a unique string and use that as the key
# for all ADD, SET, and GET operations.
key_value = val1 + ":" + val2 + ":" + val3
GET key_value

SQL:
select 'key exists!' from tbl
  where exists (select col1 from tbl where key = 'key_value') limit 1;

memcached:
# Test for existence of key by asking for its value and checking if the call succeeds,
# ignoring the value itself. For existence checking, you typically only store a very
# short value such as "1".
GET key_value
```

Taking Advantage of System Memory

For best performance, deploy the `InnoDB` `memcached` plugin on machines that are configured like typical database servers: in particular, with the majority of system RAM devoted to the `InnoDB` buffer pool through the `innodb_buffer_pool_size` configuration option. For systems with multi-gigabyte buffer pools, consider raising the value of the `innodb_buffer_pool_instances` configuration option for maximum throughput when most operations involve data already cached in memory.

Reducing Redundant I/O

`InnoDB` has a number of settings that let you choose the balance between high reliability in case of a crash, and the amount of I/O overhead during high write workloads. For example, consider setting the configuration options `innodb_doublewrite=0` and `innodb_flush_log_at_trx_commit=2`. Measure the performance with different settings for the innodb_flush_method option. If the binary log is not turned on for the server, use the setting innodb_support_xa=0.

For other ways to reduce or tune I/O for table operations, see Section 8.5.7, “Optimizing InnoDB Disk I/O”.

Reducing Transactional Overhead

The default value of 1 for the configuration options daemon_memcached_r_batch_size and daemon_memcached_w_batch_size is intended for maximum reliability of results and safety of stored or updated data.

Depending on the type of application, you might increase one or both of these settings to reduce the overhead of frequent commit operations. On a busy system, you might increase daemon_memcached_r_batch_size, knowing that changes to the data made through SQL might not become visible to memcached immediately (that is, until *N* more get operations were processed). When processing data where every write operation must be reliably stored, you would leave daemon_memcached_w_batch_size set to 1. You might increase it when processing large numbers of updates intended to only be used for statistical analysis, where it is not critical if the last *N* updates are lost in case of a crash.

For example, imagine a system that monitors traffic crossing a busy bridge, recording approximately 100,000 vehicles each day. If the application simply counts different types of vehicles to analyze traffic patterns, it might change daemon_memcached_w_batch_size from 1 to 100, reducing the I/O overhead for commit operations by 99%. In case of an unexpected outage, only a maximum of 100 records could be lost, which might be an acceptable margin of error. If instead the application was doing automated toll collection for each car, it would keep daemon_memcached_w_batch_size set to 1 to ensure that every toll record was immediately saved to disk.

Because of the way InnoDB organizes the memcached key values on disk, if you have a large number of keys to create, it can be faster to sort all the data items by the key value in your application and add them in sorted order, rather than creating them in arbitrary order.

The memslap command, which is part of the regular memcached distribution but not included with the MySQL server, can be useful for benchmarking different configurations. It can also be used to generate sample key/value pairs that you can use in your own benchmarking. See libmemcached Command-Line Utilities for details.

Controlling Transactional Behavior of the InnoDB memcached Plugin

Unlike with the traditional memcached, with the InnoDB + memcached combination you can control how “durable” are the data values produced through calls to add, set, incr, and so on. Because MySQL places a high priority on durability and consistency of data, by default all data written through the memcached interface is always stored to disk, and calls to get always return the most recent value from disk. Although this default setting does not give the highest possible raw performance, it is still very fast compared to the traditional SQL interface for InnoDB tables.

As you gain experience with this feature, you can make the decision to relax the durability settings for non-critical classes of data, at the risk of possibly losing some updated values in case of an outage, or returning data that is slightly out-of-date.

Frequency of Commits

One tradeoff between durability and raw performance is how frequently new and changed data is committed. If the data is critical, you want it to be committed immediately so that it is safe in case of any crash or outage. If the data is less critical, such as counters that would be reset after a crash, or debugging or logging data where you could afford to lose a few seconds worth, you might prefer the higher raw throughput that comes with less frequent commits.

When a memcached operation causes an insert, update, or delete in the underlying InnoDB table, that change might be committed to the underlying table instantly (if daemon_memcached_w_batch_size=1) or some time later (if that configuration option value is greater than 1). In either case, the change cannot be rolled back. If you increase the value of daemon_memcached_w_batch_size=1 to avoid high I/O overhead during busy times, commits could become very infrequent when the workload decreases. As a safety measure, a background thread automatically commits changes made through the memcached API at regular intervals. The interval is controlled by the innodb_api_bk_commit_interval configuration option, and by default is 5 seconds.

When a memcached operation causes an insert or update in the underlying InnoDB table, the changed data is immediately visible to other memcached requests because the new value remains in the memory cache, even if it is not committed yet on the MySQL side.

Transaction Isolation

When a memcached operation such as get or incr causes a query or DML operation in the underlying InnoDB table, you can control whether it sees the very latest data written to the table, only data that has been committed, or other variations of transaction isolation level. You control this feature through the innodb_api_trx_level configuration option. The numeric values specified with this option correspond to the familiar isolation level names such as REPEATABLE READ. See the description of the innodb_api_trx_level option for the full list.

The stricter the isolation level, the more certain you can be that the data you retrieve will not be rolled back or changed suddenly so that a subsequent query sees a different value. But that strictness comes with greater locking overhead that can cause waits. For a NoSQL-style application that does not use long-running transactions, you can typically stay with the default isolation level or switch to a less strict one.

Disabling Row Locks for memcached DML Operations

The innodb_api_disable_rowlock option can be used to disable row locks when InnoDB memcached performs DML operations. By default, innodb_api_disable_rowlock is set to OFF which means that memcached requests row locks for get and set operations. When innodb_api_disable_rowlock is set to ON, memcached requests a table lock instead of row locks.

The innodb_api_disable_rowlock option is not dynamic. It must be specified at startup on the mysqld command line or entered in the MySQL configuration file.

Allowing or Disallowing DDL

By default, you can perform DDL operations such as ALTER TABLE on the tables being used by the InnoDB memcached plugin. To avoid potential slowdowns when these tables are being used for high-throughput applications, you can disable DDL operations on these tables by turning on the innodb_api_enable_mdl configuration option at startup. This option is less appropriate when you are accessing the same underlying tables through both the memcached interface and SQL, because it blocks CREATE INDEX statements on the tables, which could be important for configuring the system to run reporting queries.

Data Stored on Disk, in Memory, or Both

Table innodb_memcache.cache_policies specifies whether to store data written through the memcached on disk (innodb_only, the default); to store the data in memory only, as in the traditional memcached (cache-only); or both (caching).

With the caching setting, if memcached cannot find a key in memory, it searches for the value in an InnoDB table. Values returned from get calls under the caching setting could be out-of-date, if they were updated on disk in the InnoDB table but not yet expired from the memory cache.

The caching policy can be set independently for get, set (including incr and decr), delete, and flush operations. For example:

- You might allow get and set operations to query or update a table and the memcached memory cache at the same time (through the caching setting), while making delete, flush, or both operate only on the in-memory copy (through the cache_only setting). That way, deleting or flushing an item just expires it from the cache, and the latest value is returned from the InnoDB table the next time the item is requested.

```
mysql> desc innodb_memcache.cache_policies;
+---------------+-------------------------------------------------------+------+-----+---------+-------+
| Field         | Type                                                  | Null | Key | Default | Extra |
+---------------+-------------------------------------------------------+------+-----+---------+-------+
| policy_name   | varchar(40)                                           | NO   | PRI | NULL    |       |
| get_policy    | enum('innodb_only','cache_only','caching','disabled') | NO   |     | NULL    |       |
| set_policy    | enum('innodb_only','cache_only','caching','disabled') | NO   |     | NULL    |       |
| delete_policy | enum('innodb_only','cache_only','caching','disabled') | NO   |     | NULL    |       |
| flush_policy  | enum('innodb_only','cache_only','caching','disabled') | NO   |     | NULL    |       |
+---------------+-------------------------------------------------------+------+-----+---------+-------+

mysql> select * from innodb_memcache.cache_policies;
+--------------+-------------+-------------+---------------+--------------+
| policy_name  | get_policy  | set_policy  | delete_policy | flush_policy |
+--------------+-------------+-------------+---------------+--------------+
| cache_policy | innodb_only | innodb_only | innodb_only   | innodb_only  |
+--------------+-------------+-------------+---------------+--------------+

mysql> update innodb_memcache.cache_policies set set_policy = 'caching'
    -> where policy_name = 'cache_policy';
```

The cache_policies values are only read at startup, and are tightly integrated with the operation of the memcached plugin. After changing any of the values in this table, uninstall the plugin and reinstall it:

```
mysql> uninstall plugin daemon_memcached;
Query OK, 0 rows affected (2.00 sec)
mysql> install plugin daemon_memcached soname "libmemcached.so";
Query OK, 0 rows affected (0.00 sec)
```

Adapting DML Statements to memcached Operations

Benchmarks suggest that the InnoDB memcached plugin speeds up DML operations (inserts, updates, and deletes) more than it speeds up queries. You might focus your initial development efforts on write-intensive applications that are I/O-bound, and look for opportunities to use MySQL for new kinds of write-intensive applications.

-
  ```
  INSERT INTO t1 (key,val) VALUES (some_key,some_value);
  SELECT val FROM t1 WHERE key = some_key;
  UPDATE t1 SET val = new_value WHERE key = some_key;
  UPDATE t1 SET val = val + x WHERE key = some_key;
  DELETE FROM t1 WHERE key = some_key;
  ```

  Single-row DML statements are the most straightforward kinds of statements to turn into memcached operations: INSERT becomes add, UPDATE becomes set, incr or decr, and DELETE becomes delete. When issued through the memcached interface, these operations are guaranteed to affect only 1 row because *key* is unique within the table.

In the preceding SQL examples, t1 refers to the table currently being used by the InnoDB memcached plugin based on the configuration settings in the innodb_memcache.containers table, key refers to the column listed under key_columns, and val refers to the column listed under value_columns.

-
  ```
  TRUNCATE TABLE t1;
  DELETE FROM t1;
  ```

  Corresponds to the flush_all operation, when t1 is configured as the table for memcached operations as in the previous step. Removes all the rows in the table.

Performing DML and DDL Statements on the Underlying InnoDB Table

You can access the InnoDB table (by default, test.demo_test) through the standard SQL interfaces. However, there are some restrictions:

- When query a table through SQL that is also being accessed through the memcached interface, remember that memcached operations can be configured to be committed periodically rather than after every write operation. This behavior is controlled by the daemon_memcached_w_batch_size option. If this option is set to a value greater than 1, use READ UNCOMMITTED queries to find the just-inserted rows:

  ```
  mysql> set session TRANSACTION ISOLATION LEVEL read uncommitted;
  Query OK, 0 rows affected (0.00 sec)

  mysql> select * from demo_test;
  +------+------+------+------+-----------+------+------+------+------+------+------+
  | cx   | cy   | c1   | cz   | c2        | ca   | CB   | c3   | cu   | c4   | C5   |
  +------+------+------+------+-----------+------+------+------+------+------+------+
  | NULL | NULL | a11  | NULL | 123456789 | NULL | NULL |   10 | NULL |    3 | NULL |
  +------+------+------+------+-----------+------+------+------+------+------+------+
  1 row in set (0.00 sec)
  ```

- To modify a table through SQL that is also being accessed through the memcached interface, remember that memcached operations can be configured to be start a new transaction periodically rather than for every read operation. This behavior is controlled by the daemon_memcached_r_batch_size option. If this option is set to a value greater than 1, ...

- The InnoDB table is locked IS (shared intention) or IX (exclusive intentional) for all operations in a transaction. If you increase daemon_memcached_r_batch_size and daemon_memcached_w_batch_size substantially from their default value of 1, the table is most likely intentionally locked between each operation, preventing you from running DDL statements on the table.

14.2.16.6 Using the InnoDB memcached Plugin with Replication

Because the InnoDB memcached daemon plugin supports the MySQL binary log, any updates made on a master server through the memcached interface can be replicated for backup, balancing intensive read workloads, and high availability. All memcached commands are supported for binlogging.

You do not need to set up the InnoDB memcached plugin on the slave servers. In this configuration, the primary advantage is increased write throughput on the master. The speed of the replication mechanism is not affected.

The following sections show how to use the binlog capability, to use the InnoDB memcached plugin along with MySQL replication. It assumes you have already done the basic setup described in Section 14.2.16.3, “Getting Started with InnoDB Memcached Plugin”.

Enable InnoDB Memcached Binary Log with innodb_api_enable_binlog:

- To use the InnoDB memcached plugin with the MySQL binary log, enable the innodb_api_enable_binlog configuration option on the master server. This option can only be set at server boot time. You must also enable the MySQL binary log on the master server with the --log-bin option. You can add these options to your server configuration file such as my.cnf, or on the mysqld command line.

  ```
  mysqld ... --log-bin -–innodb_api_enable_binlog=1
  ```

- Then configure your master and slave server, as described in Section 16.1.1, “How to Set Up Replication”.

- Use mysqldump to create a master data snapshot, and sync it to the slave server.

  ```
  master shell: mysqldump --all-databases --lock-all-tables > dbdump.db
  slave shell: mysql < dbdump.db
  ```

- On the master server, issue show master status to obtain the Master Binary Log Coordinates:

  ```
  mysql> show master status;
  ```

- On the slave server, use a change master to statement to set up a slave server with the above coordinates:

  ```
  mysql> CHANGE MASTER TO
         MASTER_HOST='localhost',
         MASTER_USER='root',
         MASTER_PASSWORD='',
         MASTER_PORT = 13000,
         MASTER_LOG_FILE='0.000001,
         MASTER_LOG_POS=114;
  ```

- Then start the slave:

  ```
  mysql> start slave;
  ```

  If the error log prints output similar to the following, the slave is ready for replication:

  ```
  2013-09-24T13:04:38.639684Z 49 [Note] Slave I/O thread: connected to
  master 'root@localhost:13000', replication started in log '0.000001'
  at position 114
  ```

Test with the memcached telnet interface:

To test the server with the above replication setup, we use the memcached telnet interface, and also query the master and slave servers using SQL to verify the results.

In our configuration setup SQL, one example table demo_test is created in the test database for use by memcached. We will use this default table for the demonstrations:

- Use set to insert a record, key test1, value t1, and flag 10:

```
telnet 127.0.0.1 11211
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
set test1 10 0 2
t1
STORED
```

In the master server, you can see that the row is inserted. c1 maps to the key, c2 maps to the value, c3 is the flag, c4 is the cas value, and c5 is the expiration.

```
mysql> select * from test.demo_test;
```

c1	c2	c3	c4	c5
test1	t1	10	2	0

```
1 row in set (0.00 sec)
```

In the slave server, you will see the same record is inserted by replication:

```
mysql> select * from test.demo_test;
```

c1	c2	c3	c4	c5
test1	t1	10	2	0

```
1 row in set (0.00 sec)
```

- Use set command to update the key test1 to a new value new:

```
Connected to 127.0.0.1.
Escape character is '^]'.
set test1 10 0 3
new
STORED
```

From the slave server, the update is replicated (notice the cas value also updated):

```
mysql> select * from test.demo_test;
```

c1	c2	c3	c4	c5
test1	new	10	3	0

```
1 row in set (0.00 sec)
```

- Delete the record with a `delete` command:

```
Connected to 127.0.0.1.
Escape character is '^]'.
delete test1
DELETED
```

When the delete is replicated to the slave, the record on the slave is also deleted:

```
mysql> select * from test.demo_test;
Empty set (0.00 sec)
```

- Truncate the table with the `flush_all` command.

First, insert two records by telnetting to the master server:

```
Connected to 127.0.0.1.
Escape character is '^]'
set test2 10 0 5
again
STORED
set test3 10 0 6
again1
STORED
```

In the slave server, confirm these two records are replicated:

```
mysql> select * from test.demo_test;
```

c1	c2	c3	c4	c5
test2	again	10	5	0
test3	again1	10	6	0

```
2 rows in set (0.00 sec)
```

Call `flush_all` in the telnet interface to truncate the table:

```
Connected to 127.0.0.1.
Escape character is '^]'.
flush_all
OK
```

Then check that the truncation operation is replicated on the slave server:

```
mysql> select * from test.demo_test;
Empty set (0.00 sec)
```

All memcached commands are supported in terms of replication.

Notes for the InnoDB Memcached Binlog:

Binlog Format:

- Most memcached operations are mapped to DML statements (analogous to insert, delete, update). Since there is no actual SQL statement being processed by the MySQL server, all memcached commands (except for flush_all) use Row-Based Replication (RBR) logging. This is independent of any server binlog_format setting.
- The memcached flush_all command is mapped to the TRUNCATE TABLE command. Since DDL commands can only use statement-based logging, this flush_all command is replicated by sending a TRUNCATE TABLE statement.

Transactions:

- The concept of transactions has not typically been part of memcached applications. We use daemon_memcached_r_batch_size and daemon_memcached_w_batch_size to control the read and write transaction batch size for performance considerations. These settings do not affect replication: each SQL operation on the underlying table is replicated right after successful completion.
- The default value of daemon_memcached_w_batch_size is 1, so each memcached write operation is committed immediately. This default setting incurs a certain amount of performance overhead, to avoid any inconsistency in the data visible on the master and slave servers. The replicated records will always be available immediately on the slave server. If you set daemon_memcached_w_batch_size greater than 1, records inserted or updated through the memcached interface are not immediately visible on the master server; to view these records on the master server before they are committed, issue set transaction isolation level read uncommitted.

14.2.16.7 Internals of the InnoDB memcached Plugin

InnoDB API for the InnoDB memcached Plugin

The InnoDB memcached engine accesses InnoDB through InnoDB APIs. Most of the APIs are directly adopted from embedded InnoDB. InnoDB API functions are passed to InnoDB memcached as “callback functions”. InnoDB API functions access the InnoDB table directly, and are mostly DML operations except for the TRUNCATE TABLE operation.

All memcached commands, listed below, are implemented through the InnoDB memcached API. The following table outlines how each memcached command is mapped to a DML operation.

Table 14.11 memcached Commands and Associated DML Operation

memcached Command	DML Operation
get	a read/fetch command
set	a search followed by an insertion or update (depending on whether or not a key exists)
add	a search followed by an insertion or update

memcached Command	DML Operation
replace	a search followed by an update
append	a search followed by an update (appends data to the result before update)
prepend	a search followed by an update (prepends data to the result before update)
incr	a search followed by an update
decr	a search followed by an update
delete	a search followed by a deletion
flush_all	truncate table

Underlying Tables Used by the InnoDB memcached Plugin

This section explains the details of the underlying tables used by the InnoDB / memcached plugin.

The configuration script, scripts/innodb_memcached_config.sql, installs 3 tables needed by the InnoDB memcached. These tables are created in a dedicated database innodb_memcache:

```
mysql> use innodb_memcache;
Database changed
mysql> show tables;
+---------------------------+
| Tables_in_innodb_memcache |
+---------------------------+
| cache_policies            |
| config_options            |
| containers                |
+---------------------------+
3 rows in set (0.01 sec)
```

containers Table

containers - This table is the most important table for the memcached daemon. It describes the table or tables used to store the memcached values. You must make changes to this table to start using the memcached interface with one or more of your own tables, rather than just experimenting with the test.demo_test table.

The mapping is done through specifying corresponding column values in the table:

```
mysql> desc containers;
+------------------------+--------------+------+-----+---------+-------+
| Field                  | Type         | Null | Key | Default | Extra |
+------------------------+--------------+------+-----+---------+-------+
| name                   | varchar(50)  | NO   | PRI | NULL    |       |
| db_schema              | varchar(250) | NO   |     | NULL    |       |
| db_table               | varchar(250) | NO   |     | NULL    |       |
| key_columns            | varchar(250) | NO   |     | NULL    |       |
| value_columns          | varchar(250) | YES  |     | NULL    |       |
| flags                  | varchar(250) | NO   |     | 0       |       |
| cas_column             | varchar(250) | YES  |     | NULL    |       |
| expire_time_column     | varchar(250) | YES  |     | NULL    |       |
| unique_idx_name_on_key | varchar(250) | NO   |     | NULL    |       |
+------------------------+--------------+------+-----+---------+-------+
9 rows in set (0.02 sec)
```

- db_schema and db_table columns specify the database and table name for storing the memcached value.
- key_columns specifies the single column name used as the lookup key for memcached operations.
- value_columns describes the columns (one or more) used as values for memcached operations. To specify multiple columns, separate them with pipe characters (such as col1|col2|col3 and so on).
- unique_idx_name_on_key is the name of the index on the key column. It must be a unique index. It can be the primary key or a secondary index. Preferably, make the key column the primary key of the InnoDB table. Doing so saves a lookup step over using a secondary index for this column. You cannot make a covering index for memcached lookups; InnoDB returns an error if you try to define a composite secondary index over both the key and value columns.

The above 5 column values (table name, key column, value column and index) must be supplied. Otherwise, the setup will fail.

Although the following values are optional, they are required for full compliance with the memcached protocol. If you do not use flags, cas_column, or expiration_time_column, set their value to 0 to indicate that they are unused. Failing to do so will result in an error when you attempt to load the plugin.

- flags specifies the columns used as flags (a user-defined numeric value that is stored and retrieved along with the main value) for memcached. It is also used as the column specifier for some operations (such as incr, prepend) if memcached value is mapped to multiple columns. So the operation would be done on the specified column. For example, if you have mapped a value to 3 columns, and only want the increment operation performed on one of these columns, you can use flags to specify which column will be used for these operations. If you do not use the flags column, set its value to 0 to indicate that it is unused.
- cas_column and expiration_time_column are used specifically to store the cas (compare-and-swap) and exp (expiry) value of memcached. Those values are related to the way memcached hashes requests to different servers and caches data in memory. Because the InnoDB memcached plugin is so tightly integrated with a single memcached daemon, and the in-memory caching mechanism is handled by MySQL and the buffer pool, these columns are rarely needed in this type of deployment. If you do not use these columns, set their value to 0 to indicate that the columns are unused.

containers Table Column Constraints

- key_columns: The maximum limit for a memcached key is 250 characters, which is enforced by memcached. If a mapped key longer than the maximum limit is used, the operation will fail. The mapped key must be a non-Null CHAR or VARCHAR type.
- value_columns: Must be mapped to a CHAR, VARCHAR, or BLOB column. There is no length restriction and the value can be NULL.
- cas_column: The cas value is a 64 bit integer. It must be mapped to a BIGINT of at least 8 bytes. If you do not use this column, set its value to 0 to indicate that it is unused.
- expiration_time_column: Must mapped to an INTEGER of at least 4 bytes. Expiration time is defined as a 32-bit integer for Unix time (the number of seconds since January 1, 1970, as a 32-bit value), or the number of seconds starting from the current time. For the latter, the number of seconds may not exceed 60*60*24*30 (the number of seconds in 30 days). If the number sent by a client is larger, the server will consider it to be a real Unix time value rather than an offset from the current time. If you do not use this column, set its value to 0 to indicate that it is unused.

- `flags`: Must be mapped to an `INTEGER` of at least 32-bits and can be NULL. If you do not use this column, set its value to `0` to indicate that it is unused.

A pre-check is performed at plugin load time to enforce column constraints. If any mismatches are found, the plugin will not load.

cache_policies Table

Table `cache_policies` specifies whether to use `InnoDB` as the data store of `memcached` (`innodb_only`), or to use the traditional `memcached` engine as the backstore (`cache-only`), or both (`caching`). In the last case, if `memcached` cannot find a key in memory, it searches for the value in an `InnoDB` table.

config_options Table

Table `config_options` stores `memcached`-related settings that are appropriate to change at runtime, through SQL. Currently, MySQL supports the following configuration options through this table:

`separator`: The separator used to separate values of a long string into smaller values for multiple columns values. By default, this is the `|` character. For example, if you defined `col1, col2` as value columns, And you define `|` as separator, you could issue the following command in `memcached` to insert values into `col1` and `col2` respectively:

```
set keyx 10 0 19
valuecolx|valuecoly
```

So `valuecol1x` is stored in `col1` and `valuecoly` is stored in `col2`.

`table_map_delimiter`: The character separating the schema name and the table name when you use the `@@` notation in a key name to access a key in a specific table. For example, `@@t1.some_key` and `@@t2.some_key` have the same key value, but are stored in different tables and so do not conflict.

Multiple-column Mapping

- During plugin initialization, when `InnoDB` `memcached` is configured with information defined in the `containers` table, each mapped column that is parsed from `value_columns` is verified against the mapped table. If multiple columns are mapped, there is a check to ensure that each column exists and is the right type.

- At run-time, for `memcached` insert operations, if there are more delimiters in the value than the number of mapped columns, only the number of mapped values are taken. For example, if there are 6 mapped columns and 7 delimited values are provided, only the first 6 delimited values are taken. The 7th delimited value is ignored.

- If there are fewer delimited values than mapped columns, unfilled columns are set to NULL. If an unfilled column cannot be NULL, the insert will fail.

- If a table has more columns than mapped values, the extra columns do not affect output results.

Example Tables

The configuration script, `scripts/innodb_memcached_config.sql`, creates a table `demo_test` in the `test` database as an example. It also allows the Daemon Memcached to work immediately, without creating any additional tables.

The entries in the container table define which column is used for what purpose as described above:

```
mysql> select * from innodb_memcache.containers;
+------+-----------+-----------+-------------+---------------+-------+------------+--------------------+------
| name | db_schema | db_table  | key_columns | value_columns | flags | cas_column | expire_time_column | uniqu
+------+-----------+-----------+-------------+---------------+-------+------------+--------------------+------
| aaa  | test      | demo_test | c1          | c2            | c3    | c4         | c5                 | PRIMA
+------+-----------+-----------+-------------+---------------+-------+------------+--------------------+------
1 row in set (0.00 sec)

mysql> desc test.demo_test;
+-------+---------------------+------+-----+---------+-------+
| Field | Type                | Null | Key | Default | Extra |
+-------+---------------------+------+-----+---------+-------+
| c1    | varchar(32)         | NO   | PRI |         |       |
| c2    | varchar(1024)       | YES  |     | NULL    |       |
| c3    | int(11)             | YES  |     | NULL    |       |
| c4    | bigint(20) unsigned | YES  |     | NULL    |       |
| c5    | int(11)             | YES  |     | NULL    |       |
+-------+---------------------+------+-----+---------+-------+
5 rows in set (0.01 sec)
```

When no table ID is requested through the @@ notation in the key name:

- If a row has a name value of default, the corresponding table is used by the memcached plugin. Thus, when you make your first entry in innodb_memcache.containers to move beyond the demo_test table, use a name value of default.

- If there is no innodb_memcache.containers.name value of default, the row with the first name value in alphabetical order is used.

14.2.16.8 Troubleshooting the InnoDB memcached Plugin

The following list shows some potential issues you might encounter using the InnoDB memcached daemon, and solutions or workarounds where available:

- If you see this error in your MySQL error log, the server might fail to start:

  ```
  failed to set rlimit for open files. Try running as root or requesting
  smaller maxconns value.
  ```

  The error message is actually from the memcached daemon. One solution is to raise the OS limit for the number of open files. The command varies depending on the operating system. For example, here are the commands to check and increase the limit on several operating systems:

  ```
  # Linux
  $ ulimit -n
  1024
  ulimit -n 4096
  $ ulimit -n
  4096

  # OS X Lion (10.6)
  $ ulimit -n
  256
  ulimit -n 4096
  $ ulimit -n
  ```

```
4096
```

The other solution is to reduce the number of concurrent connections available for the memcached daemon, using the -c option which defaults to 1024. Encode that memcached option using the MySQL option daemon_memcached_option inside the MySQL configuration file:

```

[mysqld]
...
loose-daemon_memcached_option='-c 64'
```

- To troubleshoot problems where the memcached daemon is unable to store data in or retrieve data from the InnoDB table, specify the memcached option -vvv through the MySQL configuration option daemon_memcached_option. Examine the MySQL error log for debug output related to memcached operations.

- If the column specified to hold the memcached item values is the wrong data type, such as a numeric type instead of a string type, attempts to store key/value pairs will fail with no specific error code or message.

- If the daemon_memcached plugin causes any issues with starting the MySQL server, disable it during troubleshooting by adding this line under the [mysqld] group in your MySQL configuration file:

  ```
  daemon_memcached=OFF
  ```

  For example, if you run the install plugin command before running the scripts/innodb_memcached_config.sql script to set up the necessary database and tables, the server might crash and be unable to start. Or, if you set up an incorrect entry in the innodb_memcache.containers table, the server might be unable to start.

  To permanently turn off the memcached plugin for a MySQL instance, issue the following command:

  ```

  mysql> uninstall plugin daemon_memcached;
  ```

- If you run more than one instance of MySQL on the same machine, with the memcached daemon plugin enabled in each, make sure to specify a unique memcached port for each one using the daemon_memcached_option configuration option.

- You might find that a SQL statement cannot find an expected table, or there is no data in the table, but memcached API calls still work and retrieve the expected data. This can happen if you do not set up the entry in the innodb_memcache.containers table, or do not switch to that table by issuing a GET or SET request with the key *@@table_id*, or make a change to an existing entry in innodb_memcache.containers without restarting the MySQL server afterward. The free-form storage mechanism is flexible enough that your requests to store or retrieve a multi-column value like col1|col2|col3 will usually still work, even if the daemon is using the test.demo_test table which stores all the data within a single column.

- When defining your own InnoDB table for use with InnoDB memcached, and columns in your table are defined as NOT NULL, ensure that values are supplied for the NOT NULL columns when inserting a descriptor for the InnoDB table into the memcached containers table (innodb_memcached.containers). If your descriptor INSERT statement contains fewer delimited values than there are mapped columns, unfilled columns are set to NULL. Attempting to insert a NULL value into a NOT NULL column causes the INSERT to fail, which may only become evident after you reinitialize the InnoDB memcached plugin to apply changes to the containers table.

- If cas_column and expire_time_column of the innodb_memcached.containers table are set to NULL, the following error will be returned when attempting to load the memcached plugin:

  ```
  InnoDB_Memcached: column 6 in the entry for config table 'containers' in
  database 'innodb_memcache' has an invalid NULL value.
  ```

  The memcached plugin rejects usage of NULL in the cas_column and expire_time_column columns. Set the value of these columns to 0 if the columns are unused.

- As the length of the memcached key and values increase, you encounter size and length limits at different points:

  - When the key exceeds 250 bytes in size, memcached operations return an error. This is currently a fixed limit within memcached.

  - You might encounter InnoDB-related limits when the value exceeds 768 bytes in size, or 3072 bytes in size, or 1/2 of the size specified by innodb_page_size. These limits primarily apply if you intend to create an index on the value column to run report-generating queries on that column from SQL. See Section 14.2.6.7, “Limits on InnoDB Tables” for details.

  - The maximum size for the combination of the key and the value is 1 MB.

- If you share configuration files across MySQL servers with different versions, using the latest configuration options for the memcached plugin could cause startup errors for older MySQL versions. To avoid compatibility problems, use the loose forms of these option names, for example loose-daemon_memcached_option='-c 64' instead of daemon_memcached_option='-c 64'.

- There is no restriction or check in place to validate character set settings. memcached stores and retrieves keys and values in bytes and is therefore not character set sensitive. However, you must ensure that the memcached client and the MySQL table use the same character set.

14.2.17 InnoDB Troubleshooting

The following general guidelines apply to troubleshooting InnoDB problems:

- When an operation fails or you suspect a bug, look at the MySQL server error log (see Section 5.2.2, “The Error Log”).

- If the failure is related to a deadlock, run with the innodb_print_all_deadlocks option enabled so that details about each InnoDB deadlock are printed to the MySQL server error log.

- Issues relating to the InnoDB data dictionary include failed CREATE TABLE statements (orphaned table files), inability to open .InnoDB files, and system cannot find the path specified errors. For information about these sorts of problems and errors, see Section 14.2.17.3, “Troubleshooting InnoDB Data Dictionary Operations”.

- When troubleshooting, it is usually best to run the MySQL server from the command prompt, rather than through mysqld_safe or as a Windows service. You can then see what mysqld prints to the console, and so have a better grasp of what is going on. On Windows, start mysqld with the --console option to direct the output to the console window.

- Enable the InnoDB Monitors to obtain information about a problem (see Section 14.2.12.4, “InnoDB Monitors”). If the problem is performance-related, or your server appears to be hung, you should enable the standard Monitor to print information about the internal state of InnoDB. If the problem is with locks, enable the Lock Monitor. If the problem is in creation of tables or other data dictionary operations, enable the Table Monitor to print the contents of the InnoDB internal data dictionary. To see tablespace information enable the Tablespace Monitor.

InnoDB temporarily enables standard InnoDB Monitor output under the following conditions:

- A long semaphore wait
- InnoDB cannot find free blocks in the buffer pool
- Over 67% of the buffer pool is occupied by lock heaps or the adaptive hash index

- If you suspect that a table is corrupt, run CHECK TABLE on that table.

14.2.17.1 Troubleshooting InnoDB I/O Problems

The troubleshooting steps for InnoDB I/O problems depend on when the problem occurs: during startup of the MySQL server, or during normal operations when a DML or DDL statement fails due to problems at the file system level.

Initialization Problems

If something goes wrong when InnoDB attempts to initialize its tablespace or its log files, delete all files created by InnoDB: all ibdata files and all ib_logfile files. If you already created some InnoDB tables, also delete the corresponding .frm files for these tables, and any .ibd files if you are using multiple tablespaces, from the MySQL database directories. Then try the InnoDB database creation again. For easiest troubleshooting, start the MySQL server from a command prompt so that you see what is happening.

Runtime Problems

If InnoDB prints an operating system error during a file operation, usually the problem has one of the following solutions:

- Make sure the InnoDB data file directory and the InnoDB log directory exist.
- Make sure mysqld has access rights to create files in those directories.
- Make sure mysqld can read the proper my.cnf or my.ini option file, so that it starts with the options that you specified.
- Make sure the disk is not full and you are not exceeding any disk quota.
- Make sure that the names you specify for subdirectories and data files do not clash.
- Doublecheck the syntax of the innodb_data_home_dir and innodb_data_file_path values. In particular, any MAX value in the innodb_data_file_path option is a hard limit, and exceeding that limit causes a fatal error.

14.2.17.2 Starting InnoDB on a Corrupted Database

To investigate database page corruption, you might dump your tables from the database with SELECT ... INTO OUTFILE. Usually, most of the data obtained in this way is intact. Serious corruption might cause SELECT * FROM *tbl_name* statements or InnoDB background operations to crash or assert, or even cause InnoDB roll-forward recovery to crash. In such cases, use the innodb_force_recovery option to force the InnoDB storage engine to start up while preventing background operations from running, so that you can dump your tables. For example, you can add the following line to the [mysqld] section of your option file before restarting the server:

```
[mysqld]
innodb_force_recovery = 1
```

Warning

Before using innodb_force_recovery ensure that you have a backup copy of your database in case you need to start over. You should always begin by setting innodb_force_recovery to a lower value. Incrementally increase the setting as required. Only use an innodb_force_recovery setting of 3 or greater on a production server instance after you have successfully tested the setting on separate physical copy of your database. As of 5.7.3, an innodb_force_recovery setting of 4 or greater places InnoDB in read-only mode.

innodb_force_recovery is 0 by default (normal startup without forced recovery). The permissible nonzero values for innodb_force_recovery are 1 to 6. A larger value includes the functionality of lesser values. For example, a value of 3 includes all of the functionality 1 and 2. If you are able to dump your tables with an option value of at most 3, then you are relatively safe that only some data on corrupt individual pages is lost. A value of 6 is considered drastic because database pages are left in an obsolete state, which in turn may introduce more corruption into B-trees and other database structures.

- 1 (SRV_FORCE_IGNORE_CORRUPT)

  Lets the server run even if it detects a corrupt page. Tries to make SELECT * FROM *tbl_name* jump over corrupt index records and pages, which helps in dumping tables.

- 2 (SRV_FORCE_NO_BACKGROUND)

  Prevents the master thread and any purge threads from running. If a crash would occur during the purge operation, this recovery value prevents it.

- 3 (SRV_FORCE_NO_TRX_UNDO)

  Does not run transaction rollbacks after crash recovery.

- 4 (SRV_FORCE_NO_IBUF_MERGE)

  Prevents insert buffer merge operations. If they would cause a crash, does not do them. Does not calculate table statistics. As of MySQL 5.7.3, sets InnoDB to read-only.

- 5 (SRV_FORCE_NO_UNDO_LOG_SCAN)

  Does not look at undo logs when starting the database: InnoDB treats even incomplete transactions as committed. As of MySQL 5.7.3, sets InnoDB to read-only.

- 6 (SRV_FORCE_NO_LOG_REDO)

  Does not do the redo log roll-forward in connection with recovery. As of MySQL 5.7.3, sets InnoDB to read-only.

  With this value, you might not be able to do queries other than a basic SELECT * FROM t, with no WHERE, ORDER BY, or other clauses. More complex queries could encounter corrupted data structures and fail.

If corruption within the table data prevents you from dumping the entire table contents, a query with an ORDER BY *primary_key* DESC clause might be able to dump the portion of the table after the corrupted part.

The database must not otherwise be used with any nonzero value of innodb_force_recovery. As a safety measure, InnoDB prevents INSERT, UPDATE, or DELETE operations when innodb_force_recovery is greater than 0. Also, as of 5.7.3, an innodb_force_recovery setting of 4 or greater places InnoDB in read-only mode.

You can SELECT from tables to dump them, or DROP or CREATE tables even if forced recovery is used. If you know that a given table is causing a crash on rollback, you can drop it. You can also use this to stop a runaway rollback caused by a failing mass import or ALTER TABLE: kill the mysqld process and set innodb_force_recovery to 3 to bring the database up without the rollback, then DROP the table that is causing the runaway rollback.

Note

As of MySQL 5.7.3, an innodb_force_recovery setting of 4 or greater places InnoDB into read-only mode, which means that you can only DROP or CREATE tables when using forced recovery if the innodb_force_recovery setting is less 4.

14.2.17.3 Troubleshooting InnoDB Data Dictionary Operations

Information about table definitions is stored both in the .frm files, and in the InnoDB data dictionary. If you move .frm files around, or if the server crashes in the middle of a data dictionary operation, these sources of information can become inconsistent.

Problem with CREATE TABLE

A symptom of an out-of-sync data dictionary is that a CREATE TABLE statement fails. If this occurs, look in the server's error log. If the log says that the table already exists inside the InnoDB internal data dictionary, you have an orphaned table inside the InnoDB tablespace files that has no corresponding .frm file. The error message looks like this:

```
InnoDB: Error: table test/parent already exists in InnoDB internal
InnoDB: data dictionary. Have you deleted the .frm file
InnoDB: and not used DROP TABLE? Have you used DROP DATABASE
InnoDB: for InnoDB tables in MySQL version <= 3.23.43?
InnoDB: See the Restrictions section of the InnoDB manual.
InnoDB: You can drop the orphaned table inside InnoDB by
InnoDB: creating an InnoDB table with the same name in another
InnoDB: database and moving the .frm file to the current database.
InnoDB: Then MySQL thinks the table exists, and DROP TABLE will
InnoDB: succeed.
```

You can drop the orphaned table by following the instructions given in the error message. If you are still unable to use DROP TABLE successfully, the problem may be due to name completion in the mysql client. To work around this problem, start the mysql client with the --skip-auto-rehash option and try DROP TABLE again. (With name completion on, mysql tries to construct a list of table names, which fails when a problem such as just described exists.)

Problem Opening Table

Another symptom of an out-of-sync data dictionary is that MySQL prints an error that it cannot open a .InnoDB file:

```
ERROR 1016: Can't open file: 'child2.InnoDB'. (errno: 1)
```

In the error log you can find a message like this:

```
InnoDB: Cannot find table test/child2 from the internal data dictionary
InnoDB: of InnoDB though the .frm file for the table exists. Maybe you
InnoDB: have deleted and recreated InnoDB data files but have forgotten
InnoDB: to delete the corresponding .frm files of InnoDB tables?
```

This means that there is an orphaned `.frm` file without a corresponding table inside `InnoDB`. You can drop the orphaned `.frm` file by deleting it manually.

Orphaned Intermediate Tables

If MySQL crashes in the middle of an `ALTER TABLE` operation, you may end up with an orphaned intermediate table inside the `InnoDB` tablespace. Orphaned intermediate table names begin with an `#sql-` prefix. You can view a list of tables that are present in the `InnoDB` tablespace, including tables named with an `#sql-` prefix, using the Table Monitor.

`ALTER TABLE` creates intermediate table files in the same directory as the original table.

The following example uses the `salaries` table of the `employees` sample database.

```
mysql> SHOW CREATE TABLE salaries\G
*************************** 1. row ***************************
       Table: salaries
Create Table: CREATE TABLE `salaries` (
  `emp_no` int(11) NOT NULL,
  `salary` int(11) NOT NULL,
  `from_date` date NOT NULL,
  `to_date` date NOT NULL,
  PRIMARY KEY (`emp_no`,`from_date`),
  KEY `emp_no` (`emp_no`),
  CONSTRAINT `salaries_ibfk_1` FOREIGN KEY (`emp_no`) REFERENCES `employees` (`emp_no`) ON DELETE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=latin1
1 row in set (0.00 sec)
```

The MySQL server connection is lost while performing an `ALTER TABLE` operation to remove the `to_date` column from the `employees.salaries` table:

```
mysql> ALTER TABLE salaries DROP COLUMN to_date;
ERROR 2013 (HY000): Lost connection to MySQL server during query
mysql> use employees;
No connection. Trying to reconnect...
Connection id:    1
Current database: *** NONE ***
```

The `ALTER TABLE` operation leaves an orphaned intermediate table (`#sql-ib87.ibd`) and an accompanying table format file (`#sql-22d0_1.frm`):

```
mysql> \! ls /path/to/datadir/employees/
db.opt           dept_emp.ibd      employees.ibd    #sql-ib87.ibd
departments.frm  dept_manager.frm  salaries.frm     titles.frm
departments.ibd  dept_manager.ibd  salaries.ibd     titles.ibd
dept_emp.frm     employees.frm     #sql-22d0_1.frm
```

To remove the orphaned intermediate table, perform the following steps:

1. In the directory where the intermediate table resides, rename the `#sql-*.frm` file to match the name of the `#sql-*.ibd` file.

   ```
   shell> mv "#sql-247a_2.frm" "#sql-ib87.frm"
   ```

Note

To rename or copy a file in the Unix shell, you must enclose the file name in double quotation marks if the file name contains “#”.

2. Drop the intermediate table by issuing a DROP TABLE statement, prefixing the name of the table with #mysql50# and enclosing table name in backticks. (The #mysql50# prefix prevents MySQL from escaping the hash mark and hyphen.)

   ```
   mysql> DROP TABLE `#mysql50##sql-ib87`;
   Query OK, 0 rows affected (0.01 sec)
   ```

Note

You can perform SQL statements on tables whose name contains the character “#” if you enclose the name within backticks.

If you have number of orphaned intermediate tables that have accumulated over time, you may need to look at the date modified for each file to match temporary table files (.ibd files) with intermediate table format files (.frm files), as the file names may not match. If the intermediate table format file (.frm file) is not available, create one using the following steps:

1. In some other database directory, create a table with the structure that the table would have if the ALTER TABLE operation completed successfully:

   ```
   mysql> CREATE TABLE tmp LIKE salaries; ALTER TABLE tmp DROP COLUMN to_date;
   Query OK, 0 rows affected (0.02 sec)

   Query OK, 0 rows affected (0.06 sec)
   Records: 0  Duplicates: 0  Warnings: 0
   ```

2. Shut down MySQL server.

3. Copy the tmp.frm file that you created and rename it so that it matches the #sql-*.ibd file name.

   ```
   shell> cp tmp.frm employees/#sql-ib87.frm
   ```

4. Once the .frm file is in place, issue DROP TABLE for the intermediate table, as described above.

Problem with Missing Tablespace

With innodb_file_per_table enabled, the following message might occur if the .frm or .ibd files (or both) are missing:

```
InnoDB: in InnoDB data dictionary has tablespace id N,
InnoDB: but tablespace with that id or name does not exist. Have
InnoDB: you deleted or moved .ibd files?
InnoDB: This may also be a table created with CREATE TEMPORARY TABLE
InnoDB: whose .ibd and .frm files MySQL automatically removed, but the
InnoDB: table still exists in the InnoDB internal data dictionary.
```

If this occurs, try the following procedure to resolve the problem:

1. Create a matching .frm file in some other database directory and copy it to the database directory where the orphan table is located.

2. Issue DROP TABLE for the original table. That should successfully drop the table and InnoDB should print a warning to the error log that the .ibd file was missing.

14.2.17.4 InnoDB Error Handling

Error handling in InnoDB is not always the same as specified in the SQL standard. According to the standard, any error during an SQL statement should cause rollback of that statement. InnoDB sometimes rolls back only part of the statement, or the whole transaction. The following items describe how InnoDB performs error handling:

- If you run out of file space in a tablespace, a MySQL Table is full error occurs and InnoDB rolls back the SQL statement.
- A transaction deadlock causes InnoDB to roll back the entire transaction. Retry the whole transaction when this happens.

  A lock wait timeout causes InnoDB to roll back only the single statement that was waiting for the lock and encountered the timeout. (To have the entire transaction roll back, start the server with the --innodb_rollback_on_timeout option.) Retry the statement if using the current behavior, or the entire transaction if using --innodb_rollback_on_timeout.

  Both deadlocks and lock wait timeouts are normal on busy servers and it is necessary for applications to be aware that they may happen and handle them by retrying. You can make them less likely by doing as little work as possible between the first change to data during a transaction and the commit, so the locks are held for the shortest possible time and for the smallest possible number of rows. Sometimes splitting work between different transactions may be practical and helpful.

  When a transaction rollback occurs due to a deadlock or lock wait timeout, it cancels the effect of the statements within the transaction. But if the start-transaction statement was START TRANSACTION or BEGIN statement, rollback does not cancel that statement. Further SQL statements become part of the transaction until the occurrence of COMMIT, ROLLBACK, or some SQL statement that causes an implicit commit.
- A duplicate-key error rolls back the SQL statement, if you have not specified the IGNORE option in your statement.
- A row too long error rolls back the SQL statement.
- Other errors are mostly detected by the MySQL layer of code (above the InnoDB storage engine level), and they roll back the corresponding SQL statement. Locks are not released in a rollback of a single SQL statement.

During implicit rollbacks, as well as during the execution of an explicit ROLLBACK SQL statement, SHOW PROCESSLIST displays Rolling back in the State column for the relevant connection.

14.2.17.5 InnoDB Error Codes

The following is a nonexhaustive list of common InnoDB-specific errors that you may encounter, with information about why each occurs and how to resolve the problem.

- 1005 (ER_CANT_CREATE_TABLE)

  Cannot create table. If the error message refers to error 150, table creation failed because a foreign key constraint was not correctly formed. If the error message refers to error –1, table creation probably failed because the table includes a column name that matched the name of an internal InnoDB table.
- 1016 (ER_CANT_OPEN_FILE)

  Cannot find the InnoDB table from the InnoDB data files, although the .frm file for the table exists. See Section 14.2.17.3, “Troubleshooting InnoDB Data Dictionary Operations”.

- 1114 (ER_RECORD_FILE_FULL)

  InnoDB has run out of free space in the tablespace. Reconfigure the tablespace to add a new data file.

- 1205 (ER_LOCK_WAIT_TIMEOUT)

  Lock wait timeout expired. The statement that waited too long was rolled back (not the entire transaction). You can increase the value of the innodb_lock_wait_timeout configuration option if SQL statements should wait longer for other transactions to complete, or decrease it if too many long-running transactions are causing locking problems and reducing concurrency on a busy system.

- 1206 (ER_LOCK_TABLE_FULL)

  The total number of locks exceeds the amount of memory InnoDB devotes to managing locks. To avoid this error, increase the value of innodb_buffer_pool_size. Within an individual application, a workaround may be to break a large operation into smaller pieces. For example, if the error occurs for a large INSERT, perform several smaller INSERT operations.

- 1213 (ER_LOCK_DEADLOCK)

  The transaction encountered a deadlock and was automatically rolled back so that your application could take corrective action. To recover from this error, run all the operations in this transaction again. A deadlock occurs when requests for locks arrive in inconsistent order between transactions. The transaction that was rolled back released all its locks, and the other transaction can now get all the locks it requested. Thus when you re-run the transaction that was rolled back, it might have to wait for other transactions to complete, but typically the deadlock does not recur. If you encounter frequent deadlocks, make the sequence of locking operations (LOCK TABLES, SELECT ... FOR UPDATE, and so on) consistent between the different transactions or applications that experience the issue. See Section 14.2.2.11, “How to Cope with Deadlocks” for details.

- 1216 (ER_NO_REFERENCED_ROW)

  You are trying to add a row but there is no parent row, and a foreign key constraint fails. Add the parent row first.

- 1217 (ER_ROW_IS_REFERENCED)

  You are trying to delete a parent row that has children, and a foreign key constraint fails. Delete the children first.

- ERROR 1553 (HY000): Cannot drop index '*fooIdx*': needed in a foreign key constraint

  This error message is reported when you attempt to drop the last index that can enforce a particular referential constraint.

  For optimal performance with DML statements, InnoDB requires an index to exist on foreign key columns, so that UPDATE and DELETE operations on a parent table can easily check whether corresponding rows exist in the child table. MySQL creates or drops such indexes automatically when needed, as a side-effect of CREATE TABLE, CREATE INDEX, and ALTER TABLE statements.

  When you drop an index, InnoDB checks whether the index is not used for checking a foreign key constraint. It is still OK to drop the index if there is another index that can be used to enforce the same constraint. InnoDB prevents you from dropping the last index that can enforce a particular referential constraint.

14.2.17.6 Operating System Error Codes

To print the error message for an operating system error number, you can use one of the following options. The perror program is provided with the MySQL distribution.

```
$ perror 123
$ perl -MPOSIX -le 'print strerror 123'
$ python -c 'import os; print os.strerror(123)'
```

- **Linux System Error Codes**

  The following table provides a partial list of Linux system error codes.

Number	Macro	Description
1	EPERM	Operation not permitted
2	ENOENT	No such file or directory
3	ESRCH	No such process
4	EINTR	Interrupted system call
5	EIO	I/O error
6	ENXIO	No such device or address
7	E2BIG	Arg list too long
8	ENOEXEC	Exec format error
9	EBADF	Bad file number
10	ECHILD	No child processes
11	EAGAIN	Try again
12	ENOMEM	Out of memory
13	EACCES	Permission denied
14	EFAULT	Bad address
15	ENOTBLK	Block device required
16	EBUSY	Device or resource busy
17	EEXIST	File exists
18	EXDEV	Cross-device link
19	ENODEV	No such device
20	ENOTDIR	Not a directory
21	EISDIR	Is a directory
22	EINVAL	Invalid argument
23	ENFILE	File table overflow
24	EMFILE	Too many open files
25	ENOTTY	Inappropriate ioctl for device
26	ETXTBSY	Text file busy
27	EFBIG	File too large
28	ENOSPC	No space left on device
29	ESPIPE	File descriptor does not allow seeking
30	EROFS	Read-only file system

Number	Macro	Description
31	EMLINK	Too many links

- **Windows System Error Codes**

The following table provides a list of some common Windows system error codes. For a complete list, see the Microsoft Web site.

Number	Macro	Description
1	ERROR_INVALID_FUNCTION	Incorrect function.
2	ERROR_FILE_NOT_FOUND	The system cannot find the file specified.
3	ERROR_PATH_NOT_FOUND	The system cannot find the path specified.
4	ERROR_TOO_MANY_OPEN_FILES	The system cannot open the file.
5	ERROR_ACCESS_DENIED	Access is denied.
6	ERROR_INVALID_HANDLE	The handle is invalid.
7	ERROR_ARENA_TRASHED	The storage control blocks were destroyed.
8	ERROR_NOT_ENOUGH_MEMORY	Not enough storage is available to process this command.
9	ERROR_INVALID_BLOCK	The storage control block address is invalid.
10	ERROR_BAD_ENVIRONMENT	The environment is incorrect.
11	ERROR_BAD_FORMAT	An attempt was made to load a program with an incorrect format.
12	ERROR_INVALID_ACCESS	The access code is invalid.
13	ERROR_INVALID_DATA	The data is invalid.
14	ERROR_OUTOFMEMORY	Not enough storage is available to complete this operation.
15	ERROR_INVALID_DRIVE	The system cannot find the drive specified.
16	ERROR_CURRENT_DIRECTORY	The directory cannot be removed.
17	ERROR_NOT_SAME_DEVICE	The system cannot move the file to a different disk drive.
18	ERROR_NO_MORE_FILES	There are no more files.
19	ERROR_WRITE_PROTECT	The media is write protected.
20	ERROR_BAD_UNIT	The system cannot find the device specified.
21	ERROR_NOT_READY	The device is not ready.
22	ERROR_BAD_COMMAND	The device does not recognize the command.
23	ERROR_CRC	Data error (cyclic redundancy check).
24	ERROR_BAD_LENGTH	The program issued a command but the command length is incorrect.
25	ERROR_SEEK	The drive cannot locate a specific area or track on the disk.
26	ERROR_NOT_DOS_DISK	The specified disk or diskette cannot be accessed.
27	ERROR_SECTOR_NOT_FOUND	The drive cannot find the sector requested.
28	ERROR_OUT_OF_PAPER	The printer is out of paper.
29	ERROR_WRITE_FAULT	The system cannot write to the specified device.
30	ERROR_READ_FAULT	The system cannot read from the specified device.
31	ERROR_GEN_FAILURE	A device attached to the system is not functioning.

Number	Macro	Description
32	ERROR_SHARING_VIOLATION	The process cannot access the file because it is being used by another process.
33	ERROR_LOCK_VIOLATION	The process cannot access the file because another process has locked a portion of the file.
34	ERROR_WRONG_DISK	The wrong diskette is in the drive. Insert %2 (Volume Serial Number: %3) into drive %1.
36	ERROR_SHARING_BUFFER_EXCEEDED	Too many files opened for sharing.
38	ERROR_HANDLE_EOF	Reached the end of the file.
39	ERROR_HANDLE_DISK_FULL	The disk is full.
87	ERROR_INVALID_PARAMETER	The parameter is incorrect.
112	ERROR_DISK_FULL	The disk is full.
123	ERROR_INVALID_NAME	The file name, directory name, or volume label syntax is incorrect.
1450	ERROR_NO_SYSTEM_RESOURCES	Insufficient system resources exist to complete the requested service.

14.3 The MyISAM Storage Engine

MyISAM is based on the older (and no longer available) ISAM storage engine but has many useful extensions.

Table 14.12 MyISAM Storage Engine Features

Storage limits	256TB	**Transactions**	No	**Locking granularity**	Table
MVCC	No	**Geospatial data type support**	Yes	**Geospatial indexing support**	Yes
B-tree indexes	Yes	**T-tree indexes**	No	**Hash indexes**	No
Full-text search indexes	Yes	**Clustered indexes**	No	**Data caches**	No
Index caches	Yes	**Compressed data**	Yes[a]	**Encrypted data**[b]	Yes
Cluster database support	No	**Replication support**[c]	Yes	**Foreign key support**	No
Backup / point-in-time recovery[d]	Yes	**Query cache support**	Yes	**Update statistics for data dictionary**	Yes

[a] Compressed MyISAM tables are supported only when using the compressed row format. Tables using the compressed row format with MyISAM are read only.

[b] Implemented in the server (via encryption functions), rather than in the storage engine.

[c] Implemented in the server, rather than in the storage engine.

[d] Implemented in the server, rather than in the storage engine.

Each MyISAM table is stored on disk in three files. The files have names that begin with the table name and have an extension to indicate the file type. An .frm file stores the table format. The data file has an .MYD (MYData) extension. The index file has an .MYI (MYIndex) extension.

To specify explicitly that you want a MyISAM table, indicate that with an ENGINE table option:

```
CREATE TABLE t (i INT) ENGINE = MYISAM;
```

In MySQL 5.7, it is normally necessary to use ENGINE to specify the MyISAM storage engine because InnoDB is the default engine.

You can check or repair MyISAM tables with the mysqlcheck client or myisamchk utility. You can also compress MyISAM tables with myisampack to take up much less space. See Section 4.5.3, “mysqlcheck — A Table Maintenance Program”, Section 4.6.3, “myisamchk — MyISAM Table-Maintenance Utility”, and Section 4.6.5, “myisampack — Generate Compressed, Read-Only MyISAM Tables”.

MyISAM tables have the following characteristics:

- All data values are stored with the low byte first. This makes the data machine and operating system independent. The only requirements for binary portability are that the machine uses two's-complement signed integers and IEEE floating-point format. These requirements are widely used among mainstream machines. Binary compatibility might not be applicable to embedded systems, which sometimes have peculiar processors.

  There is no significant speed penalty for storing data low byte first; the bytes in a table row normally are unaligned and it takes little more processing to read an unaligned byte in order than in reverse order. Also, the code in the server that fetches column values is not time critical compared to other code.

- All numeric key values are stored with the high byte first to permit better index compression.

- Large files (up to 63-bit file length) are supported on file systems and operating systems that support large files.

- There is a limit of $(2^{32})^2$ (1.844E+19) rows in a MyISAM table.

- The maximum number of indexes per MyISAM table is 64.

  The maximum number of columns per index is 16.

- The maximum key length is 1000 bytes. This can also be changed by changing the source and recompiling. For the case of a key longer than 250 bytes, a larger key block size than the default of 1024 bytes is used.

- When rows are inserted in sorted order (as when you are using an AUTO_INCREMENT column), the index tree is split so that the high node only contains one key. This improves space utilization in the index tree.

- Internal handling of one AUTO_INCREMENT column per table is supported. MyISAM automatically updates this column for INSERT and UPDATE operations. This makes AUTO_INCREMENT columns faster (at least 10%). Values at the top of the sequence are not reused after being deleted. (When an AUTO_INCREMENT column is defined as the last column of a multiple-column index, reuse of values deleted from the top of a sequence does occur.) The AUTO_INCREMENT value can be reset with ALTER TABLE or myisamchk.

- Dynamic-sized rows are much less fragmented when mixing deletes with updates and inserts. This is done by automatically combining adjacent deleted blocks and by extending blocks if the next block is deleted.

- MyISAM supports concurrent inserts: If a table has no free blocks in the middle of the data file, you can INSERT new rows into it at the same time that other threads are reading from the table. A free block can occur as a result of deleting rows or an update of a dynamic length row with more data than its current contents. When all free blocks are used up (filled in), future inserts become concurrent again. See Section 8.10.3, “Concurrent Inserts”.

- You can put the data file and index file in different directories on different physical devices to get more speed with the DATA DIRECTORY and INDEX DIRECTORY table options to CREATE TABLE. See Section 13.1.14, “CREATE TABLE Syntax”.

- BLOB and TEXT columns can be indexed.

- NULL values are permitted in indexed columns. This takes 0 to 1 bytes per key.
- Each character column can have a different character set. See Section 10.1, “Character Set Support”.
- There is a flag in the MyISAM index file that indicates whether the table was closed correctly. If mysqld is started with the --myisam-recover-options option, MyISAM tables are automatically checked when opened, and are repaired if the table wasn't closed properly.
- myisamchk marks tables as checked if you run it with the --update-state option. myisamchk --fast checks only those tables that don't have this mark.
- myisamchk --analyze stores statistics for portions of keys, as well as for entire keys.
- myisampack can pack BLOB and VARCHAR columns.

MyISAM also supports the following features:

- Support for a true VARCHAR type; a VARCHAR column starts with a length stored in one or two bytes.
- Tables with VARCHAR columns may have fixed or dynamic row length.
- The sum of the lengths of the VARCHAR and CHAR columns in a table may be up to 64KB.
- Arbitrary length UNIQUE constraints.

Additional Resources

- A forum dedicated to the MyISAM storage engine is available at http://forums.mysql.com/list.php?21.

14.3.1 MyISAM Startup Options

The following options to mysqld can be used to change the behavior of MyISAM tables. For additional information, see Section 5.1.3, “Server Command Options”.

Table 14.13 MyISAM Option/Variable Reference

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
bulk_insert_buffer_size	Yes	Yes	Yes		Both	Yes
concurrent_insert	Yes	Yes	Yes		Global	Yes
delay-key-write	Yes	Yes			Global	Yes
- *Variable*: delay_key_write			Yes		Global	Yes
have_rtree_keys			Yes		Global	No
key_buffer_size	Yes	Yes	Yes		Global	Yes
log-isam	Yes	Yes				
myisam-block-size	Yes	Yes				
myisam_data_pointer_size	Yes	Yes	Yes		Global	Yes
myisam_max_sort_file_size	Yes	Yes	Yes		Global	Yes
myisam_mmap_size	Yes	Yes	Yes		Global	No
myisam-recover-options	Yes	Yes				

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
- *Variable*: myisam_recover_options						
myisam_recover_options			Yes		Global	No
myisam_repair_threads	Yes	Yes	Yes		Both	Yes
myisam_sort_buffer_size	Yes	Yes	Yes		Both	Yes
myisam_stats_method	Yes	Yes	Yes		Both	Yes
myisam_use_mmap	Yes	Yes	Yes		Global	Yes
skip-concurrent-insert	Yes	Yes				
- *Variable*: concurrent_insert						
tmp_table_size	Yes	Yes	Yes		Both	Yes

- `--myisam-recover-options=*mode*`

  Set the mode for automatic recovery of crashed `MyISAM` tables.

- `--delay-key-write=ALL`

  Don't flush key buffers between writes for any `MyISAM` table.

  Note

  If you do this, you should not access `MyISAM` tables from another program (such as from another MySQL server or with `myisamchk`) when the tables are in use. Doing so risks index corruption. Using `--external-locking` does not eliminate this risk.

The following system variables affect the behavior of `MyISAM` tables. For additional information, see Section 5.1.4, “Server System Variables”.

- `bulk_insert_buffer_size`

  The size of the tree cache used in bulk insert optimization.

  Note

  This is a limit *per thread*!

- `myisam_max_sort_file_size`

  The maximum size of the temporary file that MySQL is permitted to use while re-creating a `MyISAM` index (during `REPAIR TABLE`, `ALTER TABLE`, or `LOAD DATA INFILE`). If the file size would be larger than this value, the index is created using the key cache instead, which is slower. The value is given in bytes.

- `myisam_sort_buffer_size`

  Set the size of the buffer used when recovering tables.

Automatic recovery is activated if you start `mysqld` with the `--myisam-recover-options` option. In this case, when the server opens a `MyISAM` table, it checks whether the table is marked as crashed or whether the open count variable for the table is not 0 and you are running the server with external locking disabled. If either of these conditions is true, the following happens:

- The server checks the table for errors.
- If the server finds an error, it tries to do a fast table repair (with sorting and without re-creating the data file).
- If the repair fails because of an error in the data file (for example, a duplicate-key error), the server tries again, this time re-creating the data file.
- If the repair still fails, the server tries once more with the old repair option method (write row by row without sorting). This method should be able to repair any type of error and has low disk space requirements.

If the recovery wouldn't be able to recover all rows from previously completed statements and you didn't specify `FORCE` in the value of the `--myisam-recover-options` option, automatic repair aborts with an error message in the error log:

```
Error: Couldn't repair table: test.g00pages
```

If you specify `FORCE`, a warning like this is written instead:

```
Warning: Found 344 of 354 rows when repairing ./test/g00pages
```

Note that if the automatic recovery value includes `BACKUP`, the recovery process creates files with names of the form *`tbl_name-datetime`*`.BAK`. You should have a `cron` script that automatically moves these files from the database directories to backup media.

14.3.2 Space Needed for Keys

`MyISAM` tables use B-tree indexes. You can roughly calculate the size for the index file as `(key_length+4)/0.67`, summed over all keys. This is for the worst case when all keys are inserted in sorted order and the table doesn't have any compressed keys.

String indexes are space compressed. If the first index part is a string, it is also prefix compressed. Space compression makes the index file smaller than the worst-case figure if a string column has a lot of trailing space or is a `VARCHAR` column that is not always used to the full length. Prefix compression is used on keys that start with a string. Prefix compression helps if there are many strings with an identical prefix.

In `MyISAM` tables, you can also prefix compress numbers by specifying the `PACK_KEYS=1` table option when you create the table. Numbers are stored with the high byte first, so this helps when you have many integer keys that have an identical prefix.

14.3.3 `MyISAM` Table Storage Formats

`MyISAM` supports three different storage formats. Two of them, fixed and dynamic format, are chosen automatically depending on the type of columns you are using. The third, compressed format, can be created only with the `myisampack` utility (see Section 4.6.5, “`myisampack` — Generate Compressed, Read-Only MyISAM Tables”).

When you use `CREATE TABLE` or `ALTER TABLE` for a table that has no `BLOB` or `TEXT` columns, you can force the table format to `FIXED` or `DYNAMIC` with the `ROW_FORMAT` table option.

See Section 13.1.14, “`CREATE TABLE` Syntax”, for information about `ROW_FORMAT`.

You can decompress (unpack) compressed MyISAM tables using myisamchk --unpack; see Section 4.6.3, "myisamchk — MyISAM Table-Maintenance Utility", for more information.

14.3.3.1 Static (Fixed-Length) Table Characteristics

Static format is the default for MyISAM tables. It is used when the table contains no variable-length columns (VARCHAR, VARBINARY, BLOB, or TEXT). Each row is stored using a fixed number of bytes.

Of the three MyISAM storage formats, static format is the simplest and most secure (least subject to corruption). It is also the fastest of the on-disk formats due to the ease with which rows in the data file can be found on disk: To look up a row based on a row number in the index, multiply the row number by the row length to calculate the row position. Also, when scanning a table, it is very easy to read a constant number of rows with each disk read operation.

The security is evidenced if your computer crashes while the MySQL server is writing to a fixed-format MyISAM file. In this case, myisamchk can easily determine where each row starts and ends, so it can usually reclaim all rows except the partially written one. Note that MyISAM table indexes can always be reconstructed based on the data rows.

> **Note**
>
> Fixed-length row format is only available for tables without BLOB or TEXT columns. Creating a table with these columns with an explicit ROW_FORMAT clause will not raise an error or warning; the format specification will be ignored.

Static-format tables have these characteristics:

- CHAR and VARCHAR columns are space-padded to the specified column width, although the column type is not altered. BINARY and VARBINARY columns are padded with 0x00 bytes to the column width.
- Very quick.
- Easy to cache.
- Easy to reconstruct after a crash, because rows are located in fixed positions.
- Reorganization is unnecessary unless you delete a huge number of rows and want to return free disk space to the operating system. To do this, use OPTIMIZE TABLE or myisamchk -r.
- Usually require more disk space than dynamic-format tables.

14.3.3.2 Dynamic Table Characteristics

Dynamic storage format is used if a MyISAM table contains any variable-length columns (VARCHAR, VARBINARY, BLOB, or TEXT), or if the table was created with the ROW_FORMAT=DYNAMIC table option.

Dynamic format is a little more complex than static format because each row has a header that indicates how long it is. A row can become fragmented (stored in noncontiguous pieces) when it is made longer as a result of an update.

You can use OPTIMIZE TABLE or myisamchk -r to defragment a table. If you have fixed-length columns that you access or change frequently in a table that also contains some variable-length columns, it might be a good idea to move the variable-length columns to other tables just to avoid fragmentation.

Dynamic-format tables have these characteristics:

- All string columns are dynamic except those with a length less than four.

- Each row is preceded by a bitmap that indicates which columns contain the empty string (for string columns) or zero (for numeric columns). Note that this does not include columns that contain NULL values. If a string column has a length of zero after trailing space removal, or a numeric column has a value of zero, it is marked in the bitmap and not saved to disk. Nonempty strings are saved as a length byte plus the string contents.
- Much less disk space usually is required than for fixed-length tables.
- Each row uses only as much space as is required. However, if a row becomes larger, it is split into as many pieces as are required, resulting in row fragmentation. For example, if you update a row with information that extends the row length, the row becomes fragmented. In this case, you may have to run OPTIMIZE TABLE or myisamchk -r from time to time to improve performance. Use myisamchk -ei to obtain table statistics.
- More difficult than static-format tables to reconstruct after a crash, because rows may be fragmented into many pieces and links (fragments) may be missing.
- The expected row length for dynamic-sized rows is calculated using the following expression:

```
3
+ (number of columns + 7) / 8
+ (number of char columns)
+ (packed size of numeric columns)
+ (length of strings)
+ (number of NULL columns + 7) / 8
```

  There is a penalty of 6 bytes for each link. A dynamic row is linked whenever an update causes an enlargement of the row. Each new link is at least 20 bytes, so the next enlargement probably goes in the same link. If not, another link is created. You can find the number of links using myisamchk -ed. All links may be removed with OPTIMIZE TABLE or myisamchk -r.

14.3.3.3 Compressed Table Characteristics

Compressed storage format is a read-only format that is generated with the myisampack tool. Compressed tables can be uncompressed with myisamchk.

Compressed tables have the following characteristics:

- Compressed tables take very little disk space. This minimizes disk usage, which is helpful when using slow disks (such as CD-ROMs).
- Each row is compressed separately, so there is very little access overhead. The header for a row takes up one to three bytes depending on the biggest row in the table. Each column is compressed differently. There is usually a different Huffman tree for each column. Some of the compression types are:
  - Suffix space compression.
  - Prefix space compression.
  - Numbers with a value of zero are stored using one bit.
  - If values in an integer column have a small range, the column is stored using the smallest possible type. For example, a BIGINT column (eight bytes) can be stored as a TINYINT column (one byte) if all its values are in the range from -128 to 127.
  - If a column has only a small set of possible values, the data type is converted to ENUM.
  - A column may use any combination of the preceding compression types.

- Can be used for fixed-length or dynamic-length rows.

Note

While a compressed table is read only, and you cannot therefore update or add rows in the table, DDL (Data Definition Language) operations are still valid. For example, you may still use DROP to drop the table, and TRUNCATE TABLE to empty the table.

14.3.4 MyISAM Table Problems

The file format that MySQL uses to store data has been extensively tested, but there are always circumstances that may cause database tables to become corrupted. The following discussion describes how this can happen and how to handle it.

14.3.4.1 Corrupted MyISAM Tables

Even though the MyISAM table format is very reliable (all changes to a table made by an SQL statement are written before the statement returns), you can still get corrupted tables if any of the following events occur:

- The mysqld process is killed in the middle of a write.
- An unexpected computer shutdown occurs (for example, the computer is turned off).
- Hardware failures.
- You are using an external program (such as myisamchk) to modify a table that is being modified by the server at the same time.
- A software bug in the MySQL or MyISAM code.

Typical symptoms of a corrupt table are:

- You get the following error while selecting data from the table:

```
Incorrect key file for table: '...'. Try to repair it
```

- Queries don't find rows in the table or return incomplete results.

You can check the health of a MyISAM table using the CHECK TABLE statement, and repair a corrupted MyISAM table with REPAIR TABLE. When mysqld is not running, you can also check or repair a table with the myisamchk command. See Section 13.7.2.2, “CHECK TABLE Syntax”, Section 13.7.2.5, “REPAIR TABLE Syntax”, and Section 4.6.3, “myisamchk — MyISAM Table-Maintenance Utility”.

If your tables become corrupted frequently, you should try to determine why this is happening. The most important thing to know is whether the table became corrupted as a result of a server crash. You can verify this easily by looking for a recent restarted mysqld message in the error log. If there is such a message, it is likely that table corruption is a result of the server dying. Otherwise, corruption may have occurred during normal operation. This is a bug. You should try to create a reproducible test case that demonstrates the problem. See Section C.5.4.2, “What to Do If MySQL Keeps Crashing”, and Section 22.4, “Debugging and Porting MySQL”.

14.3.4.2 Problems from Tables Not Being Closed Properly

Each MyISAM index file (.MYI file) has a counter in the header that can be used to check whether a table has been closed properly. If you get the following warning from CHECK TABLE or myisamchk, it means that this counter has gone out of sync:

```
clients are using or haven't closed the table properly
```

This warning doesn't necessarily mean that the table is corrupted, but you should at least check the table.

The counter works as follows:

- The first time a table is updated in MySQL, a counter in the header of the index files is incremented.
- The counter is not changed during further updates.
- When the last instance of a table is closed (because a `FLUSH TABLES` operation was performed or because there is no room in the table cache), the counter is decremented if the table has been updated at any point.
- When you repair the table or check the table and it is found to be okay, the counter is reset to zero.
- To avoid problems with interaction with other processes that might check the table, the counter is not decremented on close if it was zero.

In other words, the counter can become incorrect only under these conditions:

- A `MyISAM` table is copied without first issuing `LOCK TABLES` and `FLUSH TABLES`.
- MySQL has crashed between an update and the final close. (Note that the table may still be okay, because MySQL always issues writes for everything between each statement.)
- A table was modified by `myisamchk --recover` or `myisamchk --update-state` at the same time that it was in use by `mysqld`.
- Multiple `mysqld` servers are using the table and one server performed a `REPAIR TABLE` or `CHECK TABLE` on the table while it was in use by another server. In this setup, it is safe to use `CHECK TABLE`, although you might get the warning from other servers. However, `REPAIR TABLE` should be avoided because when one server replaces the data file with a new one, this is not known to the other servers.

  In general, it is a bad idea to share a data directory among multiple servers. See Section 5.3, "Running Multiple MySQL Instances on One Machine", for additional discussion.

14.4 The MEMORY Storage Engine

The `MEMORY` storage engine (formerly known as `HEAP`) creates special-purpose tables with contents that are stored in memory. Because the data is vulnerable to crashes, hardware issues, or power outages, only use these tables as temporary work areas or read-only caches for data pulled from other tables.

Table 14.14 `MEMORY` Storage Engine Features

Storage limits	RAM	**Transactions**	No	**Locking granularity**	Table
MVCC	No	**Geospatial data type support**	No	**Geospatial indexing support**	No
B-tree indexes	Yes	**T-tree indexes**	No	**Hash indexes**	Yes
Full-text search indexes	No	**Clustered indexes**	No	**Data caches**	N/A
Index caches	N/A	**Compressed data**	No	**Encrypted data**[a]	Yes
Cluster database support	No	**Replication support**[b]	Yes	**Foreign key support**	No

Backup / point-in-time recovery[c]	Yes	**Query cache support**	Yes	**Update statistics for data dictionary**	Yes

[a] Implemented in the server (via encryption functions), rather than in the storage engine.

[b] Implemented in the server, rather than in the storage engine.

[c] Implemented in the server, rather than in the storage engine.

When to Use MEMORY or MySQL Cluster. Developers looking to deploy applications that use the MEMORY storage engine for important, highly available, or frequently updated data should consider whether MySQL Cluster is a better choice. A typical use case for the MEMORY engine involves these characteristics:

- Operations involving transient, non-critical data such as session management or caching. When the MySQL server halts or restarts, the data in MEMORY tables is lost.
- In-memory storage for fast access and low latency. Data volume can fit entirely in memory without causing the operating system to swap out virtual memory pages.
- A read-only or read-mostly data access pattern (limited updates).

MySQL Cluster offers the same features as the MEMORY engine with higher performance levels, and provides additional features not available with MEMORY:

- Row-level locking and multiple-thread operation for low contention between clients.
- Scalability even with statement mixes that include writes.
- Optional disk-backed operation for data durability.
- Shared-nothing architecture and multiple-host operation with no single point of failure, enabling 99.999% availability.
- Automatic data distribution across nodes; application developers need not craft custom sharding or partitioning solutions.
- Support for variable-length data types (including BLOB and TEXT) not supported by MEMORY.

For a white paper with more detailed comparison of the MEMORY storage engine and MySQL Cluster, see Scaling Web Services with MySQL Cluster: An Alternative to the MySQL Memory Storage Engine. This white paper includes a performance study of the two technologies and a step-by-step guide describing how existing MEMORY users can migrate to MySQL Cluster.

Performance Characteristics

MEMORY performance is constrained by contention resulting from single-thread execution and table lock overhead when processing updates. This limits scalability when load increases, particularly for statement mixes that include writes.

Despite the in-memory processing for MEMORY tables, they are not necessarily faster than InnoDB tables on a busy server, for general-purpose queries, or under a read/write workload. In particular, the table locking involved with performing updates can slow down concurrent usage of MEMORY tables from multiple sessions.

Depending on the kinds of queries performed on a MEMORY table, you might create indexes as either the default hash data structure (for looking up single values based on a unique key), or a general-purpose B-tree data structure (for all kinds of queries involving equality, inequality, or range operators such as less than or greater than). The following sections illustrate the syntax for creating both kinds of indexes. A common performance issue is using the default hash indexes in workloads where B-tree indexes are more efficient.

Physical Characteristics of MEMORY Tables

The MEMORY storage engine associates each table with one disk file, which stores the table definition (not the data). The file name begins with the table name and has an extension of .frm.

MEMORY tables have the following characteristics:

- Space for MEMORY tables is allocated in small blocks. Tables use 100% dynamic hashing for inserts. No overflow area or extra key space is needed. No extra space is needed for free lists. Deleted rows are put in a linked list and are reused when you insert new data into the table. MEMORY tables also have none of the problems commonly associated with deletes plus inserts in hashed tables.
- MEMORY tables use a fixed-length row-storage format. Variable-length types such as VARCHAR are stored using a fixed length.
- MEMORY tables cannot contain BLOB or TEXT columns.
- MEMORY includes support for AUTO_INCREMENT columns.
- Non-TEMPORARY MEMORY tables are shared among all clients, just like any other non-TEMPORARY table.

DDL Operations for MEMORY Tables

To create a MEMORY table, specify the clause ENGINE=MEMORY on the CREATE TABLE statement.

```
CREATE TABLE t (i INT) ENGINE = MEMORY;
```

As indicated by the engine name, MEMORY tables are stored in memory. They use hash indexes by default, which makes them very fast for single-value lookups, and very useful for creating temporary tables. However, when the server shuts down, all rows stored in MEMORY tables are lost. The tables themselves continue to exist because their definitions are stored in .frm files on disk, but they are empty when the server restarts.

This example shows how you might create, use, and remove a MEMORY table:

```
mysql> CREATE TABLE test ENGINE=MEMORY
    ->     SELECT ip,SUM(downloads) AS down
    ->     FROM log_table GROUP BY ip;
mysql> SELECT COUNT(ip),AVG(down) FROM test;
mysql> DROP TABLE test;
```

The maximum size of MEMORY tables is limited by the max_heap_table_size system variable, which has a default value of 16MB. To enforce different size limits for MEMORY tables, change the value of this variable. The value in effect for CREATE TABLE, or a subsequent ALTER TABLE or TRUNCATE TABLE, is the value used for the life of the table. A server restart also sets the maximum size of existing MEMORY tables to the global max_heap_table_size value. You can set the size for individual tables as described later in this section.

Indexes

The MEMORY storage engine supports both HASH and BTREE indexes. You can specify one or the other for a given index by adding a USING clause as shown here:

```
CREATE TABLE lookup
    (id INT, INDEX USING HASH (id))
    ENGINE = MEMORY;
```

```
CREATE TABLE lookup
    (id INT, INDEX USING BTREE (id))
    ENGINE = MEMORY;
```

For general characteristics of B-tree and hash indexes, see Section 8.3.1, “How MySQL Uses Indexes”.

MEMORY tables can have up to 64 indexes per table, 16 columns per index and a maximum key length of 3072 bytes.

If a MEMORY table hash index has a high degree of key duplication (many index entries containing the same value), updates to the table that affect key values and all deletes are significantly slower. The degree of this slowdown is proportional to the degree of duplication (or, inversely proportional to the index cardinality). You can use a BTREE index to avoid this problem.

MEMORY tables can have nonunique keys. (This is an uncommon feature for implementations of hash indexes.)

Columns that are indexed can contain NULL values.

User-Created and Temporary Tables

MEMORY table contents are stored in memory, which is a property that MEMORY tables share with internal temporary tables that the server creates on the fly while processing queries. However, the two types of tables differ in that MEMORY tables are not subject to storage conversion, whereas internal temporary tables are:

- If an internal temporary table becomes too large, the server automatically converts it to on-disk storage, as described in Section 8.4.4, “How MySQL Uses Internal Temporary Tables”.
- User-created MEMORY tables are never converted to disk tables.

Loading Data

To populate a MEMORY table when the MySQL server starts, you can use the --init-file option. For example, you can put statements such as INSERT INTO ... SELECT or LOAD DATA INFILE into this file to load the table from a persistent data source. See Section 5.1.3, “Server Command Options”, and Section 13.2.6, “LOAD DATA INFILE Syntax”.

MEMORY Tables and Replication

A server's MEMORY tables become empty when it is shut down and restarted. If the server is a replication master, its slaves are not aware that these tables have become empty, so you see out-of-date content if you select data from the tables on the slaves. To synchronize master and slave MEMORY tables, when a MEMORY table is used on a master for the first time since it was started, a DELETE statement is written to the master's binary log, to empty the table on the slaves also. The slave still has outdated data in the table during the interval between the master's restart and its first use of the table. To avoid this interval when a direct query to the slave could return stale data, use the --init-file option to populate the MEMORY table on the master at startup.

Managing Memory Use

The server needs sufficient memory to maintain all MEMORY tables that are in use at the same time.

Memory is not reclaimed if you delete individual rows from a MEMORY table. Memory is reclaimed only when the entire table is deleted. Memory that was previously used for deleted rows is re-used for new rows within the same table. To free all the memory used by a MEMORY table when you no longer require its contents, execute DELETE or TRUNCATE TABLE to remove all rows, or remove the table altogether using DROP TABLE. To free up the memory used by deleted rows, use ALTER TABLE ENGINE=MEMORY to force a table rebuild.

The memory needed for one row in a MEMORY table is calculated using the following expression:

```
SUM_OVER_ALL_BTREE_KEYS(max_length_of_key + sizeof(char*) * 4)
+ SUM_OVER_ALL_HASH_KEYS(sizeof(char*) * 2)
+ ALIGN(length_of_row+1, sizeof(char*))
```

ALIGN() represents a round-up factor to cause the row length to be an exact multiple of the char pointer size. sizeof(char*) is 4 on 32-bit machines and 8 on 64-bit machines.

As mentioned earlier, the max_heap_table_size system variable sets the limit on the maximum size of MEMORY tables. To control the maximum size for individual tables, set the session value of this variable before creating each table. (Do not change the global max_heap_table_size value unless you intend the value to be used for MEMORY tables created by all clients.) The following example creates two MEMORY tables, with a maximum size of 1MB and 2MB, respectively:

```
mysql> SET max_heap_table_size = 1024*1024;
Query OK, 0 rows affected (0.00 sec)

mysql> CREATE TABLE t1 (id INT, UNIQUE(id)) ENGINE = MEMORY;
Query OK, 0 rows affected (0.01 sec)

mysql> SET max_heap_table_size = 1024*1024*2;
Query OK, 0 rows affected (0.00 sec)

mysql> CREATE TABLE t2 (id INT, UNIQUE(id)) ENGINE = MEMORY;
Query OK, 0 rows affected (0.00 sec)
```

Both tables revert to the server's global max_heap_table_size value if the server restarts.

You can also specify a MAX_ROWS table option in CREATE TABLE statements for MEMORY tables to provide a hint about the number of rows you plan to store in them. This does not enable the table to grow beyond the max_heap_table_size value, which still acts as a constraint on maximum table size. For maximum flexibility in being able to use MAX_ROWS, set max_heap_table_size at least as high as the value to which you want each MEMORY table to be able to grow.

Additional Resources

A forum dedicated to the MEMORY storage engine is available at http://forums.mysql.com/list.php?92.

14.5 The CSV Storage Engine

The CSV storage engine stores data in text files using comma-separated values format.

The CSV storage engine is always compiled into the MySQL server.

To examine the source for the CSV engine, look in the storage/csv directory of a MySQL source distribution.

When you create a CSV table, the server creates a table format file in the database directory. The file begins with the table name and has an .frm extension. The storage engine also creates a data file. Its name begins with the table name and has a .CSV extension. The data file is a plain text file. When you store data into the table, the storage engine saves it into the data file in comma-separated values format.

```
mysql> CREATE TABLE test (i INT NOT NULL, c CHAR(10) NOT NULL)
    -> ENGINE = CSV;
```

```
Query OK, 0 rows affected (0.12 sec)

mysql> INSERT INTO test VALUES(1,'record one'),(2,'record two');
Query OK, 2 rows affected (0.00 sec)
Records: 2  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM test;
+------+------------+
| i    | c          |
+------+------------+
|    1 | record one |
|    2 | record two |
+------+------------+
2 rows in set (0.00 sec)
```

Creating a CSV table also creates a corresponding Metafile that stores the state of the table and the number of rows that exist in the table. The name of this file is the same as the name of the table with the extension `CSM`.

If you examine the `test.CSV` file in the database directory created by executing the preceding statements, its contents should look like this:

```

"1","record one"
"2","record two"
```

This format can be read, and even written, by spreadsheet applications such as Microsoft Excel or StarOffice Calc.

14.5.1 Repairing and Checking CSV Tables

The CSV storage engines supports the `CHECK` and `REPAIR` statements to verify and if possible repair a damaged CSV table.

When running the `CHECK` statement, the CSV file will be checked for validity by looking for the correct field separators, escaped fields (matching or missing quotation marks), the correct number of fields compared to the table definition and the existence of a corresponding CSV metafile. The first invalid row discovered will report an error. Checking a valid table produces output like that shown below:

```
mysql> check table csvtest;
+--------------+-------+----------+----------+
| Table        | Op    | Msg_type | Msg_text |
+--------------+-------+----------+----------+
| test.csvtest | check | status   | OK       |
+--------------+-------+----------+----------+
1 row in set (0.00 sec)
```

A check on a corrupted table returns a fault:

```
mysql> check table csvtest;
+--------------+-------+----------+----------+
| Table        | Op    | Msg_type | Msg_text |
+--------------+-------+----------+----------+
| test.csvtest | check | error    | Corrupt  |
+--------------+-------+----------+----------+
1 row in set (0.01 sec)
```

If the check fails, the table is marked as crashed (corrupt). Once a table has been marked as corrupt, it is automatically repaired when you next run `CHECK` or execute a `SELECT` statement. The corresponding corrupt status and new status will be displayed when running `CHECK`:

```
mysql> check table csvtest;
+--------------+-------+----------+----------------------------+
| Table        | Op    | Msg_type | Msg_text                   |
+--------------+-------+----------+----------------------------+
| test.csvtest | check | warning  | Table is marked as crashed |
| test.csvtest | check | status   | OK                         |
+--------------+-------+----------+----------------------------+
2 rows in set (0.08 sec)
```

To repair a table you can use REPAIR, this copies as many valid rows from the existing CSV data as possible, and then replaces the existing CSV file with the recovered rows. Any rows beyond the corrupted data are lost.

```
mysql> repair table csvtest;
+--------------+--------+----------+----------+
| Table        | Op     | Msg_type | Msg_text |
+--------------+--------+----------+----------+
| test.csvtest | repair | status   | OK       |
+--------------+--------+----------+----------+
1 row in set (0.02 sec)
```

Warning

Note that during repair, only the rows from the CSV file up to the first damaged row are copied to the new table. All other rows from the first damaged row to the end of the table are removed, even valid rows.

14.5.2 CSV Limitations

The CSV storage engine does not support indexing.

Partitioning is not supported for tables using the CSV storage engine.

All tables that you create using the CSV storage engine must have the NOT NULL attribute on all columns. However, for backward compatibility, you can continue to use tables with nullable columns that were created in previous MySQL releases. (Bug #32050)

14.6 The ARCHIVE Storage Engine

The ARCHIVE storage engine produces special-purpose tables that store large amounts of unindexed data in a very small footprint.

Table 14.15 ARCHIVE Storage Engine Features

Storage limits	None	**Transactions**	No	**Locking granularity**	Table
MVCC	No	**Geospatial data type support**	Yes	**Geospatial indexing support**	No
B-tree indexes	No	**T-tree indexes**	No	**Hash indexes**	No
Full-text search indexes	No	**Clustered indexes**	No	**Data caches**	No
Index caches	No	**Compressed data**	Yes	**Encrypted data**[a]	Yes
Cluster database support	No	**Replication support**[b]	Yes	**Foreign key support**	No
Backup / point-in-time recovery[c]	Yes	**Query cache support**	Yes	**Update statistics for data dictionary**	Yes

[a] Implemented in the server (via encryption functions), rather than in the storage engine.

[b] Implemented in the server, rather than in the storage engine.

[c] Implemented in the server, rather than in the storage engine.

The ARCHIVE storage engine is included in MySQL binary distributions. To enable this storage engine if you build MySQL from source, invoke CMake with the -DWITH_ARCHIVE_STORAGE_ENGINE option.

To examine the source for the ARCHIVE engine, look in the storage/archive directory of a MySQL source distribution.

You can check whether the ARCHIVE storage engine is available with the SHOW ENGINES statement.

When you create an ARCHIVE table, the server creates a table format file in the database directory. The file begins with the table name and has an .frm extension. The storage engine creates other files, all having names beginning with the table name. The data file has an extension of .ARZ. An .ARN file may appear during optimization operations.

The ARCHIVE engine supports INSERT and SELECT, but not DELETE, REPLACE, or UPDATE. It does support ORDER BY operations, BLOB columns, and basically all but spatial data types (see Section 12.18.4.1, “MySQL Spatial Data Types”). The ARCHIVE engine uses row-level locking.

The ARCHIVE engine supports the AUTO_INCREMENT column attribute. The AUTO_INCREMENT column can have either a unique or nonunique index. Attempting to create an index on any other column results in an error. The ARCHIVE engine also supports the AUTO_INCREMENT table option in CREATE TABLE statements to specify the initial sequence value for a new table or reset the sequence value for an existing table, respectively.

ARCHIVE does not support inserting a value into an AUTO_INCREMENT column less than the current maximum column value. Attempts to do so result in an ER_DUP_KEY error.

The ARCHIVE engine ignores BLOB columns if they are not requested and scans past them while reading.

Storage: Rows are compressed as they are inserted. The ARCHIVE engine uses zlib lossless data compression (see http://www.zlib.net/). You can use OPTIMIZE TABLE to analyze the table and pack it into a smaller format (for a reason to use OPTIMIZE TABLE, see later in this section). The engine also supports CHECK TABLE. There are several types of insertions that are used:

- An INSERT statement just pushes rows into a compression buffer, and that buffer flushes as necessary. The insertion into the buffer is protected by a lock. A SELECT forces a flush to occur.

- A bulk insert is visible only after it completes, unless other inserts occur at the same time, in which case it can be seen partially. A SELECT never causes a flush of a bulk insert unless a normal insert occurs while it is loading.

Retrieval: On retrieval, rows are uncompressed on demand; there is no row cache. A SELECT operation performs a complete table scan: When a SELECT occurs, it finds out how many rows are currently available and reads that number of rows. SELECT is performed as a consistent read. Note that lots of SELECT statements during insertion can deteriorate the compression, unless only bulk or delayed inserts are used. To achieve better compression, you can use OPTIMIZE TABLE or REPAIR TABLE. The number of rows in ARCHIVE tables reported by SHOW TABLE STATUS is always accurate. See Section 13.7.2.4, “OPTIMIZE TABLE Syntax”, Section 13.7.2.5, “REPAIR TABLE Syntax”, and Section 13.7.5.35, “SHOW TABLE STATUS Syntax”.

Additional Resources

- A forum dedicated to the ARCHIVE storage engine is available at http://forums.mysql.com/list.php?112.

14.7 The BLACKHOLE Storage Engine

The BLACKHOLE storage engine acts as a “black hole” that accepts data but throws it away and does not store it. Retrievals always return an empty result:

```
mysql> CREATE TABLE test(i INT, c CHAR(10)) ENGINE = BLACKHOLE;
Query OK, 0 rows affected (0.03 sec)

mysql> INSERT INTO test VALUES(1,'record one'),(2,'record two');
Query OK, 2 rows affected (0.00 sec)
Records: 2  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM test;
Empty set (0.00 sec)
```

To enable the BLACKHOLE storage engine if you build MySQL from source, invoke CMake with the -DWITH_BLACKHOLE_STORAGE_ENGINE option.

To examine the source for the BLACKHOLE engine, look in the sql directory of a MySQL source distribution.

When you create a BLACKHOLE table, the server creates a table format file in the database directory. The file begins with the table name and has an .frm extension. There are no other files associated with the table.

The BLACKHOLE storage engine supports all kinds of indexes. That is, you can include index declarations in the table definition.

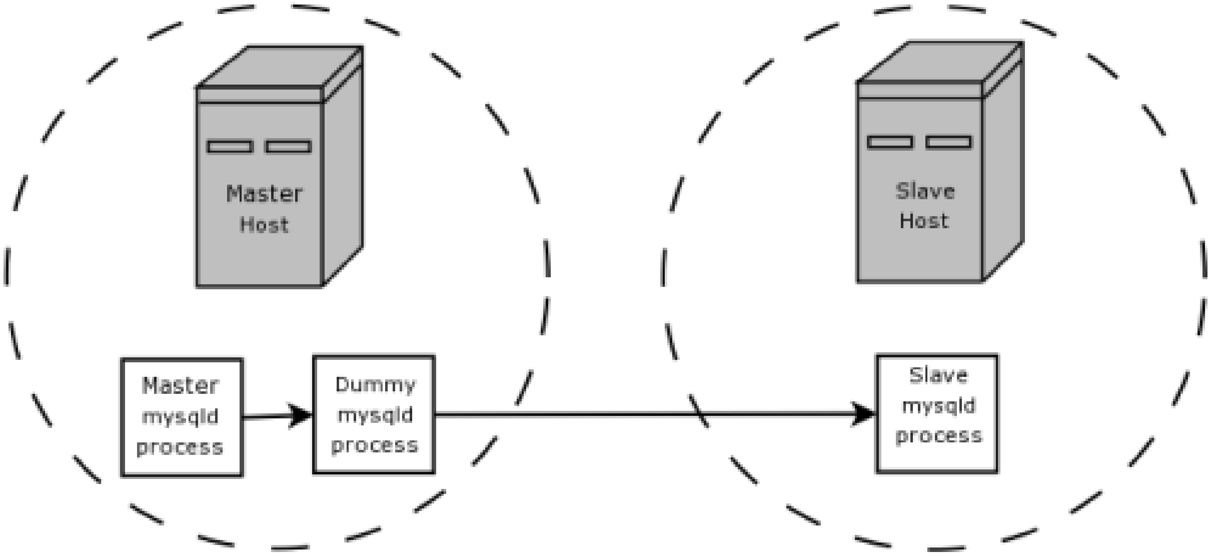
You can check whether the BLACKHOLE storage engine is available with the SHOW ENGINES statement.

Inserts into a BLACKHOLE table do not store any data, but if statement based binary logging is enabled, the SQL statements are logged and replicated to slave servers. This can be useful as a repeater or filter mechanism.

Note

When using the row based format for the binary log, updates and deletes are skipped, and neither logged nor applied. For this reason, you should use STATEMENT for the binary logging format, and not ROW or MIXED.

Suppose that your application requires slave-side filtering rules, but transferring all binary log data to the slave first results in too much traffic. In such a case, it is possible to set up on the master host a “dummy” slave process whose default storage engine is BLACKHOLE, depicted as follows:



The master writes to its binary log. The “dummy” mysqld process acts as a slave, applying the desired combination of replicate-do-* and replicate-ignore-* rules, and writes a new, filtered binary log of its own. (See Section 16.1.4, “Replication and Binary Logging Options and Variables”.) This filtered log is provided to the slave.

The dummy process does not actually store any data, so there is little processing overhead incurred by running the additional mysqld process on the replication master host. This type of setup can be repeated with additional replication slaves.

INSERT triggers for BLACKHOLE tables work as expected. However, because the BLACKHOLE table does not actually store any data, UPDATE and DELETE triggers are not activated: The FOR EACH ROW clause in the trigger definition does not apply because there are no rows.

Other possible uses for the BLACKHOLE storage engine include:

- Verification of dump file syntax.
- Measurement of the overhead from binary logging, by comparing performance using BLACKHOLE with and without binary logging enabled.
- BLACKHOLE is essentially a “no-op” storage engine, so it could be used for finding performance bottlenecks not related to the storage engine itself.

The BLACKHOLE engine is transaction-aware, in the sense that committed transactions are written to the binary log and rolled-back transactions are not.

Blackhole Engine and Auto Increment Columns

The Blackhole engine is a no-op engine. Any operations performed on a table using Blackhole will have no effect. This should be born in mind when considering the behavior of primary key columns that auto increment. The engine will not automatically increment field values, and does not retain auto increment field state. This has important implications in replication.

Consider the following replication scenario where all three of the following conditions apply:

1. On a master server there is a blackhole table with an auto increment field that is a primary key.
2. On a slave the same table exists but using the MyISAM engine.
3. Inserts are performed into the master's table without explicitly setting the auto increment value in the INSERT statement itself or through using a SET INSERT_ID statement.

In this scenario replication will fail with a duplicate entry error on the primary key column.

In statement based replication, the value of INSERT_ID in the context event will always be the same. Replication will therefore fail due to trying insert a row with a duplicate value for a primary key column.

In row based replication, the value that the engine returns for the row always be the same for each insert. This will result in the slave attempting to replay two insert log entries using the same value for the primary key column, and so replication will fail.

Column Filtering

When using row-based replication, (binlog_format=ROW), a slave where the last columns are missing from a table is supported, as described in the section Section 16.4.1.9, “Replication with Differing Table Definitions on Master and Slave”.

This filtering works on the slave side, that is, the columns are copied to the slave before they are filtered out. There are at least two cases where it is not desirable to copy the columns to the slave:

1. If the data is confidential, so the slave server should not have access to it.

2. If the master has many slaves, filtering before sending to the slaves may reduce network traffic.

Master column filtering can be achieved using the BLACKHOLE engine. This is carried out in a way similar to how master table filtering is achieved - by using the BLACKHOLE engine and the --replicate-do-table or --replicate-ignore-table option.

The setup for the master is:

```
CREATE TABLE t1 (public_col_1, ..., public_col_N,
                 secret_col_1, ..., secret_col_M) ENGINE=MyISAM;
```

The setup for the trusted slave is:

```
CREATE TABLE t1 (public_col_1, ..., public_col_N) ENGINE=BLACKHOLE;
```

The setup for the untrusted slave is:

```
CREATE TABLE t1 (public_col_1, ..., public_col_N) ENGINE=MyISAM;
```

14.8 The MERGE Storage Engine

The MERGE storage engine, also known as the MRG_MyISAM engine, is a collection of identical MyISAM tables that can be used as one. “Identical” means that all tables have identical column and index information. You cannot merge MyISAM tables in which the columns are listed in a different order, do not have exactly the same columns, or have the indexes in different order. However, any or all of the MyISAM tables can be compressed with myisampack. See Section 4.6.5, “myisampack — Generate Compressed, Read-Only MyISAM Tables”. Differences in table options such as AVG_ROW_LENGTH, MAX_ROWS, or PACK_KEYS do not matter.

An alternative to a MERGE table is a partitioned table, which stores partitions of a single table in separate files. Partitioning enables some operations to be performed more efficiently and is not limited to the MyISAM storage engine. For more information, see Chapter 17, *Partitioning*.

When you create a MERGE table, MySQL creates two files on disk. The files have names that begin with the table name and have an extension to indicate the file type. An .frm file stores the table format, and an .MRG file contains the names of the underlying MyISAM tables that should be used as one. The tables do not have to be in the same database as the MERGE table.

You can use SELECT, DELETE, UPDATE, and INSERT on MERGE tables. You must have SELECT, DELETE, and UPDATE privileges on the MyISAM tables that you map to a MERGE table.

Note

The use of MERGE tables entails the following security issue: If a user has access to MyISAM table *t*, that user can create a MERGE table *m* that accesses *t*. However, if the user's privileges on *t* are subsequently revoked, the user can continue to access *t* by doing so through *m*.

Use of DROP TABLE with a MERGE table drops only the MERGE specification. The underlying tables are not affected.

To create a MERGE table, you must specify a UNION=(*list-of-tables*) option that indicates which MyISAM tables to use. You can optionally specify an INSERT_METHOD option to control how inserts into the MERGE table take place. Use a value of FIRST or LAST to cause inserts to be made in the first or last underlying table, respectively. If you specify no INSERT_METHOD option or if you specify it with a value of NO, inserts into the MERGE table are not permitted and attempts to do so result in an error.

The following example shows how to create a MERGE table:

```
mysql> CREATE TABLE t1 (
    ->    a INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    ->    message CHAR(20)) ENGINE=MyISAM;
mysql> CREATE TABLE t2 (
    ->    a INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    ->    message CHAR(20)) ENGINE=MyISAM;
mysql> INSERT INTO t1 (message) VALUES ('Testing'),('table'),('t1');
mysql> INSERT INTO t2 (message) VALUES ('Testing'),('table'),('t2');
mysql> CREATE TABLE total (
    ->    a INT NOT NULL AUTO_INCREMENT,
    ->    message CHAR(20), INDEX(a))
    ->    ENGINE=MERGE UNION=(t1,t2) INSERT_METHOD=LAST;
```

Note that column a is indexed as a PRIMARY KEY in the underlying MyISAM tables, but not in the MERGE table. There it is indexed but not as a PRIMARY KEY because a MERGE table cannot enforce uniqueness over the set of underlying tables. (Similarly, a column with a UNIQUE index in the underlying tables should be indexed in the MERGE table but not as a UNIQUE index.)

After creating the MERGE table, you can use it to issue queries that operate on the group of tables as a whole:

```
mysql> SELECT * FROM total;
+---+---------+
| a | message |
+---+---------+
| 1 | Testing |
| 2 | table   |
| 3 | t1      |
| 1 | Testing |
| 2 | table   |
| 3 | t2      |
+---+---------+
```

To remap a MERGE table to a different collection of MyISAM tables, you can use one of the following methods:

- DROP the MERGE table and re-create it.
- Use ALTER TABLE *tbl_name* UNION=(...) to change the list of underlying tables.

  It is also possible to use ALTER TABLE ... UNION=() (that is, with an empty UNION clause) to remove all of the underlying tables. However, in this case, the table is effectively empty and inserts fail because there is no underlying table to take new rows. Such a table might be useful as a template for creating new MERGE tables with CREATE TABLE ... LIKE.

The underlying table definitions and indexes must conform closely to the definition of the MERGE table. Conformance is checked when a table that is part of a MERGE table is opened, not when the MERGE table is created. If any table fails the conformance checks, the operation that triggered the opening of the table fails. This means that changes to the definitions of tables within a MERGE may cause a failure when the MERGE table is accessed. The conformance checks applied to each table are:

- The underlying table and the MERGE table must have the same number of columns.

- The column order in the underlying table and the MERGE table must match.
- Additionally, the specification for each corresponding column in the parent MERGE table and the underlying tables are compared and must satisfy these checks:
    - The column type in the underlying table and the MERGE table must be equal.
    - The column length in the underlying table and the MERGE table must be equal.
    - The column of the underlying table and the MERGE table can be NULL.
- The underlying table must have at least as many indexes as the MERGE table. The underlying table may have more indexes than the MERGE table, but cannot have fewer.

    Note

    A known issue exists where indexes on the same columns must be in identical order, in both the MERGE table and the underlying MyISAM table. See Bug #33653.

    Each index must satisfy these checks:

    - The index type of the underlying table and the MERGE table must be the same.
    - The number of index parts (that is, multiple columns within a compound index) in the index definition for the underlying table and the MERGE table must be the same.
    - For each index part:
        - Index part lengths must be equal.
        - Index part types must be equal.
        - Index part languages must be equal.
        - Check whether index parts can be NULL.

If a MERGE table cannot be opened or used because of a problem with an underlying table, CHECK TABLE displays information about which table caused the problem.

Additional Resources

- A forum dedicated to the MERGE storage engine is available at http://forums.mysql.com/list.php?93.

14.8.1 MERGE Table Advantages and Disadvantages

MERGE tables can help you solve the following problems:

- Easily manage a set of log tables. For example, you can put data from different months into separate tables, compress some of them with myisampack, and then create a MERGE table to use them as one.
- Obtain more speed. You can split a large read-only table based on some criteria, and then put individual tables on different disks. A MERGE table structured this way could be much faster than using a single large table.
- Perform more efficient searches. If you know exactly what you are looking for, you can search in just one of the underlying tables for some queries and use a MERGE table for others. You can even have many different MERGE tables that use overlapping sets of tables.

- Perform more efficient repairs. It is easier to repair individual smaller tables that are mapped to a MERGE table than to repair a single large table.
- Instantly map many tables as one. A MERGE table need not maintain an index of its own because it uses the indexes of the individual tables. As a result, MERGE table collections are *very* fast to create or remap. (You must still specify the index definitions when you create a MERGE table, even though no indexes are created.)
- If you have a set of tables from which you create a large table on demand, you can instead create a MERGE table from them on demand. This is much faster and saves a lot of disk space.
- Exceed the file size limit for the operating system. Each MyISAM table is bound by this limit, but a collection of MyISAM tables is not.
- You can create an alias or synonym for a MyISAM table by defining a MERGE table that maps to that single table. There should be no really notable performance impact from doing this (only a couple of indirect calls and memcpy() calls for each read).

The disadvantages of MERGE tables are:

- You can use only identical MyISAM tables for a MERGE table.
- Some MyISAM features are unavailable in MERGE tables. For example, you cannot create FULLTEXT indexes on MERGE tables. (You can create FULLTEXT indexes on the underlying MyISAM tables, but you cannot search the MERGE table with a full-text search.)
- If the MERGE table is nontemporary, all underlying MyISAM tables must be nontemporary. If the MERGE table is temporary, the MyISAM tables can be any mix of temporary and nontemporary.
- MERGE tables use more file descriptors than MyISAM tables. If 10 clients are using a MERGE table that maps to 10 tables, the server uses (10 × 10) + 10 file descriptors. (10 data file descriptors for each of the 10 clients, and 10 index file descriptors shared among the clients.)
- Index reads are slower. When you read an index, the MERGE storage engine needs to issue a read on all underlying tables to check which one most closely matches a given index value. To read the next index value, the MERGE storage engine needs to search the read buffers to find the next value. Only when one index buffer is used up does the storage engine need to read the next index block. This makes MERGE indexes much slower on eq_ref searches, but not much slower on ref searches. For more information about eq_ref and ref, see Section 13.8.2, “EXPLAIN Syntax”.

14.8.2 MERGE Table Problems

The following are known problems with MERGE tables:

- In versions of MySQL Server prior to 5.1.23, it was possible to create temporary merge tables with nontemporary child MyISAM tables.

  From versions 5.1.23, MERGE children were locked through the parent table. If the parent was temporary, it was not locked and so the children were not locked either. Parallel use of the MyISAM tables corrupted them.
- If you use ALTER TABLE to change a MERGE table to another storage engine, the mapping to the underlying tables is lost. Instead, the rows from the underlying MyISAM tables are copied into the altered table, which then uses the specified storage engine.
- The INSERT_METHOD table option for a MERGE table indicates which underlying MyISAM table to use for inserts into the MERGE table. However, use of the AUTO_INCREMENT table option for that MyISAM table

has no effect for inserts into the MERGE table until at least one row has been inserted directly into the MyISAM table.

- A MERGE table cannot maintain uniqueness constraints over the entire table. When you perform an INSERT, the data goes into the first or last MyISAM table (as determined by the INSERT_METHOD option). MySQL ensures that unique key values remain unique within that MyISAM table, but not over all the underlying tables in the collection.
- Because the MERGE engine cannot enforce uniqueness over the set of underlying tables, REPLACE does not work as expected. The two key facts are:

  - REPLACE can detect unique key violations only in the underlying table to which it is going to write (which is determined by the INSERT_METHOD option). This differs from violations in the MERGE table itself.
  - If REPLACE detects a unique key violation, it will change only the corresponding row in the underlying table it is writing to; that is, the first or last table, as determined by the INSERT_METHOD option.

  Similar considerations apply for INSERT ... ON DUPLICATE KEY UPDATE.
- MERGE tables do not support partitioning. That is, you cannot partition a MERGE table, nor can any of a MERGE table's underlying MyISAM tables be partitioned.
- You should not use ANALYZE TABLE, REPAIR TABLE, OPTIMIZE TABLE, ALTER TABLE, DROP TABLE, DELETE without a WHERE clause, or TRUNCATE TABLE on any of the tables that are mapped into an open MERGE table. If you do so, the MERGE table may still refer to the original table and yield unexpected results. To work around this problem, ensure that no MERGE tables remain open by issuing a FLUSH TABLES statement prior to performing any of the named operations.

  The unexpected results include the possibility that the operation on the MERGE table will report table corruption. If this occurs after one of the named operations on the underlying MyISAM tables, the corruption message is spurious. To deal with this, issue a FLUSH TABLES statement after modifying the MyISAM tables.
- DROP TABLE on a table that is in use by a MERGE table does not work on Windows because the MERGE storage engine's table mapping is hidden from the upper layer of MySQL. Windows does not permit open files to be deleted, so you first must flush all MERGE tables (with FLUSH TABLES) or drop the MERGE table before dropping the table.
- The definition of the MyISAM tables and the MERGE table are checked when the tables are accessed (for example, as part of a SELECT or INSERT statement). The checks ensure that the definitions of the tables and the parent MERGE table definition match by comparing column order, types, sizes and associated indexes. If there is a difference between the tables, an error is returned and the statement fails. Because these checks take place when the tables are opened, any changes to the definition of a single table, including column changes, column ordering, and engine alterations will cause the statement to fail.
- The order of indexes in the MERGE table and its underlying tables should be the same. If you use ALTER TABLE to add a UNIQUE index to a table used in a MERGE table, and then use ALTER TABLE to add a nonunique index on the MERGE table, the index ordering is different for the tables if there was already a nonunique index in the underlying table. (This happens because ALTER TABLE puts UNIQUE indexes before nonunique indexes to facilitate rapid detection of duplicate keys.) Consequently, queries on tables with such indexes may return unexpected results.
- If you encounter an error message similar to ERROR 1017 (HY000): Can't find file: '*tbl_name*.MRG' (errno: 2), it generally indicates that some of the underlying tables do not use the MyISAM storage engine. Confirm that all of these tables are MyISAM.

- The maximum number of rows in a MERGE table is 2^{64} (~1.844E+19; the same as for a MyISAM table). It is not possible to merge multiple MyISAM tables into a single MERGE table that would have more than this number of rows.

- Use of underlying MyISAM tables of differing row formats with a parent MERGE table is currently known to fail. See Bug #32364.

- You cannot change the union list of a nontemporary MERGE table when LOCK TABLES is in effect. The following does *not* work:

  ```
  CREATE TABLE m1 ... ENGINE=MRG_MYISAM ...;
  LOCK TABLES t1 WRITE, t2 WRITE, m1 WRITE;
  ALTER TABLE m1 ... UNION=(t1,t2) ...;
  ```

  However, you can do this with a temporary MERGE table.

- You cannot create a MERGE table with CREATE ... SELECT, neither as a temporary MERGE table, nor as a nontemporary MERGE table. For example:

  ```
  CREATE TABLE m1 ... ENGINE=MRG_MYISAM ... SELECT ...;
  ```

  Attempts to do this result in an error: *tbl_name* is not BASE TABLE.

- In some cases, differing PACK_KEYS table option values among the MERGE and underlying tables cause unexpected results if the underlying tables contain CHAR or BINARY columns. As a workaround, use ALTER TABLE to ensure that all involved tables have the same PACK_KEYS value. (Bug #50646)

14.9 The FEDERATED Storage Engine

The FEDERATED storage engine lets you access data from a remote MySQL database without using replication or cluster technology. Querying a local FEDERATED table automatically pulls the data from the remote (federated) tables. No data is stored on the local tables.

To include the FEDERATED storage engine if you build MySQL from source, invoke CMake with the -DWITH_FEDERATED_STORAGE_ENGINE option.

The FEDERATED storage engine is not enabled by default in the running server; to enable FEDERATED, you must start the MySQL server binary using the --federated option.

To examine the source for the FEDERATED engine, look in the storage/federated directory of a MySQL source distribution.

14.9.1 FEDERATED Storage Engine Overview

When you create a table using one of the standard storage engines (such as MyISAM, CSV or InnoDB), the table consists of the table definition and the associated data. When you create a FEDERATED table, the table definition is the same, but the physical storage of the data is handled on a remote server.

A FEDERATED table consists of two elements:

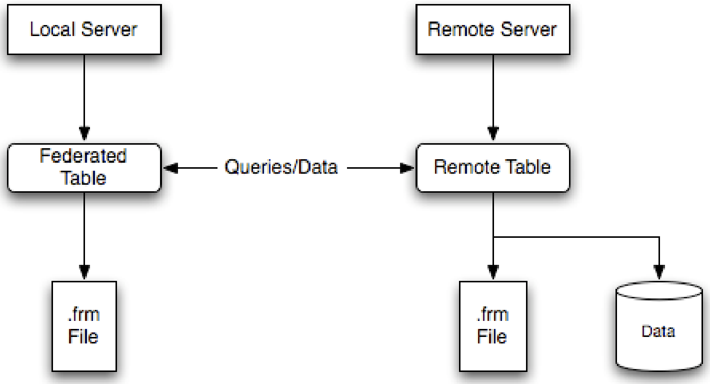
- A *remote server* with a database table, which in turn consists of the table definition (stored in the .frm file) and the associated table. The table type of the remote table may be any type supported by the remote mysqld server, including MyISAM or InnoDB.

- A *local server* with a database table, where the table definition matches that of the corresponding table on the remote server. The table definition is stored within the .frm file. However, there is no data file on the local server. Instead, the table definition includes a connection string that points to the remote table.

When executing queries and statements on a FEDERATED table on the local server, the operations that would normally insert, update or delete information from a local data file are instead sent to the remote server for execution, where they update the data file on the remote server or return matching rows from the remote server.

The basic structure of a FEDERATED table setup is shown in Figure 14.1, “FEDERATED Table Structure”.

Figure 14.1 FEDERATED Table Structure



When a client issues an SQL statement that refers to a FEDERATED table, the flow of information between the local server (where the SQL statement is executed) and the remote server (where the data is physically stored) is as follows:

1. The storage engine looks through each column that the FEDERATED table has and constructs an appropriate SQL statement that refers to the remote table.

2. The statement is sent to the remote server using the MySQL client API.

3. The remote server processes the statement and the local server retrieves any result that the statement produces (an affected-rows count or a result set).

4. If the statement produces a result set, each column is converted to internal storage engine format that the FEDERATED engine expects and can use to display the result to the client that issued the original statement.

The local server communicates with the remote server using MySQL client C API functions. It invokes mysql_real_query() to send the statement. To read a result set, it uses mysql_store_result() and fetches rows one at a time using mysql_fetch_row().

14.9.2 How to Create FEDERATED Tables

To create a FEDERATED table you should follow these steps:

1. Create the table on the remote server. Alternatively, make a note of the table definition of an existing table, perhaps using the SHOW CREATE TABLE statement.

2. Create the table on the local server with an identical table definition, but adding the connection information that links the local table to the remote table.

For example, you could create the following table on the remote server:

```
CREATE TABLE test_table (
    id     INT(20) NOT NULL AUTO_INCREMENT,
    name   VARCHAR(32) NOT NULL DEFAULT '',
    other  INT(20) NOT NULL DEFAULT '0',
    PRIMARY KEY  (id),
    INDEX name (name),
    INDEX other_key (other)
)
ENGINE=MyISAM
DEFAULT CHARSET=latin1;
```

To create the local table that will be federated to the remote table, there are two options available. You can either create the local table and specify the connection string (containing the server name, login, password) to be used to connect to the remote table using the CONNECTION, or you can use an existing connection that you have previously created using the CREATE SERVER statement.

Important

When you create the local table it *must* have an identical field definition to the remote table.

Note

You can improve the performance of a FEDERATED table by adding indexes to the table on the host. The optimization will occur because the query sent to the remote server will include the contents of the WHERE clause and will be sent to the remote server and subsequently executed locally. This reduces the network traffic that would otherwise request the entire table from the server for local processing.

14.9.2.1 Creating a FEDERATED Table Using CONNECTION

To use the first method, you must specify the CONNECTION string after the engine type in a CREATE TABLE statement. For example:

```
CREATE TABLE federated_table (
    id     INT(20) NOT NULL AUTO_INCREMENT,
    name   VARCHAR(32) NOT NULL DEFAULT '',
    other  INT(20) NOT NULL DEFAULT '0',
    PRIMARY KEY  (id),
    INDEX name (name),
    INDEX other_key (other)
)
ENGINE=FEDERATED
DEFAULT CHARSET=latin1
CONNECTION='mysql://fed_user@remote_host:9306/federated/test_table';
```

Note

CONNECTION replaces the COMMENT used in some previous versions of MySQL.

The CONNECTION string contains the information required to connect to the remote server containing the table that will be used to physically store the data. The connection string specifies the server name, login credentials, port number and database/table information. In the example, the remote table is on the server remote_host, using port 9306. The name and port number should match the host name (or IP address) and port number of the remote MySQL server instance you want to use as your remote table.

The format of the connection string is as follows:

```
scheme://user_name[:password]@host_name[:port_num]/db_name/tbl_name
```

Where:

- *scheme*: A recognized connection protocol. Only `mysql` is supported as the *scheme* value at this point.
- *user_name*: The user name for the connection. This user must have been created on the remote server, and must have suitable privileges to perform the required actions (`SELECT`, `INSERT`, `UPDATE`, and so forth) on the remote table.
- *password*: (Optional) The corresponding password for *user_name*.
- *host_name*: The host name or IP address of the remote server.
- *port_num*: (Optional) The port number for the remote server. The default is 3306.
- *db_name*: The name of the database holding the remote table.
- *tbl_name*: The name of the remote table. The name of the local and the remote table do not have to match.

Sample connection strings:

```
CONNECTION='mysql://username:password@hostname:port/database/tablename'
CONNECTION='mysql://username@hostname/database/tablename'
CONNECTION='mysql://username:password@hostname/database/tablename'
```

14.9.2.2 Creating a FEDERATED Table Using CREATE SERVER

If you are creating a number of `FEDERATED` tables on the same server, or if you want to simplify the process of creating `FEDERATED` tables, you can use the `CREATE SERVER` statement to define the server connection parameters, just as you would with the `CONNECTION` string.

The format of the `CREATE SERVER` statement is:

```
CREATE SERVER
server_name
FOREIGN DATA WRAPPER wrapper_name
OPTIONS (option [, option] ...)
```

The *server_name* is used in the connection string when creating a new `FEDERATED` table.

For example, to create a server connection identical to the `CONNECTION` string:

```
CONNECTION='mysql://fed_user@remote_host:9306/federated/test_table';
```

You would use the following statement:

```
CREATE SERVER fedlink
FOREIGN DATA WRAPPER mysql
OPTIONS (USER 'fed_user', HOST 'remote_host', PORT 9306, DATABASE 'federated');
```

To create a `FEDERATED` table that uses this connection, you still use the `CONNECTION` keyword, but specify the name you used in the `CREATE SERVER` statement.

```
CREATE TABLE test_table (
    id     INT(20) NOT NULL AUTO_INCREMENT,
    name   VARCHAR(32) NOT NULL DEFAULT '',
    other  INT(20) NOT NULL DEFAULT '0',
    PRIMARY KEY  (id),
    INDEX name (name),
    INDEX other_key (other)
)
ENGINE=FEDERATED
DEFAULT CHARSET=latin1
CONNECTION='fedlink/test_table';
```

The connection name in this example contains the name of the connection (`fedlink`) and the name of the table (`test_table`) to link to, separated by a slash. If you specify only the connection name without a table name, the table name of the local table is used instead.

For more information on `CREATE SERVER`, see Section 13.1.13, “`CREATE SERVER` Syntax”.

The `CREATE SERVER` statement accepts the same arguments as the `CONNECTION` string. The `CREATE SERVER` statement updates the rows in the `mysql.servers` table. See the following table for information on the correspondence between parameters in a connection string, options in the `CREATE SERVER` statement, and the columns in the `mysql.servers` table. For reference, the format of the `CONNECTION` string is as follows:

```
scheme://user_name[:password]@host_name[:port_num]/db_name/tbl_name
```

Description	`CONNECTION` string	`CREATE SERVER` option	`mysql.servers` column
Connection scheme	*`scheme`*	`wrapper_name`	`Wrapper`
Remote user	*`user_name`*	`USER`	`Username`
Remote password	*`password`*	`PASSWORD`	`Password`
Remote host	*`host_name`*	`HOST`	`Host`
Remote port	*`port_num`*	`PORT`	`Port`
Remote database	*`db_name`*	`DATABASE`	`Db`

14.9.3 `FEDERATED` Storage Engine Notes and Tips

You should be aware of the following points when using the `FEDERATED` storage engine:

- `FEDERATED` tables may be replicated to other slaves, but you must ensure that the slave servers are able to use the user/password combination that is defined in the `CONNECTION` string (or the row in the `mysql.servers` table) to connect to the remote server.

The following items indicate features that the `FEDERATED` storage engine does and does not support:

- The remote server must be a MySQL server.
- The remote table that a `FEDERATED` table points to *must* exist before you try to access the table through the `FEDERATED` table.
- It is possible for one `FEDERATED` table to point to another, but you must be careful not to create a loop.
- A `FEDERATED` table does not support indexes per se. Because access to the table is handled remotely, it is the remote table that supports the indexes. Care should be taken when creating a `FEDERATED` table since the index definition from an equivalent `MyISAM` or other table may not be supported. For example, creating a `FEDERATED` table with an index prefix on `VARCHAR`, `TEXT` or `BLOB` columns will fail. The following definition in `MyISAM` is valid:

```
CREATE TABLE `T1`(`A` VARCHAR(100),UNIQUE KEY(`A`(30))) ENGINE=MYISAM;
```

The key prefix in this example is incompatible with the `FEDERATED` engine, and the equivalent statement will fail:

```
CREATE TABLE `T1`(`A` VARCHAR(100),UNIQUE KEY(`A`(30))) ENGINE=FEDERATED
  CONNECTION='MYSQL://127.0.0.1:3306/TEST/T1';
```

If possible, you should try to separate the column and index definition when creating tables on both the remote server and the local server to avoid these index issues.

- Internally, the implementation uses `SELECT`, `INSERT`, `UPDATE`, and `DELETE`, but not `HANDLER`.
- The `FEDERATED` storage engine supports `SELECT`, `INSERT`, `UPDATE`, `DELETE`, `TRUNCATE TABLE`, and indexes. It does not support `ALTER TABLE`, or any Data Definition Language statements that directly affect the structure of the table, other than `DROP TABLE`. The current implementation does not use prepared statements.
- `FEDERATED` accepts `INSERT ... ON DUPLICATE KEY UPDATE` statements, but if a duplicate-key violation occurs, the statement fails with an error.
- Performance on a `FEDERATED` table when performing bulk inserts (for example, on a `INSERT INTO ... SELECT ...` statement) is slower than with other table types because each selected row is treated as an individual `INSERT` statement on the `FEDERATED` table.
- Transactions are not supported.
- `FEDERATED` performs bulk-insert handling such that multiple rows are sent to the remote table in a batch. This provides a performance improvement and enables the remote table to perform improvement. Also, if the remote table is transactional, it enables the remote storage engine to perform statement rollback properly should an error occur. This capability has the following limitations:
  - The size of the insert cannot exceed the maximum packet size between servers. If the insert exceeds this size, it is broken into multiple packets and the rollback problem can occur.
  - Bulk-insert handling does not occur for `INSERT ... ON DUPLICATE KEY UPDATE`.
- There is no way for the `FEDERATED` engine to know if the remote table has changed. The reason for this is that this table must work like a data file that would never be written to by anything other than the database system. The integrity of the data in the local table could be breached if there was any change to the remote database.
- When using a `CONNECTION` string, you cannot use an '@' character in the password. You can get round this limitation by using the `CREATE SERVER` statement to create a server connection.
- The `insert_id` and `timestamp` options are not propagated to the data provider.
- Any `DROP TABLE` statement issued against a `FEDERATED` table drops only the local table, not the remote table.
- `FEDERATED` tables do not work with the query cache.
- User-defined partitioning is not supported for `FEDERATED` tables.

14.9.4 `FEDERATED` Storage Engine Resources

The following additional resources are available for the `FEDERATED` storage engine:

- A forum dedicated to the FEDERATED storage engine is available at http://forums.mysql.com/list.php?105.

14.10 The EXAMPLE Storage Engine

The EXAMPLE storage engine is a stub engine that does nothing. Its purpose is to serve as an example in the MySQL source code that illustrates how to begin writing new storage engines. As such, it is primarily of interest to developers.

To enable the EXAMPLE storage engine if you build MySQL from source, invoke CMake with the -DWITH_EXAMPLE_STORAGE_ENGINE option.

To examine the source for the EXAMPLE engine, look in the storage/example directory of a MySQL source distribution.

When you create an EXAMPLE table, the server creates a table format file in the database directory. The file begins with the table name and has an .frm extension. No other files are created. No data can be stored into the table. Retrievals return an empty result.

```
mysql> CREATE TABLE test (i INT) ENGINE = EXAMPLE;
Query OK, 0 rows affected (0.78 sec)

mysql> INSERT INTO test VALUES(1),(2),(3);
ERROR 1031 (HY000): Table storage engine for 'test' doesn't »
                    have this option

mysql> SELECT * FROM test;
Empty set (0.31 sec)
```

The EXAMPLE storage engine does not support indexing.

14.11 Other Storage Engines

Other storage engines may be available from third parties and community members that have used the Custom Storage Engine interface.

Third party engines are not supported by MySQL. For further information, documentation, installation guides, bug reporting or for any help or assistance with these engines, please contact the developer of the engine directly.

For more information on developing a customer storage engine that can be used with the Pluggable Storage Engine Architecture, see MySQL Internals: Writing a Custom Storage Engine.

14.12 Overview of MySQL Storage Engine Architecture

The MySQL pluggable storage engine architecture enables a database professional to select a specialized storage engine for a particular application need while being completely shielded from the need to manage any specific application coding requirements. The MySQL server architecture isolates the application programmer and DBA from all of the low-level implementation details at the storage level, providing a consistent and easy application model and API. Thus, although there are different capabilities across different storage engines, the application is shielded from these differences.

The pluggable storage engine architecture provides a standard set of management and support services that are common among all underlying storage engines. The storage engines themselves are the components of the database server that actually perform actions on the underlying data that is maintained at the physical server level.

This efficient and modular architecture provides huge benefits for those wishing to specifically target a particular application need—such as data warehousing, transaction processing, or high availability situations—while enjoying the advantage of utilizing a set of interfaces and services that are independent of any one storage engine.

The application programmer and DBA interact with the MySQL database through Connector APIs and service layers that are above the storage engines. If application changes bring about requirements that demand the underlying storage engine change, or that one or more storage engines be added to support new needs, no significant coding or process changes are required to make things work. The MySQL server architecture shields the application from the underlying complexity of the storage engine by presenting a consistent and easy-to-use API that applies across storage engines.

14.12.1 Pluggable Storage Engine Architecture

MySQL Server uses a pluggable storage engine architecture that enables storage engines to be loaded into and unloaded from a running MySQL server.

Plugging in a Storage Engine

Before a storage engine can be used, the storage engine plugin shared library must be loaded into MySQL using the `INSTALL PLUGIN` statement. For example, if the `EXAMPLE` engine plugin is named `example` and the shared library is named `ha_example.so`, you load it with the following statement:

```
mysql> INSTALL PLUGIN example SONAME 'ha_example.so';
```

To install a pluggable storage engine, the plugin file must be located in the MySQL plugin directory, and the user issuing the `INSTALL PLUGIN` statement must have `INSERT` privilege for the `mysql.plugin` table.

The shared library must be located in the MySQL server plugin directory, the location of which is given by the `plugin_dir` system variable.

Unplugging a Storage Engine

To unplug a storage engine, use the `UNINSTALL PLUGIN` statement:

```
mysql> UNINSTALL PLUGIN example;
```

If you unplug a storage engine that is needed by existing tables, those tables become inaccessible, but will still be present on disk (where applicable). Ensure that there are no tables using a storage engine before you unplug the storage engine.

14.12.2 The Common Database Server Layer

A MySQL pluggable storage engine is the component in the MySQL database server that is responsible for performing the actual data I/O operations for a database as well as enabling and enforcing certain feature sets that target a specific application need. A major benefit of using specific storage engines is that you are only delivered the features needed for a particular application, and therefore you have less system overhead in the database, with the end result being more efficient and higher database performance. This is one of the reasons that MySQL has always been known to have such high performance, matching or beating proprietary monolithic databases in industry standard benchmarks.

From a technical perspective, what are some of the unique supporting infrastructure components that are in a storage engine? Some of the key feature differentiations include:

- *Concurrency*: Some applications have more granular lock requirements (such as row-level locks) than others. Choosing the right locking strategy can reduce overhead and therefore improve overall performance. This area also includes support for capabilities such as multi-version concurrency control or “snapshot” read.

- *Transaction Support*: Not every application needs transactions, but for those that do, there are very well defined requirements such as ACID compliance and more.

- *Referential Integrity*: The need to have the server enforce relational database referential integrity through DDL defined foreign keys.

- *Physical Storage*: This involves everything from the overall page size for tables and indexes as well as the format used for storing data to physical disk.

- *Index Support*: Different application scenarios tend to benefit from different index strategies. Each storage engine generally has its own indexing methods, although some (such as B-tree indexes) are common to nearly all engines.

- *Memory Caches*: Different applications respond better to some memory caching strategies than others, so although some memory caches are common to all storage engines (such as those used for user connections or MySQL's high-speed Query Cache), others are uniquely defined only when a particular storage engine is put in play.

- *Performance Aids*: This includes multiple I/O threads for parallel operations, thread concurrency, database checkpointing, bulk insert handling, and more.

- *Miscellaneous Target Features*: This may include support for geospatial operations, security restrictions for certain data manipulation operations, and other similar features.

Each set of the pluggable storage engine infrastructure components are designed to offer a selective set of benefits for a particular application. Conversely, avoiding a set of component features helps reduce unnecessary overhead. It stands to reason that understanding a particular application's set of requirements and selecting the proper MySQL storage engine can have a dramatic impact on overall system efficiency and performance.

Chapter 15 High Availability and Scalability

Table of Contents

Data is the currency of today's web, mobile, social, enterprise and cloud applications. Ensuring data is always available is a top priority for any organization. Minutes of downtime can result in significant loss of revenue and reputation.

There is no “one size fits all” approach to delivering High Availability (HA). Unique application attributes, business requirements, operational capabilities and legacy infrastructure can all influence HA technology selection. And technology is only one element in delivering HA: people and processes are just as critical as the technology itself.

MySQL is deployed into many applications demanding availability and scalability. **Availability** refers to the ability to cope with, and if necessary recover from, failures on the host, including failures of MySQL, the operating system, or the hardware and maintenance activity that may otherwise cause downtime. Scalability refers to the ability to spread both the database and the load of your application queries across multiple MySQL servers.

Because each application has different operational and availability requirements, MySQL offers a range of certified and supported solutions, delivering the appropriate levels of High Availability (HA) and scalability to meet service level requirements. Such solutions extend from replication, through virtualization and geographically redundant, multi-data center solutions delivering 99.999% uptime.

Selecting the right high availability solution for an application largely depends on:

- The level of availability required.

- The type of application being deployed.
- Accepted best practices within your own environment.

The primary solutions supported by MySQL include:

- MySQL Replication. Learn more: Chapter 16, *Replication*
- MySQL Cluster. Learn more: MySQL Cluster NDB 7.3
- Oracle VM Template for MySQL. Learn more: Section 15.1, “Oracle VM Template for MySQL Enterprise Edition”.
- MySQL with DRBD with Corosync and Pacemaker. Learn more: Section 15.2, “Overview of MySQL with DRBD/Pacemaker/Corosync/Oracle Linux”.
- MySQL with Windows Failover Clustering. Learn more: Section 15.3, “Overview of MySQL with Windows Failover Clustering”.
- MySQL with Solaris Cluster. Learn more about Solaris Cluster.

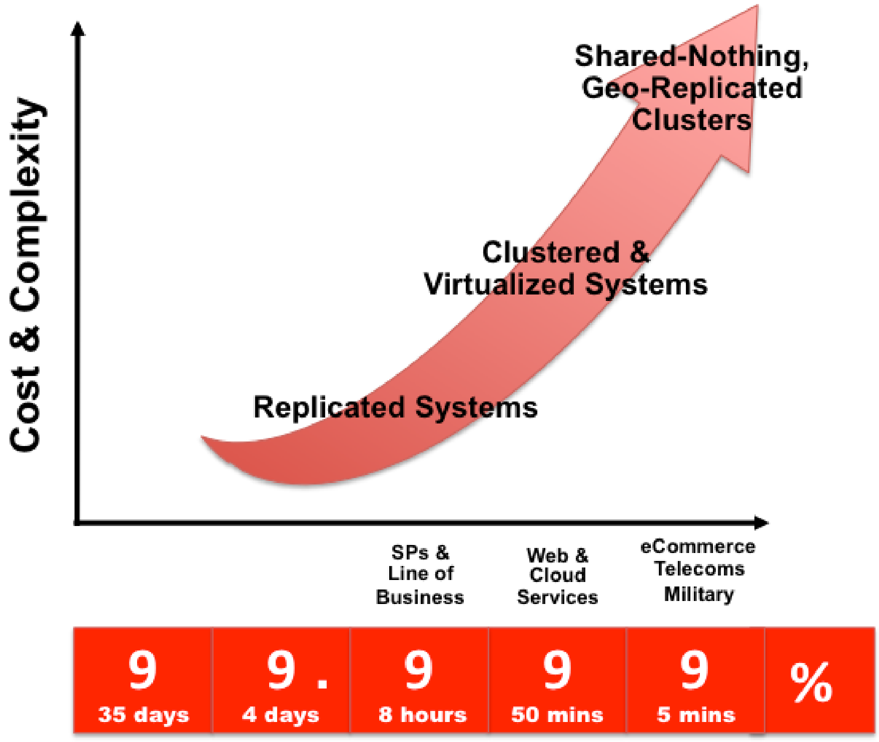
Further options are available using third-party solutions.

Each architecture used to achieve highly available database services is differentiated by the levels of uptime it offers. These architectures can be grouped into three main categories:

- Data Replication.
- Clustered & Virtualized Systems.
- Shared-Nothing, Geographically-Replicated Clusters.

As illustrated in the following figure, each of these architectures offers progressively higher levels of uptime, which must be balanced against potentially greater levels of cost and complexity that each can incur. Simply deploying a high availability architecture is not a guarantee of actually delivering HA. In fact, a poorly implemented and maintained shared-nothing cluster could easily deliver lower levels of availability than a simple data replication solution.

Figure 15.1 Tradeoffs: Cost and Complexity versus Availability



The following table compares the HA and Scalability capabilities of the various MySQL solutions:

Table 15.1 Feature Comparison of MySQL HA Solutions

Requirement	MySQL Replication	DRBD	Oracle VM Template	MySQL Cluster
Availability				
Platform Support	All Supported by MySQL Server (http://www.mysql.com/support/supportedplatforms/database.html)	Linux	Oracle Linux	All Supported by MySQL Cluster (http://www.mysql.com/support/supportedplatforms/cluster.html)
Automated IP Failover	No	Yes	Yes	Depends on Connector and Configuration
Automated Database Failover	No	Yes	Yes	Yes
Automatic Data Resynchronization	No	Yes	N/A - Shared Storage	Yes
Typical Failover Time	User / Script Dependent	Configuration Dependent, 60 seconds and Above	Configuration Dependent, 60 seconds and Above	1 Second and Less

Requirement	**MySQL Replication**	**DRBD**	**Oracle VM Template**	**MySQL Cluster**
Synchronous Replication	No, Asynchronous and Semisynchronous	Yes	N/A - Shared Storage	Yes
Shared Storage	No, Distributed	No, Distributed	Yes	No, Distributed
Geographic redundancy support	Yes	Yes, via MySQL Replication	Yes, via MySQL Replication	Yes, via MySQL Replication
Update Schema On-Line	No	No	No	Yes
Scalability				
Number of Nodes	One Master, Multiple Slaves	One Active (primary), one Passive (secondary) Node	One Active (primary), one Passive (secondary) Node	255
Built-in Load Balancing	Reads, via MySQL Replication	Reads, via MySQL Replication	Reads, via MySQL Replication & During Failover	Yes, Reads and Writes
Supports Read-Intensive Workloads	Yes	Yes	Yes	Yes
Supports Write-Intensive Workloads	Yes, via Application-Level Sharding	Yes, via Application-Level Sharding to Multiple Active/Passive Pairs	Yes, via Application-Level Sharding to Multiple Active/Passive Pairs	Yes, via Auto-Sharding
Scale On-Line (add nodes, repartition, etc.)	No	No	No	Yes

15.1 Oracle VM Template for MySQL Enterprise Edition

Virtualization is a key technology to enable data center efficiency and high availability while providing the foundation for cloud computing. Integrating MySQL Enterprise Edition with Oracle Linux, the Oracle VM Template is the fastest, easiest, and most reliable way to provision virtualized MySQL instances, enabling users to meet the explosive demand for highly available services.

The Oracle VM Template enables rapid deployment and eliminates manual configuration efforts. It provides a preinstalled and pre-configured virtualized MySQL 5.5 Enterprise Edition software image running on Oracle Linux and Oracle VM, certified for production use. The MySQL software image has undergone extensive integration and quality assurance testing as part of the development process.

In addition to rapid provisioning, MySQL users also benefit from the integrated high availability features of Oracle VM which are designed to enable organizations to meet stringent SLA (Service Level Agreement) demands through a combination of:

- **Automatic recovery from failures**, with Oracle VM automatically restarting failed instances on available servers in the server pool after outages of the physical server, VM or MySQL database.
- **Live Migration**, enabling operations staff to move running instances of MySQL to alternative hosts within a server pool during maintenance operations.

Instructions for the creation, deployment and use of the Oracle VM Template for MySQL Enterprise Edition are available from:

- The Oracle VM Template for MySQL Enterprise Edition whitepaper: http://www.mysql.com/why-mysql/white-papers/mysql_wp_oracle-vm-template-for-mee.php.
- The README file accompanying the download of the Template.

To download the Oracle VM Template for MySQL Enterprise, go to http://edelivery.oracle.com/oraclevm and follow these instructions:

- Complete your registration information (Name, Company Name, Email Address and Country) and click on the download agreement.
- Select "Oracle VM Templates" from the "Select a Product Pack" pull-down menu and click "Go".
- Select MySQL Enterprise from the list of Oracle VM Templates.
- Download and unzip the files and refer to the README for further instructions.

15.2 Overview of MySQL with DRBD/Pacemaker/Corosync/Oracle Linux

DRBD (Distributed Replication Block Device) is one of the leading solutions for MySQL HA (High Availability). When combined with Pacemaker and Corosync, users have:

- An end-to-end, integrated stack of mature and proven open source technologies, fully supported by Oracle (as part of MySQL Enterprise Edition).
- Automatic failover and recovery for service continuity.
- Mirroring, via synchronous replication, to ensure failover between nodes without the risk of losing committed transactions.
- Building of HA clusters from commodity hardware, without the requirement for shared-storage.

The following figure illustrates the stack that can be used to deliver a level of High Availability for the MySQL service.

At the lowest level, 2 hosts are required in order to provide physical redundancy; if using a virtual environment, those 2 hosts should be on different physical machines. It is an important feature that no shared storage is required. At any point in time, the services will be active on one host and in standby mode on the other.

Pacemaker and Corosync combine to provide the clustering layer that sits between the services and the underlying hosts and operating systems. Pacemaker is responsible for starting and stopping services, ensuring that they are running on exactly one host, thus delivering high availability and avoiding data corruption. Corosync provides the underlying messaging infrastructure between the nodes that enables Pacemaker to do its job; it also handles the nodes membership within the cluster and informs Pacemaker of any changes.

Figure 15.2 MySQL, DRBD, Pacemaker, and Corosync Stack

The core Pacemaker process does not have built-in knowledge of the specific services to be managed; instead, it uses agents that provide a wrapper for the service-specific actions. For example, in this solution we use agents for Virtual IP Addresses, MySQL and DRBD: these are all existing agents and come packaged with Pacemaker.

The essential services managed by Pacemaker in this configuration are DRBD, MySQL and the Virtual IP Address that applications use to connect to the active MySQL service.

DRBD synchronizes data at the block device (typically a spinning or solid state disk) – transparent to the application, database and even the file system. DRBD requires the use of a journaling file system such as `ext3` or `ext4`. For this solution, it acts in an active-standby mode: at any point in time, the directories being managed by DRBD are accessible for reads and writes on exactly one of the two hosts and inaccessible (even for reads) on the other. Any changes made on the active host are synchronously replicated to the standby host by DRBD.

Download the following guide for detailed instructions on installing, configuring, provisioning and testing the complete MySQL and DRBD stack, including:

- MySQL Database.
- DRBD kernel module and userland utilities.
- Pacemaker and Corosync cluster messaging and management processes.
- Oracle Linux operating system.

Download the guide at: http://www.mysql.com/why-mysql/white-papers/mysql-high-availability-drbd-configuration-deployment-guide/.

Support for DRBD

The complete DRBD stack for MySQL has been certified by Oracle. Commercial support, which provides a single point of contact for the entire stack, whether issues relate to the operating system, DRBD, clustering software or MySQL, is available to those who have both MySQL Enterprise Edition and Oracle Linux Premier Support contracts.

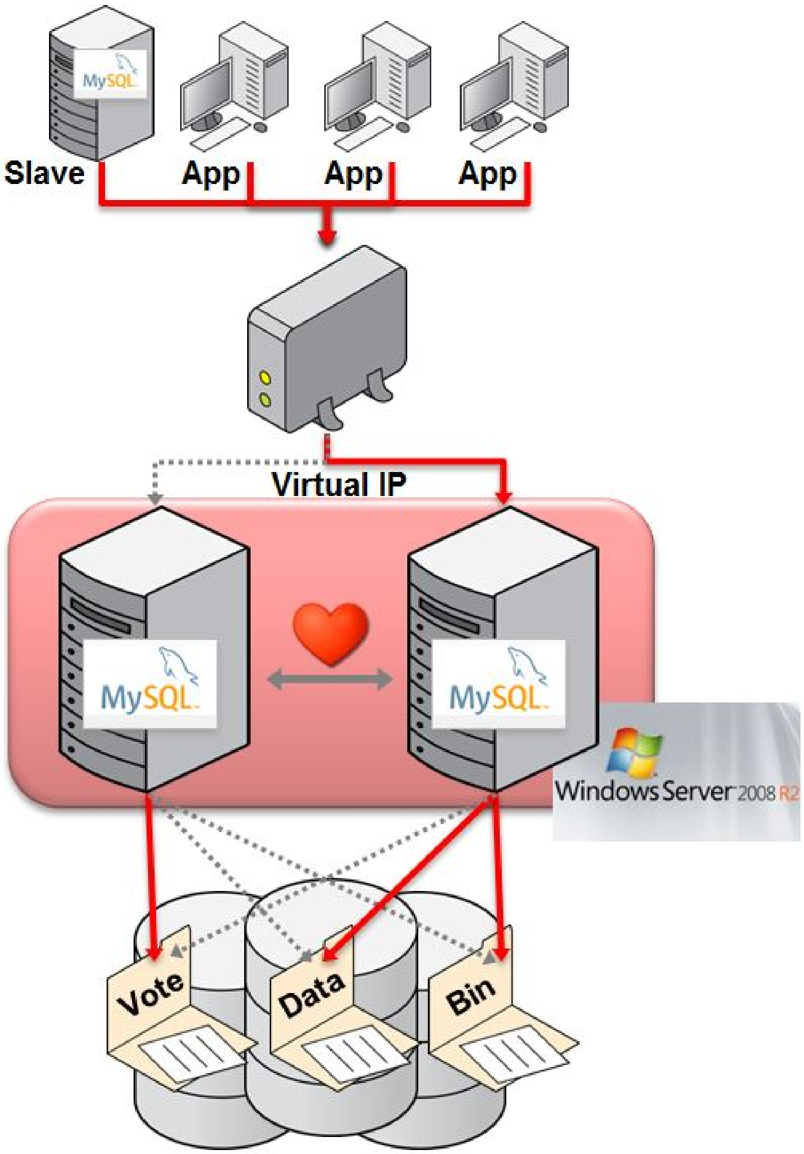
15.3 Overview of MySQL with Windows Failover Clustering

Microsoft Windows is consistently ranked as the top development platform for MySQL, based on surveys of the MySQL user community.

MySQL Enterprise Edition is certified and supported with Windows Server 2008 R2 Failover Clustering (WSFC), enabling organizations to safely deploy business-critical applications demanding high levels of availability using Microsoft's native Windows clustering services.

The following figure illustrates the integration of MySQL with Windows Server Failover Clustering to provide a highly available service:

Figure 15.3 Typical MySQL HA Configuration with Windows Server Failover Clustering



In this architecture, MySQL is deployed in an Active / Passive configuration. Failures of either MySQL or the underlying server are automatically detected and the MySQL instance is restarted on the Passive node. Applications accessing the database, as well as any MySQL replication slaves, can automatically reconnect to the new MySQL process using the same Virtual IP address once MySQL recovery has completed and it starts accepting connections.

MySQL with Windows Failover Clustering requires at least 2 servers within the cluster together with shared storage (for example, FC-AL SAN or iSCSI disks).

The MySQL binaries and data files are stored in the shared storage and Windows Failover Clustering ensures that only one of the cluster nodes will access those files at any point in time.

Clients connect to the MySQL service through a Virtual IP Address (VIP). In the event of failover they experience a brief loss of connection, but otherwise do not need to be aware that the failover has happened, other than to handle the failure of any transactions that were active when the failover occurred.

You can learn more about configuring MySQL with Windows Server Failover Clustering from the whitepaper posted here: http://www.mysql.com/why-mysql/white-papers/mysql_wp_windows_failover_clustering.php

For background and usage information about Windows Server Failover Clustering, see these pages on the Microsoft Technet site:

- Failover Clustering
- Failover Clusters in Windows Server 2008 R2

15.4 Using MySQL within an Amazon EC2 Instance

The Amazon Elastic Compute Cloud (EC2) service provides virtual servers that you can build and deploy to run a variety of different applications and services, including MySQL. The EC2 service is based around the Xen framework, supporting x86, Linux based, platforms with individual instances of a virtual machine referred to as an Amazon Machine Image (AMI). You have complete (root) access to the AMI instance that you create, enabling you to configure and install your AMI in any way you choose.

To use EC2, you create an AMI based on the configuration and applications that you intend to use, and upload the AMI to the Amazon Simple Storage Service (S3). From the S3 resource, you can deploy one or more copies of the AMI to run as an instance within the EC2 environment. The EC2 environment provides management and control of the instance and contextual information about the instance while it is running.

Because you can create and control the AMI, the configuration, and the applications, you can deploy and create any environment you choose. This includes a basic MySQL server in addition to more extensive replication, HA and scalability scenarios that enable you to take advantage of the EC2 environment, and the ability to deploy additional instances as the demand for your MySQL services and applications grow.

To aid the deployment and distribution of work, three different Amazon EC2 instances are available, small (identified as `m1.small`), large (`m1.large`) and extra large (`m1.xlarge`). The different types provide different levels of computing power measured in EC2 computer units (ECU). A summary of the different instance configurations is shown in the following table.

EC2 Attribute	Small	Large	Extra Large
Platform	32-bit	64-bit	64-bit
CPU cores	1	2	4
ECUs	1	4	8
RAM	1.7GB	7.5GB	15GB
Storage	150GB	840GB	1680GB
I/O Performance	Medium	High	High

The typical model for deploying and using MySQL within the EC2 environment is to create a basic AMI that you can use to hold your database data and application. Once the basic environment for your database and application has been created you can then choose to deploy the AMI to a suitable instance. Here the flexibility of having an AMI that can be re-deployed from the small to the large or extra large EC2 instance makes it easy to upgrade the hardware environment without rebuilding your application or database stack.

To get started with MySQL on EC2, including information on how to set up and install MySQL within an EC2 installation and how to port and migrate your data to the running instance, see Section 15.4.1, “Setting Up MySQL on an EC2 AMI”.

For tips and advice on how to create a scalable EC2 environment using MySQL, including guides on setting up replication, see Section 15.4.3, “Deploying a MySQL Database Using EC2”.

15.4.1 Setting Up MySQL on an EC2 AMI

There are many different ways of setting up an EC2 AMI with MySQL, including using any of the pre-configured AMIs supplied by Amazon.

The default *Getting Started* AMI provided by Amazon uses Fedora Core 4, and you can install MySQL by using `yum`:

```
shell> yum install mysql
```

This installs both the MySQL server and the Perl DBD::mysql driver for the Perl DBI API.

Alternatively, you can use one of the AMIs that include MySQL within the standard installation.

Finally, you can also install a standard version of MySQL downloaded from the MySQL Web site. The installation process and instructions are identical to any other installation of MySQL on Linux. See Chapter 2, *Installing and Upgrading MySQL*.

The standard configuration for MySQL places the data files in the default location, `/var/lib/mysql`. The default data directory on an EC2 instance is `/mnt` (although on the large and extra large instance you can alter this configuration). You must edit `/etc/my.cnf` to set the `datadir` option to point to the larger storage area.

Important

The first time you use the main storage location within an EC2 instance it needs to be initialized. The initialization process starts automatically the first time you write to the device. You can start using the device right away, but the write performance of the new device is significantly lower on the initial writes until the initialization process has finished.

To avoid this problem when setting up a new instance, you should start the initialization process before populating your MySQL database. One way to do this is to use `dd` to write to the file system:

```
root-shell> dd if=/dev/zero of=initialize bs=1024M count=50
```

The preceding creates a 50GB on the file system and starts the initialization process. Delete the file once the process has finished.

The initialization process can be time-consuming. On the small instance, initialization takes between two and three hours. For the large and extra large drives, the initialization can be 10 or 20 hours, respectively.

In addition to configuring the correct storage location for your MySQL data files, also consider setting the following other settings in your instance before you save the instance configuration for deployment:

- Set the MySQL server ID, so that when you use it for replication, the ID information is set correctly.

- Enabling binary logging, so that replication can be initialized without starting and stopping the server.

- Set the caching and memory parameters for your storage engines. There are no limitations or restrictions on what storage engines you use in your EC2 environment. Choose a configuration, possibly using one of the standard configurations provided with MySQL appropriate for the instance on which you expect to deploy. The large and extra large instances have RAM that can be dedicated to caching. Be aware that if you choose to install `memcached` on the servers as part of your application stack you must ensure there is enough memory for both MySQL and `memcached`.

Once you have configured your AMI with MySQL and the rest of your application stack, save the AMI so that you can deploy and reuse the instance.

Once you have your application stack configured in an AMI, populating your MySQL database with data should be performed by creating a dump of your database using `mysqldump`, transferring the dump to the EC2 instance, and then reloading the information into the EC2 instance database.

Before using your instance with your application in a production situation, be aware of the limitations of the EC2 instance environment. See Section 15.4.2, “EC2 Instance Limitations”. To begin using your MySQL AMI, consult the notes on deployment. See Section 15.4.3, “Deploying a MySQL Database Using EC2”.

15.4.2 EC2 Instance Limitations

Be aware of the following limitations of the EC2 instances before deploying your applications. Although these shouldn't affect your ability to deploy within the Amazon EC2 environment, they may alter the way you setup and configure your environment to support your application.

- Data stored within instances is not persistent. If you create an instance and populate the instance with data, then the data only remains in place while the machine is running, and does not survive a reboot. If you shut down the instance, any data it contained is lost.

  To ensure that you do not lose information, take regular backups using `mysqldump`. If the data being stored is critical, consider using replication to keep a “live” backup of your data in the event of a failure. When creating a backup, write the data to the Amazon S3 service to avoid the transfer charges applied when copying data offsite.

- EC2 instances are not persistent. If the hardware on which an instance is running fails, the instance is shut down. This can lead to loss of data or service.

  However, if you use EBS, you can attach an EBS storage volume to an EC2 instance, and that EBS volume is persistent. Like a disk, an EBS volume can fail, but it is possible to create point-in-time snapshots of the volume. Snapshots are persisted to Amazon S3 and can be used to restore data in the event of volume failure.

- To replicate your EC2 instances to a non-EC2 environment, be aware of the transfer costs to and from the EC2 service. Data transfer between different EC2 instances is free, so using replication within the EC2 environment does not incur additional charges.

- Certain HA features are either not directly supported, or have limiting factors or problems that could reduce their utility. For example, using DRBD or MySQL Cluster might not work. The default storage configuration is also not redundant. You can use software-based RAID to improve redundancy, but this implies a further performance hit.

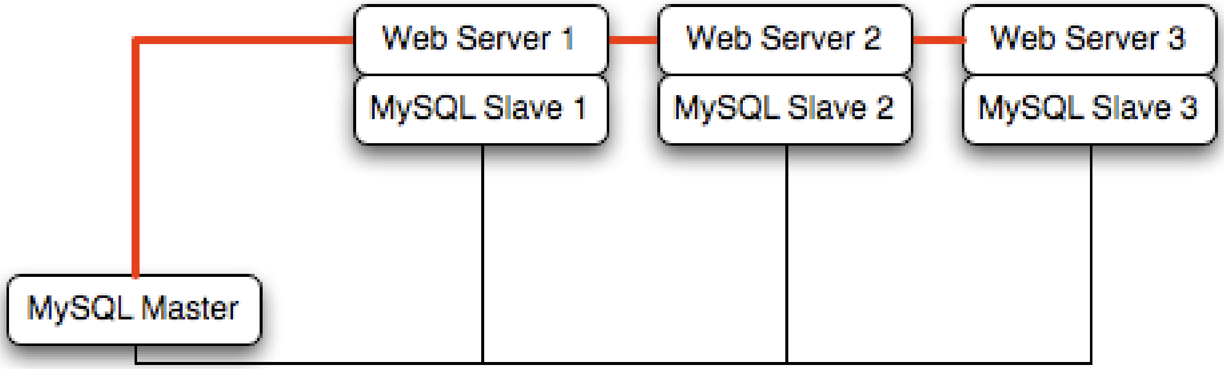
15.4.3 Deploying a MySQL Database Using EC2

Because you cannot guarantee the uptime and availability of your EC2 instances, when deploying MySQL within the EC2 environment, use an approach that enables you to easily distribute work among your EC2 instances. There are a number of ways of doing this. Using sharding techniques, where you split the application across multiple servers dedicating specific blocks of your dataset and users to different servers is an effective way of doing this. As a general rule, it is easier to create more EC2 instances to support more users than to upgrade the instance to a larger machine.

The EC2 architecture works best when you treat the EC2 instances as temporary, cache-based solutions, rather than as a long-term, high availability solution. In addition to using multiple machines, take advantage of other services, such as `memcached` to provide additional caching for your application to help reduce the load on the MySQL server so that it can concentrate on writes. On the large and extra large instances within EC2, the RAM available can provide a large memory cache for data.

Most types of scale-out topology that you would use with your own hardware can be used and applied within the EC2 environment. However, use the limitations and advice already given to ensure that any potential failures do not lose you any data. Also, because the relative power of each EC2 instance is so low, be prepared to alter your application to use sharding and add further EC2 instances to improve the performance of your application.

For example, take the typical scale-out environment shown following, where a single master replicates to one or more slaves (three in this example), with a web server running on each replication slave.

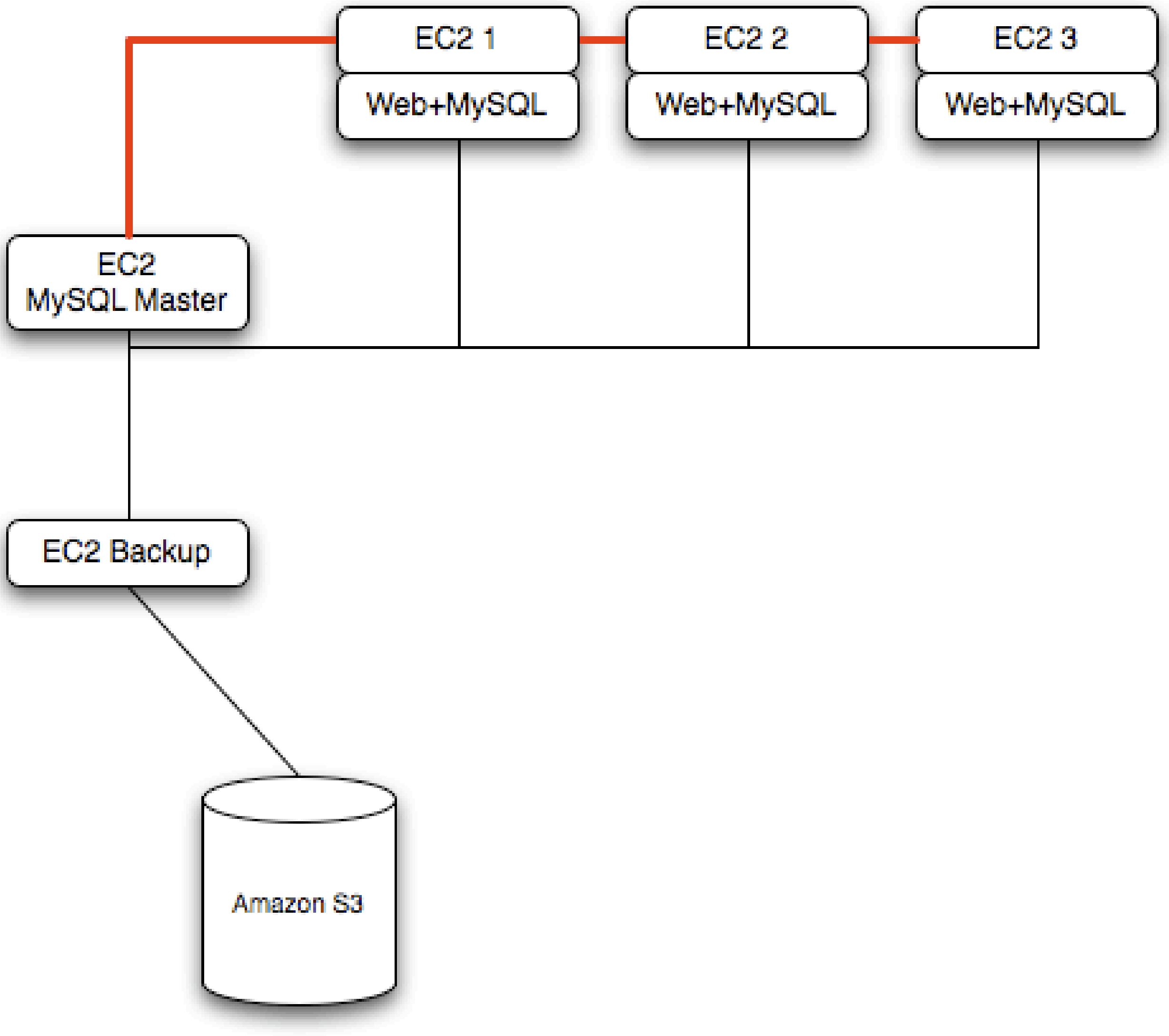


You can reproduce this structure completely within the EC2 environment, using an EC2 instance for the master, and one instance for each of the web and MySQL slave servers.

Note

Within the EC2 environment, internal (private) IP addresses used by the EC2 instances are constant. Always use these internal addresses and names when communicating between instances. Only use public IP addresses when communicating with the outside world - for example, when publicizing your application.

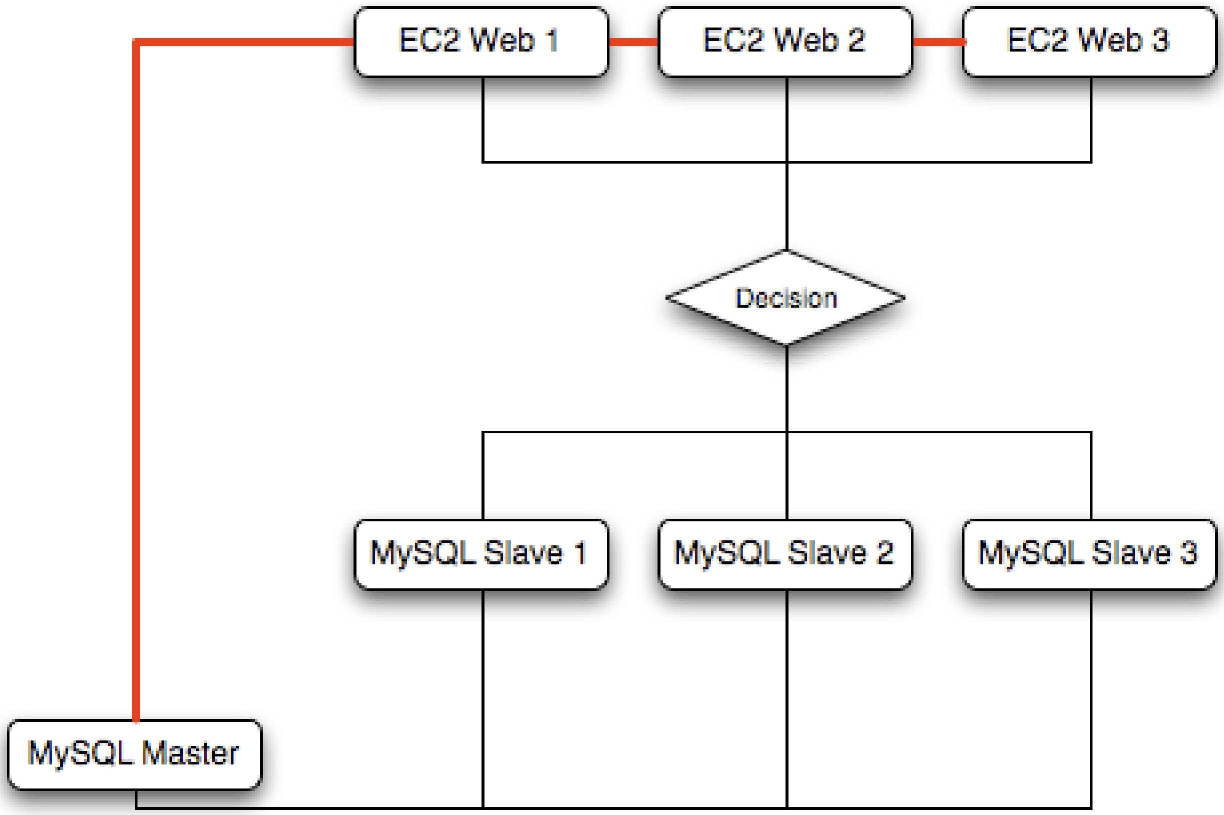
To ensure reliability of your database, add at least one replication slave dedicated to providing an active backup and storage to the Amazon S3 facility. You can see an example of this in the following topology.



Using `memcached` within your EC2 instances should provide better performance. The large and extra large instances have a significant amount of RAM. To use `memcached` in your application, when loading information from the database, first check whether the item exists in the cache. If the data you are looking for exists in the cache, use it. If not, reload the data from the database and populate the cache.

Sharding divides up data in your entire database by allocating individual machines or machine groups to provide a unique set of data according to an appropriate group. For example, you might put all users with a surname ending in the letters A-D onto a single server. When a user connects to the application and their surname is known, queries can be redirected to the appropriate MySQL server.

When using sharding with EC2, separate the web server and MySQL server into separate EC2 instances, and then apply the sharding decision logic into your application. Once you know which MySQL server you should be using for accessing the data you then distribute queries to the appropriate server. You can see a sample of this in the following illustration.



Warning

With sharding and EC2, be careful that the potential for failure of an instance does not affect your application. If the EC2 instance that provides the MySQL server for a particular shard fails, then all of the data on that shard becomes unavailable.

15.5 Using ZFS Replication

To support high availability environments, providing an instant copy of the information on both the currently active machine and the hot backup is a critical part of the HA solution. There are many solutions to this problem, including Chapter 16, *Replication* and Section 15.2, “Overview of MySQL with DRBD/Pacemaker/Corosync/Oracle Linux”.

The ZFS file system provides functionality to create a snapshot of the file system contents, transfer the snapshot to another machine, and extract the snapshot to recreate the file system. You can create a snapshot at any time, and you can create as many snapshots as you like. By continually creating, transferring, and restoring snapshots, you can provide synchronization between one or more machines in a fashion similar to DRBD.

The following example shows a simple Solaris system running with a single ZFS pool, mounted at `/scratchpool`:

```
Filesystem             size   used  avail capacity  Mounted on
/dev/dsk/c0d0s0        4.6G   3.7G   886M    82%    /
/devices                 0K     0K     0K     0%    /devices
```

```
ctfs                     0K     0K     0K     0%    /system/contract
proc                     0K     0K     0K     0%    /proc
mnttab                   0K     0K     0K     0%    /etc/mnttab
swap                   1.4G   892K   1.4G     1%    /etc/svc/volatile
objfs                    0K     0K     0K     0%    /system/object
/usr/lib/libc/libc_hwcap1.so.1
                       4.6G   3.7G   886M    82%    /lib/libc.so.1
fd                       0K     0K     0K     0%    /dev/fd
swap                   1.4G    40K   1.4G     1%    /tmp
swap                   1.4G    28K   1.4G     1%    /var/run
/dev/dsk/c0d0s7         26G   913M    25G     4%    /export/home
scratchpool             16G    24K    16G     1%    /scratchpool
```

The MySQL data is stored in a directory on `/scratchpool`. To help demonstrate some of the basic replication functionality, there are also other items stored in `/scratchpool` as well:

```
total 17
drwxr-xr-x  31 root     bin           50 Jul 21 07:32 DTT/
drwxr-xr-x   4 root     bin            5 Jul 21 07:32 SUNWmlib/
drwxr-xr-x  14 root     sys           16 Nov  5 09:56 SUNWspro/
drwxrwxrwx  19 1000     1000          40 Nov  6 19:16 emacs-22.1/
```

To create a snapshot of the file system, you use `zfs snapshot`, specifying the pool and the snapshot name:

```
root-shell> zfs snapshot scratchpool@snap1
```

To list the snapshots already taken:

```
root-shell> zfs list -t snapshot
NAME                USED  AVAIL  REFER  MOUNTPOINT
scratchpool@snap1      0      -  24.5K  -
scratchpool@snap2      0      -  24.5K  -
```

The snapshots themselves are stored within the file system metadata, and the space required to keep them varies as time goes on because of the way the snapshots are created. The initial creation of a snapshot is very quick, because instead of taking an entire copy of the data and metadata required to hold the entire snapshot, ZFS records only the point in time and metadata of when the snapshot was created.

As more changes to the original file system are made, the size of the snapshot increases because more space is required to keep the record of the old blocks. If you create lots of snapshots, say one per day, and then delete the snapshots from earlier in the week, the size of the newer snapshots might also increase, as the changes that make up the newer state have to be included in the more recent snapshots, rather than being spread over the seven snapshots that make up the week.

You cannot directly back up the snapshots because they exist within the file system metadata rather than as regular files. To get the snapshot into a format that you can copy to another file system, tape, and so on, you use the `zfs send` command to create a stream version of the snapshot.

For example, to write the snapshot out to a file:

```
root-shell> zfs send scratchpool@snap1 >/backup/scratchpool-snap1
```

Or tape:

```
root-shell> zfs send scratchpool@snap1 >/dev/rmt/0
```

You can also write out the incremental changes between two snapshots using `zfs send`:

```
root-shell> zfs send scratchpool@snap1 scratchpool@snap2 >/backup/scratchpool-changes
```

To recover a snapshot, you use `zfs recv`, which applies the snapshot information either to a new file system, or to an existing one.

15.5.1 Using ZFS for File System Replication

Because `zfs send` and `zfs recv` use streams to exchange data, you can use them to replicate information from one system to another by combining `zfs send`, `ssh`, and `zfs recv`.

For example, to copy a snapshot of the `scratchpool` file system to a new file system called `slavepool` on a new server, you would use the following command. This sequence combines the snapshot of `scratchpool`, the transmission to the slave machine (using `ssh` with login credentials), and the recovery of the snapshot on the slave using `zfs recv`:

```
root-shell> zfs send scratchpool@snap1 |ssh id@host pfexec zfs recv -F slavepool
```

The first part of the pipeline, `zfs send scratchpool@snap1`, streams the snapshot. The `ssh` command, and the command that it executes on the other server, `pfexec zfs recv -F slavepool`, receives the streamed snapshot data and writes it to slavepool. In this instance, I've specified the `-F` option which forces the snapshot data to be applied, and is therefore destructive. This is fine, as I'm creating the first version of my replicated file system.

On the slave machine, the replicated file system contains the exact same content:

```
root-shell> ls -al /slavepool/
total 23
drwxr-xr-x   6 root     root           7 Nov  8 09:13 ./
drwxr-xr-x  29 root     root          34 Nov  9 07:06 ../
drwxr-xr-x  31 root     bin           50 Jul 21 07:32 DTT/
drwxr-xr-x   4 root     bin            5 Jul 21 07:32 SUNWmlib/
drwxr-xr-x  14 root     sys           16 Nov  5 09:56 SUNWspro/
drwxrwxrwx  19 1000     1000          40 Nov  6 19:16 emacs-22.1/
```

Once a snapshot has been created, to synchronize the file system again, you create a new snapshot and then use the incremental snapshot feature of `zfs send` to send the changes between the two snapshots to the slave machine again:

```
root-shell> zfs send -i scratchpool@snapshot1 scratchpool@snapshot2 |ssh id@host pfexec zfs recv slavepool
```

This operation only succeeds if the file system on the slave machine has not been modified at all. You cannot apply the incremental changes to a destination file system that has changed. In the example above, the `ls` command would cause problems by changing the metadata, such as the last access time for files or directories.

To prevent changes on the slave file system, set the file system on the slave to be read-only:

```
root-shell> zfs set readonly=on slavepool
```

Setting `readonly` means that you cannot change the file system on the slave by normal means, including the file system metadata. Operations that would normally update metadata (like our `ls`) silently perform their function without attempting to update the file system state.

In essence, the slave file system is nothing but a static copy of the original file system. However, even when configured to to be read-only, a file system can have snapshots applied to it. With the file system set to read only, re-run the initial copy:

```
root-shell> zfs send scratchpool@snap1 |ssh id@host pfexec zfs recv -F slavepool
```

Now you can make changes to the original file system and replicate them to the slave.

15.5.2 Configuring MySQL for ZFS Replication

Configuring MySQL on the source file system is a case of creating the data on the file system that you intend to replicate. The configuration file in the example below has been updated to use `/scratchpool/mysql-data` as the data directory, and now you can initialize the tables:

```
root-shell> mysql_install_db --defaults-file=/etc/mysql/5.5/my.cnf --user=mysql
```

To synchronize the initial information, perform a new snapshot and then send an incremental snapshot to the slave using `zfs send`:

```
root-shell> zfs snapshot scratchpool@snap2
root-shell> zfs send -i scratchpool@snap1 scratchpool@snap2|ssh id@host pfexec zfs recv slavepool
```

Doublecheck that the slave has the data by looking at the MySQL data directory on the `slavepool`:

```
root-shell> ls -al /slavepool/mysql-data/
```

Now you can start up MySQL, create some data, and then replicate the changes using `zfs send`/ `zfs recv` to the slave to synchronize the changes.

The rate at which you perform the synchronization depends on your application and environment. The limitation is the speed required to perform the snapshot and then to send the changes over the network.

To automate the process, create a script that performs the snapshot, send, and receive operation, and use `cron` to synchronize the changes at set times or intervals.

15.5.3 Handling MySQL Recovery with ZFS

When using ZFS replication to provide a constant copy of your data, ensure that you can recover your tables, either manually or automatically, in the event of a failure of the original system.

In the event of a failure, follow this sequence:

1. Stop the script on the master, if it is still up and running.

2. Set the slave file system to be read/write:

   ```
   root-shell> zfs set readonly=off slavepool
   ```

3. Start up `mysqld` on the slave. If you are using `InnoDB`, you get auto-recovery, if it is needed, to make sure the table data is correct, as shown here when I started up from our mid-INSERT snapshot:

   ```
   InnoDB: The log sequence number in ibdata files does not match
   InnoDB: the log sequence number in the ib_logfiles!
   081109 15:59:59  InnoDB: Database was not shut down normally!
   InnoDB: Starting crash recovery.
   InnoDB: Reading tablespace information from the .ibd files...
   InnoDB: Restoring possible half-written data pages from the doublewrite
   InnoDB: buffer...
   081109 16:00:03  InnoDB: Started; log sequence number 0 1142807951
   ```

```
081109 16:00:03 [Note] /slavepool/mysql-5.0.67-solaris10-i386/bin/mysqld: ready for connections.
Version: '5.0.67'  socket: '/tmp/mysql.sock'  port: 3306  MySQL Community Server (GPL)
```

Use InnoDB tables and a regular synchronization schedule to reduce the risk for significant data loss. On MyISAM tables, you might need to run REPAIR TABLE, and you might even have lost some information.

15.6 Using MySQL with memcached

memcached is a simple, highly scalable key-based cache that stores data and objects wherever dedicated or spare RAM is available for quick access by applications, without going through layers of parsing or disk I/O. To use, you run the memcached command on one or more hosts and then use the shared cache to store objects. For more usage instructions, see Section 15.6.2, “Using memcached”

Benefits of using memcached include:

- Because all information is stored in RAM, the access speed is faster than loading the information each time from disk.

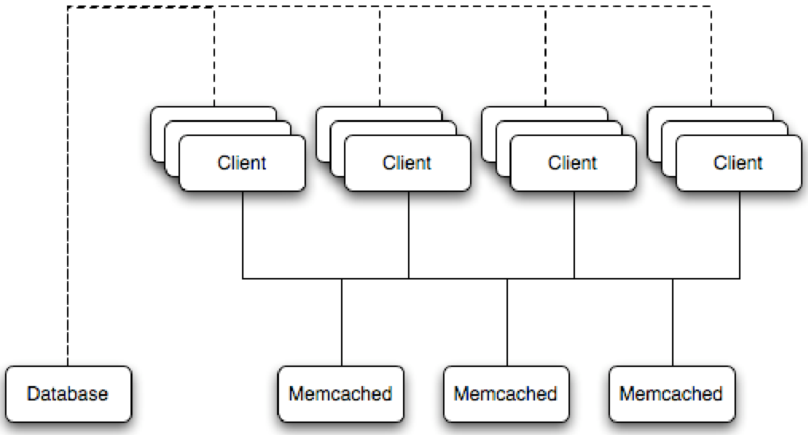
- Because the “value” portion of the key-value pair does not have any data type restrictions, you can cache data such as complex structures, documents, images, or a mixture of such things.

- If you use the in-memory cache to hold transient information, or as a read-only cache for information also stored in a database, the failure of any memcached server is not critical. For persistent data, you can fall back to an alternative lookup method using database queries, and reload the data into RAM on a different server.

The typical usage environment is to modify your application so that information is read from the cache provided by memcached. If the information is not in memcached, then the data is loaded from the MySQL database and written into the cache so that future requests for the same object benefit from the cached data.

For a typical deployment layout, see Figure 15.4, “memcached Architecture Overview”.

Figure 15.4 memcached Architecture Overview



In the example structure, any of the clients can contact one of the memcached servers to request a given key. Each client is configured to talk to all of the servers shown in the illustration. Within the client, when the request is made to store the information, the key used to reference the data is hashed and this hash is then used to select one of the memcached servers. The selection of the memcached server takes place on the client before the server is contacted, keeping the process lightweight.

The same algorithm is used again when a client requests the same key. The same key generates the same hash, and the same memcached server is selected as the source for the data. Using this method, the cached data is spread among all of the memcached servers, and the cached information is accessible from any client. The result is a distributed, memory-based, cache that can return information, particularly complex data and structures, much faster than natively reading the information from the database.

The data held within a traditional memcached server is never stored on disk (only in RAM, which means there is no persistence of data), and the RAM cache is always populated from the backing store (a MySQL database). If a memcached server fails, the data can always be recovered from the MySQL database.

15.6.1 Installing memcached

You can build and install memcached from the source code directly, or you can use an existing operating system package or installation.

Installing memcached from a Binary Distribution

To install memcached on a Red Hat, or Fedora host, use yum:

```
root-shell> yum install memcached
```

Note

On CentOS, you may be able to obtain a suitable RPM from another source, or use the source tarball.

To install memcached on a Debian or Ubuntu host, use apt-get:

```
root-shell> apt-get install memcached
```

To install memcached on a Gentoo host, use emerge:

```
root-shell> emerge install memcached
```

Building memcached from Source

On other Unix-based platforms, including Solaris, AIX, HP-UX and Mac OS X, and Linux distributions not mentioned already, you must install from source. For Linux, make sure you have a 2.6-based kernel, which includes the improved epoll interface. For all platforms, ensure that you have libevent 1.1 or higher installed. You can obtain libevent from libevent web page.

You can obtain the source for memcached from memcached Web site.

To build memcached, follow these steps:

1. Extract the memcached source package:

   ```
   shell> gunzip -c memcached-1.2.5.tar.gz | tar xf -
   ```

2. Change to the memcached-*1.2.5* directory:

   ```
   shell> cd memcached-1.2.5
   ```

3. Run configure

```
shell> ./configure
```

Some additional options you might specify to the configure:

- --prefix

  To specify a different installation directory, use the --prefix option:

  ```
  shell> ./configure --prefix=/opt
  ```

  The default is to use the /usr/local directory.

- --with-libevent

  If you have installed libevent and configure cannot find the library, use the --with-libevent option to specify the location of the installed library.

- --enable-64bit

  To build a 64-bit version of memcached (which enables you to use a single instance with a large RAM allocation), use --enable-64bit.

- --enable-threads

  To enable multi-threading support in memcached, which improves the response times on servers with a heavy load, use --enable-threads. You must have support for the POSIX threads within your operating system to enable thread support. For more information on the threading support, see Section 15.6.2.7, “memcached Thread Support”.

- --enable-dtrace

  memcached includes a range of DTrace threads that can be used to monitor and benchmark a memcached instance. For more information, see Section 15.6.2.5, “Using memcached and DTrace”.

4. Run make to build memcached:

   ```
   shell> make
   ```

5. Run make install to install memcached:

   ```
   shell> make install
   ```

15.6.2 Using memcached

To start using memcached, start the memcached service on one or more servers. Running memcached sets up the server, allocates the memory and starts listening for connections from clients.

> **Note**
>
> You do not need to be a privileged user (root) to run memcached except to listen on one of the privileged TCP/IP ports (below 1024). You must, however, use a user that has not had their memory limits restricted using setrlimit or similar.

To start the server, run memcached as a nonprivileged (that is, non-root) user:

```
shell> memcached
```

By default, memcached uses the following settings:

- Memory allocation of 64MB
- Listens for connections on all network interfaces, using port 11211
- Supports a maximum of 1024 simultaneous connections

Typically, you would specify the full combination of options that you want when starting memcached, and normally provide a startup script to handle the initialization of memcached. For example, the following line starts memcached with a maximum of 1024MB RAM for the cache, listening on port 11211 on the IP address 192.168.0.110, running as a background daemon:

```
shell> memcached -d -m 1024 -p 11211 -l 192.168.0.110
```

To ensure that memcached is started up on boot, check the init script and configuration parameters.

memcached supports the following options:

- `-u user`

  If you start memcached as root, use the -u option to specify the user for executing memcached:

  ```
  shell> memcached -u memcache
  ```

- `-m memory`

  Set the amount of memory allocated to memcached for object storage. Default is 64MB.

  To increase the amount of memory allocated for the cache, use the -m option to specify the amount of RAM to be allocated (in megabytes). The more RAM you allocate, the more data you can store and therefore the more effective your cache is.

  Warning

  Do not specify a memory allocation larger than your available RAM. If you specify too large a value, then some RAM allocated for memcached uses swap space, and not physical RAM. This may lead to delays when storing and retrieving values, because data is swapped to disk, instead of storing the data directly in RAM.

  You can use the output of the vmstat command to get the free memory, as shown in free column:

  ```
  shell> vmstat
  kthr      memory            page            disk          faults      cpu
  r b w   swap  free  re  mf pi po fr de sr s1 s2 -- --   in   sy   cs us sy id
  0 0 0 5170504 3450392 2  7  2  0  0  0  4  0  0  0  0  296   54  199  0  0 100
  ```

  For example, to allocate 3GB of RAM:

  ```
  shell> memcached -m 3072
  ```

  On 32-bit x86 systems where you are using PAE to access memory above the 4GB limit, you cannot allocate RAM beyond the maximum process size. You can get around this by running multiple instances of memcached, each listening on a different port:

```
shell> memcached -m 1024 -p11211
shell> memcached -m 1024 -p11212
shell> memcached -m 1024 -p11213
```

Note

On all systems, particularly 32-bit, ensure that you leave enough room for both memcached application in addition to the memory setting. For example, if you have a dedicated memcached host with 4GB of RAM, do not set the memory size above 3500MB. Failure to do this may cause either a crash or severe performance issues.

- `-l interface`

  Specify a network interface/address to listen for connections. The default is to listen on all available address (`INADDR_ANY`).

  ```
  shell> memcached -l 192.168.0.110
  ```

  Support for IPv6 address support was added in memcached 1.2.5.

- `-p port`

  Specify the TCP port to use for connections. Default is 18080.

  ```
  shell> memcached -p 18080
  ```

- `-U port`

  Specify the UDP port to use for connections. Default is 11211, 0 switches UDP off.

  ```
  shell> memcached -U 18080
  ```

- `-s socket`

  Specify a Unix socket to listen on.

  If you are running memcached on the same server as the clients, you can disable the network interface and use a local Unix socket using the `-s` option:

  ```
  shell> memcached -s /tmp/memcached
  ```

  Using a Unix socket automatically disables network support, and saves network ports (allowing more ports to be used by your web server or other process).

- `-a mask`

  Specify the access mask to be used for the Unix socket, in octal. Default is 0700.

- `-c connections`

  Specify the maximum number of simultaneous connections to the memcached service. The default is 1024.

```
shell> memcached -c 2048
```

Use this option, either to reduce the number of connections (to prevent overloading `memcached` service) or to increase the number to make more effective use of the server running `memcached` server.

- `-t threads`

  Specify the number of threads to use when processing incoming requests.

  By default, `memcached` is configured to use 4 concurrent threads. The threading improves the performance of storing and retrieving data in the cache, using a locking system to prevent different threads overwriting or updating the same values. To increase or decrease the number of threads, use the `-t` option:

  ```
  shell> memcached -t 8
  ```

- `-d`

  Run `memcached` as a daemon (background) process:

  ```
  shell> memcached -d
  ```

- `-r`

  Maximize the size of the core file limit. In the event of a failure, this attempts to dump the entire memory space to disk as a core file, up to any limits imposed by `setrlimit`.

- `-M`

  Return an error to the client when the memory has been exhausted. This replaces the normal behavior of removing older items from the cache to make way for new items.

- `-k`

  Lock down all paged memory. This reserves the memory before use, instead of allocating new slabs of memory as new items are stored in the cache.

  Note

  There is a user-level limit on how much memory you can lock. Trying to allocate more than the available memory fails. You can set the limit for the user you started the daemon with (not for the `-u user` user) within the shell by using `ulimit -S -l NUM_KB`

- `-v`

  Verbose mode. Prints errors and warnings while executing the main event loop.

- `-vv`

  Very verbose mode. In addition to information printed by `-v`, also prints each client command and the response.

- `-vvv`

  Extremely verbose mode. In addition to information printed by `-vv`, also show the internal state transitions.

- -h

  Print the help message and exit.

- -i

  Print the memcached and libevent license.

- -I mem

  Specify the maximum size permitted for storing an object within the memcached instance. The size supports a unit postfix (k for kilobytes, m for megabytes). For example, to increase the maximum supported object size to 32MB:

  ```
  shell> memcached -I 32m
  ```

  The maximum object size you can specify is 128MB, the default remains at 1MB.

  This option was added in 1.4.2.

- -b

  Set the backlog queue limit. The backlog queue configures how many network connections can be waiting to be processed by memcached. Increasing this limit may reduce errors received by the client that it is not able to connect to the memcached instance, but does not improve the performance of the server. The default is 1024.

- -P pidfile

  Save the process ID of the memcached instance into file.

- -f

  Set the chunk size growth factor. When allocating new memory chunks, the allocated size of new chunks is determined by multiplying the default slab size by this factor.

  To see the effects of this option without extensive testing, use the -vv command-line option to show the calculated slab sizes. For more information, see Section 15.6.2.8, “memcached Logs”.

- -n bytes

  The minimum space allocated for the key+value+flags information. The default is 48 bytes.

- -L

  On systems that support large memory pages, enables large memory page use. Using large memory pages enables memcached to allocate the item cache in one large chunk, which can improve the performance by reducing the number misses when accessing memory.

- -C

  Disable the use of compare and swap (CAS) operations.

  This option was added in memcached 1.3.x.

- -D char

Set the default character to be used as a delimiter between the key prefixes and IDs. This is used for the per-prefix statistics reporting (see Section 15.6.4, “Getting memcached Statistics”). The default is the colon (`:`). If this option is used, statistics collection is turned on automatically. If not used, you can enable stats collection by sending the `stats detail on` command to the server.

This option was added in `memcached` 1.3.x.

- `-R num`

  Sets the maximum number of requests per event process. The default is 20.

- `-B protocol`

  Set the binding protocol, that is, the default `memcached` protocol support for client connections. Options are `ascii`, `binary` or `auto`. Automatic (`auto`) is the default.

  This option was added in `memcached` 1.4.0.

15.6.2.1 `memcached` Deployment

When using `memcached` you can use a number of different potential deployment strategies and topologies. The exact strategy to use depends on your application and environment. When developing a system for deploying `memcached` within your system, keep in mind the following points:

- `memcached` is only a caching mechanism. It shouldn't be used to store information that you cannot otherwise afford to lose and then load from a different location.

- There is no security built into the `memcached` protocol. At a minimum, make sure that the servers running `memcached` are only accessible from inside your network, and that the network ports being used are blocked (using a firewall or similar). If the information on the `memcached` servers that is being stored is any sensitive, then encrypt the information before storing it in `memcached`.

- `memcached` does not provide any sort of failover. Because there is no communication between different `memcached` instances. If an instance fails, your application must capable of removing it from the list, reloading the data and then writing data to another `memcached` instance.

- Latency between the clients and the `memcached` can be a problem if you are using different physical machines for these tasks. If you find that the latency is a problem, move the `memcached` instances to be on the clients.

- Key length is determined by the `memcached` server. The default maximum key size is 250 bytes.

- Try to use at least two `memcached` instances, especially for multiple clients, to avoid having a single point of failure. Ideally, create as many `memcached` nodes as possible. When adding and removing `memcached` instances from a pool, the hashing and distribution of key/value pairs may be affected. For information on how to avoid problems, see Section 15.6.2.4, “memcached Hashing/Distribution Types”.

15.6.2.2 Using Namespaces

The `memcached` cache is a very simple massive key/value storage system, and as such there is no way of compartmentalizing data automatically into different sections. For example, if you are storing information by the unique ID returned from a MySQL database, then storing the data from two different tables could run into issues because the same ID might be valid in both tables.

Some interfaces provide an automated mechanism for creating *namespaces* when storing information into the cache. In practice, these namespaces are merely a prefix before a given ID that is applied every time a value is stored or retrieve from the cache.

You can implement the same basic principle by using keys that describe the object and the unique identifier within the key that you supply when the object is stored. For example, when storing user data, prefix the ID of the user with `user:` or `user-`.

Note

Using namespaces or prefixes only controls the keys stored/retrieved. There is no security within `memcached`, and therefore no way to enforce that a particular client only accesses keys with a particular namespace. Namespaces are only useful as a method of identifying data and preventing corruption of key/value pairs.

15.6.2.3 Data Expiry

There are two types of data expiry within a `memcached` instance. The first type is applied at the point when you store a new key/value pair into the `memcached` instance. If there is not enough space within a suitable slab to store the value, then an existing least recently used (LRU) object is removed (evicted) from the cache to make room for the new item.

The LRU algorithm ensures that the object that is removed is one that is either no longer in active use or that was used so long ago that its data is potentially out of date or of little value. However, in a system where the memory allocated to `memcached` is smaller than the number of regularly used objects required in the cache, a lot of expired items could be removed from the cache even though they are in active use. You use the statistics mechanism to get a better idea of the level of evictions (expired objects). For more information, see Section 15.6.4, “Getting `memcached` Statistics”.

You can change this eviction behavior by setting the `-M` command-line option when starting `memcached`. This option forces an error to be returned when the memory has been exhausted, instead of automatically evicting older data.

The second type of expiry system is an explicit mechanism that you can set when a key/value pair is inserted into the cache, or when deleting an item from the cache. Using an expiration time can be a useful way of ensuring that the data in the cache is up to date and in line with your application needs and requirements.

A typical scenario for explicitly setting the expiry time might include caching session data for a user when accessing a Web site. `memcached` uses a lazy expiry mechanism where the explicit expiry time that has been set is compared with the current time when the object is requested. Only objects that have not expired are returned.

You can also set the expiry time when explicitly deleting an object from the cache. In this case, the expiry time is really a timeout and indicates the period when any attempts to set the value for a given key are rejected.

15.6.2.4 `memcached` Hashing/Distribution Types

The `memcached` client interface supports a number of different distribution algorithms that are used in multi-server configurations to determine which host should be used when setting or getting data from a given `memcached` instance. When you get or set a value, a hash is constructed from the supplied key and then used to select a host from the list of configured servers. Because the hashing mechanism uses the supplied key as the basis for the hash, the same server is selected during both set and get operations.

You can think of this process as follows. Given an array of servers (a, b, and c), the client uses a hashing algorithm that returns an integer based on the key being stored or retrieved. The resulting value is then used to select a server from the list of servers configured in the client. Most standard client hashing within `memcache` clients uses a simple modulus calculation on the value against the number of configured `memcached` servers. You can summarize the process in pseudocode as:

```
@memcservers = ['a.memc','b.memc','c.memc'];
$value = hash($key);
$chosen = $value % length(@memcservers);
```

Replacing the above with values:

```
@memcservers = ['a.memc','b.memc','c.memc'];
$value = hash('myid');
$chosen = 7009 % 3;
```

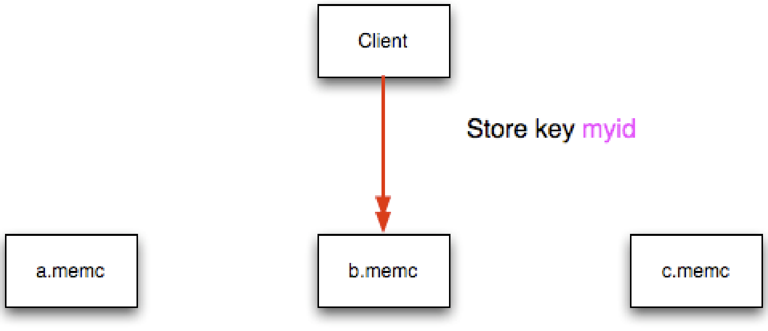
In the above example, the client hashing algorithm chooses the server at index 1 (`7009 % 3 = 1`), and store or retrieve the key and value with that server.

Note

This selection and hashing process is handled automatically by the `memcached` client you are using; you need only provide the list of `memcached` servers to use.

You can see a graphical representation of this below in Figure 15.5, "`memcached` Hash Selection".

Figure 15.5 `memcached` Hash Selection



The same hashing and selection process takes place during any operation on the specified key within the `memcached` client.

Using this method provides a number of advantages:

- The hashing and selection of the server to contact is handled entirely within the client. This eliminates the need to perform network communication to determine the right machine to contact.
- Because the determination of the `memcached` server occurs entirely within the client, the server can be selected automatically regardless of the operation being executed (set, get, increment, etc.).
- Because the determination is handled within the client, the hashing algorithm returns the same value for a given key; values are not affected or reset by differences in the server environment.
- Selection is very fast. The hashing algorithm on the key value is quick and the resulting selection of the server is from a simple array of available machines.
- Using client-side hashing simplifies the distribution of data over each `memcached` server. Natural distribution of the values returned by the hashing algorithm means that keys are automatically spread over the available servers.

Providing that the list of servers configured within the client remains the same, the same stored key returns the same value, and therefore selects the same server.

However, if you do not use the same hashing mechanism then the same data may be recorded on different servers by different interfaces, both wasting space on your memcached and leading to potential differences in the information.

Note

One way to use a multi-interface compatible hashing mechanism is to use the libmemcached library and the associated interfaces. Because the interfaces for the different languages (including C, Ruby, Perl and Python) use the same client library interface, they always generate the same hash code from the ID.

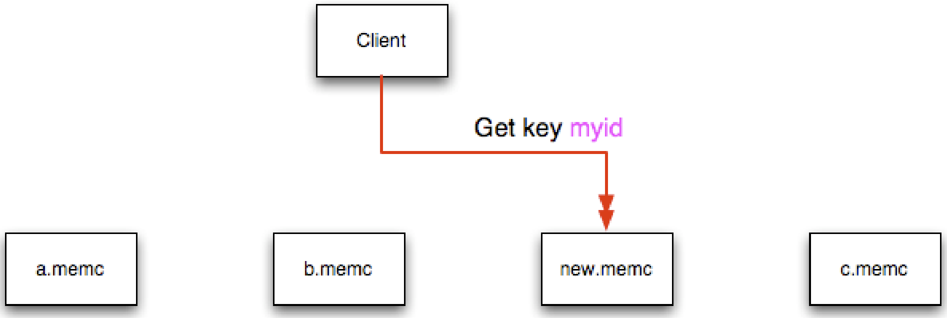
The problem with client-side selection of the server is that the list of the servers (including their sequential order) *must* remain consistent on each client using the memcached servers, and the servers must be available. If you try to perform an operation on a key when:

- A new memcached instance has been added to the list of available instances
- A memcached instance has been removed from the list of available instances
- The order of the memcached instances has changed

When the hashing algorithm is used on the given key, but with a different list of servers, the hash calculation may choose a different server from the list.

If a new memcached instance is added into the list of servers, as new.memc is in the example below, then a GET operation using the same key, myid, can result in a cache-miss. This is because the same value is computed from the key, which selects the same index from the array of servers, but index 2 now points to the new server, not the server c.memc where the data was originally stored. This would result in a cache miss, even though the key exists within the cache on another memcached instance.

Figure 15.6 memcached Hash Selection with New memcached instance



This means that servers c.memc and new.memc both contain the information for key myid, but the information stored against the key in eachs server may be different in each instance. A more significant problem is a much higher number of cache-misses when retrieving data, as the addition of a new server changes the distribution of keys, and this in turn requires rebuilding the cached data on the memcached instances, causing an increase in database reads.

The same effect can occur if you actively manage the list of servers configured in your clients, adding and removing the configured memcached instances as each instance is identified as being available. For example, removing a memcached instance when the client notices that the instance can no longer be contacted can cause the server selection to fail as described here.

To prevent this causing significant problems and invalidating your cache, you can select the hashing algorithm used to select the server. There are two common types of hashing algorithm, *consistent* and *modula*.

With *consistent* hashing algorithms, the same key when applied to a list of servers always uses the same server to store or retrieve the keys, even if the list of configured servers changes. This means that you can add and remove servers from the configure list and always use the same server for a given key. There are two types of consistent hashing algorithms available, Ketama and Wheel. Both types are supported by libmemcached, and implementations are available for PHP and Java.

Any consistent hashing algorithm has some limitations. When you add servers to an existing list of configured servers, keys are distributed to the new servers as part of the normal distribution. When you remove servers from the list, the keys are re-allocated to another server within the list, meaning that the cache needs to be re-populated with the information. Also, a consistent hashing algorithm does not resolve the issue where you want consistent selection of a server across multiple clients, but where each client contains a different list of servers. The consistency is enforced only within a single client.

With a *modula* hashing algorithm, the client selects a server by first computing the hash and then choosing a server from the list of configured servers. As the list of servers changes, so the server selected when using a modula hashing algorithm also changes. The result is the behavior described above; changes to the list of servers mean that different servers are selected when retrieving data, leading to cache misses and increase in database load as the cache is re-seeded with information.

If you use only a single memcached instance for each client, or your list of memcached servers configured for a client never changes, then the selection of a hashing algorithm is irrelevant, as it has no noticeable effect.

If you change your servers regularly, or you use a common set of servers that are shared among a large number of clients, then using a consistent hashing algorithm should help to ensure that your cache data is not duplicated and the data is evenly distributed.

15.6.2.5 Using memcached and DTrace

memcached includes a number of different DTrace probes that can be used to monitor the operation of the server. The probes included can monitor individual connections, slab allocations, and modifications to the hash table when a key/value pair is added, updated, or removed.

For more information on DTrace and writing DTrace scripts, read the DTrace User Guide.

Support for DTrace probes was added to memcached 1.2.6 includes a number of DTrace probes that can be used to help monitor your application. DTrace is supported on Solaris 10, OpenSolaris, Mac OS X 10.5 and FreeBSD. To enable the DTrace probes in memcached, build from source and use the --enable-dtrace option. For more information, see Section 15.6.1, “Installing memcached”.

The probes supported by memcached are:

- conn-allocate(connid)

  Fired when a connection object is allocated from the connection pool.

  - connid: The connection ID.

- conn-release(connid)

  Fired when a connection object is released back to the connection pool.

  Arguments:

  - `connid`: The connection ID.

- `conn-create(ptr)`

  Fired when a new connection object is being created (that is, there are no free connection objects in the connection pool).

  Arguments:

  - `ptr`: A pointer to the connection. object

- `conn-destroy(ptr)`

  Fired when a connection object is being destroyed.

  Arguments:

  - `ptr`: A pointer to the connection object.

- `conn-dispatch(connid, threadid)`

  Fired when a connection is dispatched from the main or connection-management thread to a worker thread.

  Arguments:

  - `connid`: The connection ID.
  - `threadid`: The thread ID.

- `slabs-allocate(size, slabclass, slabsize, ptr)`

  Allocate memory from the slab allocator.

  Arguments:

  - `size`: The requested size.
  - `slabclass`: The allocation is fulfilled in this class.
  - `slabsize`: The size of each item in this class.
  - `ptr`: A pointer to allocated memory.

- `slabs-allocate-failed(size, slabclass)`

  Failed to allocate memory (out of memory).

  Arguments:

  - `size`: The requested size.
  - `slabclass`: The class that failed to fulfill the request.

- `slabs-slabclass-allocate(slabclass)`

  Fired when a slab class needs more space.

  Arguments:

  - `slabclass`: The class that needs more memory.

- `slabs-slabclass-allocate-failed(slabclass)`

  Failed to allocate memory (out of memory).

  Arguments:

  - `slabclass`: The class that failed to grab more memory.

- `slabs-free(size, slabclass, ptr)`

  Release memory.

  Arguments:

  - `size`: The amount of memory to release, in bytes.
  - `slabclass`: The class the memory belongs to.
  - `ptr`: A pointer to the memory to release.

- `assoc-find(key, depth)`

  Fired when we have searched the hash table for a named key. These two elements provide an insight into how well the hash function operates. Traversals are a sign of a less optimal function, wasting CPU capacity.

  Arguments:

  - `key`: The key searched for.
  - `depth`: The depth in the list of hash table.

- `assoc-insert(key, nokeys)`

  Fired when a new item has been inserted.

  Arguments:

  - `key`: The key just inserted.
  - `nokeys`: The total number of keys currently being stored, including the key for which insert was called.

- `assoc-delete(key, nokeys)`

  Fired when a new item has been removed.

  Arguments:

  - `key`: The key just deleted.
  - `nokeys`: The total number of keys currently being stored, excluding the key for which delete was called.

- `item-link(key, size)`

Fired when an item is being linked in the cache.

Arguments:

- `key`: The items key.
- `size`: The size of the data.

- `item-unlink(key, size)`

  Fired when an item is being deleted.

  Arguments:

  - `key`: The items key.
  - `size`: The size of the data.

- `item-remove(key, size)`

  Fired when the refcount for an item is reduced.

  Arguments:

  - `key`: The item's key.
  - `size`: The size of the data.

- `item-update(key, size)`

  Fired when the "last referenced" time is updated.

  Arguments:

  - `key`: The item's key.
  - `size`: The size of the data.

- `item-replace(oldkey, oldsize, newkey, newsize)`

  Fired when an item is being replaced with another item.

  Arguments:

  - `oldkey`: The key of the item to replace.
  - `oldsize`: The size of the old item.
  - `newkey`: The key of the new item.
  - `newsize`: The size of the new item.

- `process-command-start(connid, request, size)`

  Fired when the processing of a command starts.

  Arguments:

  - `connid`: The connection ID.
  - `request`: The incoming request.
  - `size`: The size of the request.

- `process-command-end(connid, response, size)`

  Fired when the processing of a command is done.

  Arguments:

  - `connid`: The connection ID.
  - `response`: The response to send back to the client.
  - `size`: The size of the response.

- `command-get(connid, key, size)`

  Fired for a `get` command.

  Arguments:

  - `connid`: The connection ID.
  - `key`: The requested key.
  - `size`: The size of the key's data (or -1 if not found).

- `command-gets(connid, key, size, casid)`

  Fired for a `gets` command.

  Arguments:

  - `connid`: The connection ID.
  - `key`: The requested key.
  - `size`: The size of the key's data (or -1 if not found).
  - `casid`: The casid for the item.

- `command-add(connid, key, size)`

  Fired for a `add` command.

  Arguments:

  - `connid`: The connection ID.
  - `key`: The requested key.
  - `size`: The new size of the key's data (or -1 if not found).

- `command-set(connid, key, size)`

Fired for a `set` command.

Arguments:

  - `connid`: The connection ID.
  - `key`: The requested key.
  - `size`: The new size of the key's data (or -1 if not found).

- `command-replace(connid, key, size)`

  Fired for a `replace` command.

  Arguments:

  - `connid`: The connection ID.
  - `key`: The requested key.
  - `size`: The new size of the key's data (or -1 if not found).

- `command-prepend(connid, key, size)`

  Fired for a `prepend` command.

  Arguments:

  - `connid`: The connection ID.
  - `key`: The requested key.
  - `size`: The new size of the key's data (or -1 if not found).

- `command-append(connid, key, size)`

  Fired for a `append` command.

  Arguments:

  - `connid`: The connection ID.
  - `key`: The requested key.
  - `size`: The new size of the key's data (or -1 if not found).

- `command-cas(connid, key, size, casid)`

  Fired for a `cas` command.

  Arguments:

  - `connid`: The connection ID.
  - `key`: The requested key.
  - `size`: The size of the key's data (or -1 if not found).

  - casid: The cas ID requested.

- command-incr(connid, key, val)

  Fired for incr command.

  Arguments:

  - connid: The connection ID.
  - key: The requested key.
  - val: The new value.

- command-decr(connid, key, val)

  Fired for decr command.

  Arguments:

  - connid: The connection ID.
  - key: The requested key.
  - val: The new value.

- command-delete(connid, key, exptime)

  Fired for a delete command.

  Arguments:

  - connid: The connection ID.
  - key: The requested key.
  - exptime: The expiry time.

15.6.2.6 Memory Allocation within memcached

When you first start memcached, the memory that you have configured is not automatically allocated. Instead, memcached only starts allocating and reserving physical memory once you start saving information into the cache.

When you start to store data into the cache, memcached does not allocate the memory for the data on an item by item basis. Instead, a slab allocation is used to optimize memory usage and prevent memory fragmentation when information expires from the cache.

With slab allocation, memory is reserved in blocks of 1MB. The slab is divided up into a number of blocks of equal size. When you try to store a value into the cache, memcached checks the size of the value that you are adding to the cache and determines which slab contains the right size allocation for the item. If a slab with the item size already exists, the item is written to the block within the slab.

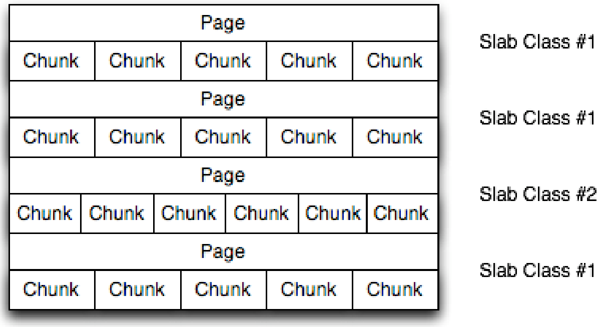
If the new item is bigger than the size of any existing blocks, then a new slab is created, divided up into blocks of a suitable size. If an existing slab with the right block size already exists, but there are no free blocks, a new slab is created. If you update an existing item with data that is larger than the existing block allocation for that key, then the key is re-allocated into a suitable slab.

For example, the default size for the smallest block is 88 bytes (40 bytes of value, and the default 48 bytes for the key and flag data). If the size of the first item you store into the cache is less than 40 bytes, then a slab with a block size of 88 bytes is created and the value stored.

If the size of the data that you intend to store is larger than this value, then the block size is increased by the chunk size factor until a block size large enough to hold the value is determined. The block size is always a function of the scale factor, rounded up to a block size which is exactly divisible into the chunk size.

For a sample of the structure, see Figure 15.7, "Memory Allocation in memcached".

Figure 15.7 Memory Allocation in memcached



The result is that you have multiple pages allocated within the range of memory allocated to memcached. Each page is 1MB in size (by default), and is split into a different number of chunks, according to the chunk size required to store the key/value pairs. Each instance has multiple pages allocated, and a page is always created when a new item needs to be created requiring a chunk of a particular size. A slab may consist of multiple pages, and each page within a slab contains an equal number of chunks.

The chunk size of a new slab is determined by the base chunk size combined with the chunk size growth factor. For example, if the initial chunks are 104 bytes in size, and the default chunk size growth factor is used (1.25), then the next chunk size allocated would be the best power of 2 fit for 104*1.25, or 136 bytes.

Allocating the pages in this way ensures that memory does not get fragmented. However, depending on the distribution of the objects that you store, it may lead to an inefficient distribution of the slabs and chunks if you have significantly different sized items. For example, having a relatively small number of items within each chunk size may waste a lot of memory with just few chunks in each allocated page.

You can tune the growth factor to reduce this effect by using the `-f` command line option, which adapts the growth factor applied to make more effective use of the chunks and slabs allocated. For information on how to determine the current slab allocation statistics, see Section 15.6.4.2, "memcached Slabs Statistics".

If your operating system supports it, you can also start memcached with the `-L` command line option. This option preallocates all the memory during startup using large memory pages. This can improve performance by reducing the number of misses in the CPU memory cache.

15.6.2.7 memcached Thread Support

If you enable the thread implementation within when building memcached from source, then memcached uses multiple threads in addition to the libevent system to handle requests.

When enabled, the threading implementation operates as follows:

- Threading is handled by wrapping functions within the code to provide basic protection from updating the same global structures at the same time.

- Each thread uses its own instance of the `libevent` to help improve performance.
- TCP/IP connections are handled with a single thread listening on the TCP/IP socket. Each connection is then distributed to one of the active threads on a simple round-robin basis. Each connection then operates solely within this thread while the connection remains open.
- For UDP connections, all the threads listen to a single UDP socket for incoming requests. Threads that are not currently dealing with another request ignore the incoming packet. One of the remaining, nonbusy, threads reads the request and sends the response. This implementation can lead to increased CPU load as threads wake from sleep to potentially process the request.

Using threads can increase the performance on servers that have multiple CPU cores available, as the requests to update the hash table can be spread between the individual threads. To minimize overhead from the locking mechanism employed, experiment with different thread values to achieve the best performance based on the number and type of requests within your given workload.

15.6.2.8 `memcached` Logs

If you enable verbose mode, using the `-v`, `-vv`, or `-vvv` options, then the information output by `memcached` includes details of the operations being performed.

Without the verbose options, `memcached` normally produces no output during normal operating.

- **Output when using `-v`**

  The lowest verbosity level shows you:

  - Errors and warnings
  - Transient errors
  - Protocol and socket errors, including exhausting available connections
  - Each registered client connection, including the socket descriptor number and the protocol used.

    For example:

    ```
    32: Client using the ascii protocol
    33: Client using the ascii protocol
    ```

    Note that the socket descriptor is only valid while the client remains connected. Non-persistent connections may not be effectively represented.

  Examples of the error messages output at this level include:

  ```
  <%d send buffer was %d, now %d
  Can't listen for events on fd %d
  Can't read from libevent pipe
  Catastrophic: event fd doesn't match conn fd!
  Couldn't build response
  Couldn't realloc input buffer
  Couldn't update event
  Failed to build UDP headers
  Failed to read, and not due to blocking
  Too many open connections
  Unexpected state %d
  ```

- **Output when using `-vv`**

When using the second level of verbosity, you get more detailed information about protocol operations, keys updated, chunk and network operatings and details.

During the initial start-up of memcached with this level of verbosity, you are shown the sizes of the individual slab classes, the chunk sizes, and the number of entries per slab. These do not show the allocation of the slabs, just the slabs that would be created when data is added. You are also given information about the listen queues and buffers used to send information. A sample of the output generated for a TCP/IP based system with the default memory and growth factors is given below:

```
shell> memcached -vv
slab class   1: chunk size     80 perslab 13107
slab class   2: chunk size    104 perslab 10082
slab class   3: chunk size    136 perslab  7710
slab class   4: chunk size    176 perslab  5957
slab class   5: chunk size    224 perslab  4681
slab class   6: chunk size    280 perslab  3744
slab class   7: chunk size    352 perslab  2978
slab class   8: chunk size    440 perslab  2383
slab class   9: chunk size    552 perslab  1899
slab class  10: chunk size    696 perslab  1506
slab class  11: chunk size    872 perslab  1202
slab class  12: chunk size   1096 perslab   956
slab class  13: chunk size   1376 perslab   762
slab class  14: chunk size   1720 perslab   609
slab class  15: chunk size   2152 perslab   487
slab class  16: chunk size   2696 perslab   388
slab class  17: chunk size   3376 perslab   310
slab class  18: chunk size   4224 perslab   248
slab class  19: chunk size   5280 perslab   198
slab class  20: chunk size   6600 perslab   158
slab class  21: chunk size   8256 perslab   127
slab class  22: chunk size  10320 perslab   101
slab class  23: chunk size  12904 perslab    81
slab class  24: chunk size  16136 perslab    64
slab class  25: chunk size  20176 perslab    51
slab class  26: chunk size  25224 perslab    41
slab class  27: chunk size  31536 perslab    33
slab class  28: chunk size  39424 perslab    26
slab class  29: chunk size  49280 perslab    21
slab class  30: chunk size  61600 perslab    17
slab class  31: chunk size  77000 perslab    13
slab class  32: chunk size  96256 perslab    10
slab class  33: chunk size 120320 perslab     8
slab class  34: chunk size 150400 perslab     6
slab class  35: chunk size 188000 perslab     5
slab class  36: chunk size 235000 perslab     4
slab class  37: chunk size 293752 perslab     3
slab class  38: chunk size 367192 perslab     2
slab class  39: chunk size 458992 perslab     2
<26 server listening (auto-negotiate)
<29 server listening (auto-negotiate)
<30 send buffer was 57344, now 2097152
<31 send buffer was 57344, now 2097152
<30 server listening (udp)
<30 server listening (udp)
<31 server listening (udp)
<30 server listening (udp)
<30 server listening (udp)
<31 server listening (udp)
<31 server listening (udp)
<31 server listening (udp)
```

Using this verbosity level can be a useful way to check the effects of the growth factor used on slabs with different memory allocations, which in turn can be used to better tune the growth factor to suit the data you are storing in the cache. For example, if you set the growth factor to 4 (quadrupling the size of each slab):

```
shell> memcached -f 4 -m 1g -vv
slab class   1: chunk size     80 perslab 13107
slab class   2: chunk size    320 perslab  3276
slab class   3: chunk size   1280 perslab   819
slab class   4: chunk size   5120 perslab   204
slab class   5: chunk size  20480 perslab    51
slab class   6: chunk size  81920 perslab    12
slab class   7: chunk size 327680 perslab     3
...
```

During use of the cache, this verbosity level also prints out detailed information on the storage and recovery of keys and other information. An example of the output during a typical set/get and increment/decrement operation is shown below.

```
32: Client using the ascii protocol
<32 set my_key 0 0 10
>32 STORED
<32 set object_key 1 0 36
>32 STORED
<32 get my_key
>32 sending key my_key
>32 END
<32 get object_key
>32 sending key object_key
>32 END
<32 set key 0 0 6
>32 STORED
<32 incr key 1
>32 789544
<32 decr key 1
>32 789543
<32 incr key 2
>32 789545
<32 set my_key 0 0 10
>32 STORED
<32 set object_key 1 0 36
>32 STORED
<32 get my_key
>32 sending key my_key
>32 END
<32 get object_key
>32 sending key object_key1 1 36

>32 END
<32 set key 0 0 6
>32 STORED
<32 incr key 1
>32 789544
<32 decr key 1
>32 789543
<32 incr key 2
>32 789545
```

During client communication, for each line, the initial character shows the direction of flow of the information. The < for communication from the client to the memcached server and > for communication back to the client. The number is the numeric socket descriptor for the connection.

- **Output when using -vvv**

  This level of verbosity includes the transitions of connections between different states in the event library while reading and writing content to/from the clients. It should be used to diagnose and identify issues in client communication. For example, you can use this information to determine if memcached is taking a long time to return information to the client, during the read of the client operation or before returning and completing the operation. An example of the typical sequence for a set operation is provided below:

  ```
  <32 new auto-negotiating client connection
  32: going from conn_new_cmd to conn_waiting
  32: going from conn_waiting to conn_read
  32: going from conn_read to conn_parse_cmd
  32: Client using the ascii protocol
  <32 set my_key 0 0 10
  32: going from conn_parse_cmd to conn_nread
  > NOT FOUND my_key
  >32 STORED
  32: going from conn_nread to conn_write
  32: going from conn_write to conn_new_cmd
  32: going from conn_new_cmd to conn_waiting
  32: going from conn_waiting to conn_read
  32: going from conn_read to conn_closing
  <32 connection closed.
  ```

All of the verbosity levels in memcached are designed to be used during debugging or examination of issues. The quantity of information generated, particularly when using -vvv, is significant, particularly on a busy server. Also be aware that writing the error information out, especially to disk, may negate some of the performance gains you achieve by using memcached. Therefore, use in production or deployment environments is not recommended.

15.6.3 Developing a memcached Application

A number of language interfaces let applications store and retrieve information with memcached servers. You can write memcached applications in popular languages such as Perl, PHP, Python, Ruby, C, and Java.

Data stored into a memcached server is referred to by a single string (the key), with storage into the cache and retrieval from the cache using the key as the reference. The cache therefore operates like a large associative array or hash table. It is not possible to structure or otherwise organize the information stored in the cache. To emulate database notions such as multiple tables or composite key values, you must encode the extra information into the strings used as keys. For example, to store or look up the address corresponding to a specific latitude and longitude, you might turn those two numeric values into a single comma-separated string to use as a key.

15.6.3.1 Basic memcached Operations

The interface to memcached supports the following methods for storing and retrieving information in the cache, and these are consistent across all the different APIs, although the language specific mechanics might be different:

- get(*key*): Retrieves information from the cache. Returns the value associated with the key if the specified key exists. Returns NULL, nil, undefined, or the closest equivalent in the corresponding language, if the specified key does not exist.

- set(*key*, *value* [, *expiry*]): Sets the item associated with a key in the cache to the specified value. Note that this either updates an existing item if the key already exists, or adds a new key/value pair if the key doesn't exist. If the expiry time is specified, then the item expires (and is deleted) when the expiry time is reached. The time is specified in seconds, and is taken as a relative time if the value is less than 30 days (30*24*60*60), or an absolute time (epoch) if larger than this value.

- `add(key, value [, expiry])`: Adds the key and associated value to the cache, if the specified key does not already exist.
- `replace(key, value [, expiry])`: Replaces the item associated with the specified `key`, only if the key already exists. The new value is given by the `value` parameter.
- `delete(key [, time])`: Deletes the `key` and its associated item from the cache. If you supply a `time`, then adding another item with the specified `key` is blocked for the specified period.
- `incr(key [, value])`: Increments the item associated with the `key` by one or the optional `value`.
- `decr(key [, value])`: Decrements the item associated with the `key` by one or the optional `value`.
- `flush_all`: Invalidates (or expires) all the current items in the cache. Technically they still exist (they are not deleted), but they are silently destroyed the next time you try to access them.

In all implementations, most or all of these functions are duplicated through the corresponding native language interface.

When practical, use `memcached` to store full items, rather than caching a single column value from the database. For example, when displaying a record about an object (invoice, user history, or blog post), load all the data for the associated entry from the database, and compile it into the internal structure that would normally be required by the application. Save the complete object in the cache.

Complex data structures cannot be stored directly. Most interfaces serialize the data for you, that is, put it in a textual form that can reconstruct the original pointers and nesting. Perl uses `Storable`, PHP uses `serialize`, Python uses `cPickle` (or `Pickle`) and Java uses the `Serializable` interface. In most cases, the serialization interface used is customizable. To share data stored in `memcached` instances between different language interfaces, consider using a common serialization solution such as JSON (Javascript Object Notation).

15.6.3.2 Using memcached as a MySQL Caching Layer

When using `memcached` to cache MySQL data, your application must retrieve data from the database and load the appropriate key-value pairs into the cache. Then, subsequent lookups can be done directly from the cache.

Because MySQL has its own in-memory caching mechanisms for queried data, such as the `InnoDB` buffer pool and the MySQL query cache, look for opportunities beyond loading individual column values or rows into the cache. Prefer to cache composite values, such as those retrieved from multiple tables through a join query, or result sets assembled from multiple rows.

Caution

Limit the information in the cache to non-sensitive data, because there is no security required to access or update the information within a `memcached` instance. Anybody with access to the machine has the ability to read, view and potentially update the information. To keep the data secure, encrypt the information before caching it. To restrict the users capable of connecting to the server, either disable network access, or use IPTables or similar techniques to restrict access to the `memcached` ports to a select set of hosts.

You can introduce `memcached` to an existing application, even if caching was not part of the original design. In many languages and environments the changes to the application will be just a few lines, first to attempt to read from the cache when loading data, fall back to the old method if the information is not cached, and to update the cache with information once the data has been read.

The general sequence for using memcached in any language as a caching solution for MySQL is as follows:

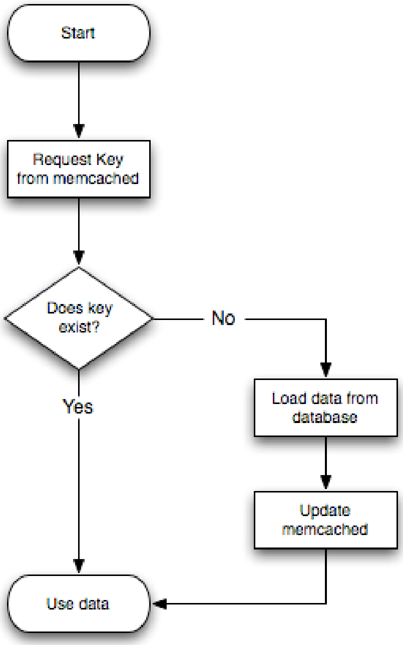
1. Request the item from the cache.

2. If the item exists, use the item data.

3. If the item does not exist, load the data from MySQL, and store the value into the cache. This means the value is available to the next client that requests it from the cache.

For a flow diagram of this sequence, see Figure 15.8, “Typical memcached Application Flowchart”.

Figure 15.8 Typical memcached Application Flowchart



Adapting Database Best Practices to memcached Applications

The most direct way to cache MySQL data is to use a 2-column table, where the first column is a primary key. Because of the uniqueness requirements for memcached keys, make sure your database schema makes appropriate use of primary keys and unique constraints.

If you combine multiple column values into a single memcached item value, choose data types to make it easy to parse the value back into its components, for example by using a separator character between numeric values.

The queries that map most easily to memcached lookups are those with a single WHERE clause, using an = or IN operator. For complicated WHERE clauses, or those using operators such as <, >, BETWEEN, or LIKE, memcached does not provide a simple or efficient way to scan through or filter the keys or associated values, so typically you perform those operations as SQL queries on the underlying database.

15.6.3.3 Using libmemcached with C and C++

The libmemcached library provides both C and C++ interfaces to memcached and is also the basis for a number of different additional API implementations, including Perl, Python and Ruby. Understanding the core libmemcached functions can help when using these other interfaces.

The C library is the most comprehensive interface library for memcached and provides functions and operational systems not always exposed in interfaces not based on the libmemcached library.

The different functions can be divided up according to their basic operation. In addition to functions that interface to the core API, a number of utility functions provide extended functionality, such as appending and prepending data.

To build and install libmemcached, download the libmemcached package, run configure, and then build and install:

```
shell> tar xjf libmemcached-0.21.tar.gz
shell> cd libmemcached-0.21
shell> ./configure
shell> make
shell> make install
```

On many Linux operating systems, you can install the corresponding libmemcached package through the usual yum, apt-get, or similar commands.

To build an application that uses the library, first set the list of servers. Either directly manipulate the servers configured within the main memcached_st structure, or separately populate a list of servers, and then add this list to the memcached_st structure. The latter method is used in the following example. Once the server list has been set, you can call the functions to store or retrieve data. A simple application for setting a preset value to localhost is provided here:

```
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <libmemcached/memcached.h>

int main(int argc, char *argv[])
{
  memcached_server_st *servers = NULL;
  memcached_st *memc;
  memcached_return rc;
  char *key= "keystring";
  char *value= "keyvalue";

  memcached_server_st *memcached_servers_parse (char *server_strings);
  memc= memcached_create(NULL);

  servers= memcached_server_list_append(servers, "localhost", 11211, &rc);
  rc= memcached_server_push(memc, servers);

  if (rc == MEMCACHED_SUCCESS)
    fprintf(stderr,"Added server successfully\n");
  else
    fprintf(stderr,"Couldn't add server: %s\n",memcached_strerror(memc, rc));

  rc= memcached_set(memc, key, strlen(key), value, strlen(value), (time_t)0, (uint32_t)0);

  if (rc == MEMCACHED_SUCCESS)
    fprintf(stderr,"Key stored successfully\n");
  else
```

```
    fprintf(stderr,"Couldn't store key: %s\n",memcached_strerror(memc, rc));

  return 0;
}
```

To test the success of an operation, use the return value, or populated result code, for a given function. The value is always set to MEMCACHED_SUCCESS if the operation succeeded. In the event of a failure, use the memcached_strerror() function to translate the result code into a printable string.

To build the application, specify the memcached library:

```
shell> gcc -o memc_basic memc_basic.c -lmemcached
```

Running the above sample application, after starting a memcached server, should return a success message:

```
shell> memc_basic
Added server successfully
Key stored successfully
```

libmemcached Base Functions

The base libmemcached functions let you create, destroy and clone the main memcached_st structure that is used to interface with the memcached servers. The main functions are defined below:

```
memcached_st *memcached_create (memcached_st *ptr);
```

Creates a new memcached_st structure for use with the other libmemcached API functions. You can supply an existing, static, memcached_st structure, or NULL to have a new structured allocated. Returns a pointer to the created structure, or NULL on failure.

```
void memcached_free (memcached_st *ptr);
```

Frees the structure and memory allocated to a previously created memcached_st structure.

```
memcached_st *memcached_clone(memcached_st *clone, memcached_st *source);
```

Clones an existing memcached structure from the specified source, copying the defaults and list of servers defined in the structure.

libmemcached Server Functions

The libmemcached API uses a list of servers, stored within the memcached_server_st structure, to act as the list of servers used by the rest of the functions. To use memcached, you first create the server list, and then apply the list of servers to a valid libmemcached object.

Because the list of servers, and the list of servers within an active libmemcached object can be manipulated separately, you can update and manage server lists while an active libmemcached interface is running.

The functions for manipulating the list of servers within a memcached_st structure are:

```
memcached_return
   memcached_server_add (memcached_st *ptr,
                         char *hostname,
                         unsigned int port);
```

Adds a server, using the given hostname and port into the memcached_st structure given in ptr.

```
memcached_return
   memcached_server_add_unix_socket (memcached_st *ptr,
                                     char *socket);
```

Adds a Unix socket to the list of servers configured in the memcached_st structure.

```
unsigned int memcached_server_count (memcached_st *ptr);
```

Returns a count of the number of configured servers within the memcached_st structure.

```
memcached_server_st *
   memcached_server_list (memcached_st *ptr);
```

Returns an array of all the defined hosts within a memcached_st structure.

```
memcached_return
   memcached_server_push (memcached_st *ptr,
                          memcached_server_st *list);
```

Pushes an existing list of servers onto list of servers configured for a current memcached_st structure. This adds servers to the end of the existing list, and duplicates are not checked.

The memcached_server_st structure can be used to create a list of memcached servers which can then be applied individually to memcached_st structures.

```
memcached_server_st *
   memcached_server_list_append (memcached_server_st *ptr,
                                 char *hostname,
                                 unsigned int port,
                                 memcached_return *error);
```

Adds a server, with hostname and port, to the server list in ptr. The result code is handled by the error argument, which should point to an existing memcached_return variable. The function returns a pointer to the returned list.

```
unsigned int memcached_server_list_count (memcached_server_st *ptr);
```

Returns the number of the servers in the server list.

```
void memcached_server_list_free (memcached_server_st *ptr);
```

Frees the memory associated with a server list.

```
memcached_server_st *memcached_servers_parse (char *server_strings);
```

Parses a string containing a list of servers, where individual servers are separated by a comma, space, or both, and where individual servers are of the form *server*[:*port*]. The return value is a server list structure.

libmemcached Set Functions

The set-related functions within libmemcached provide the same functionality as the core functions supported by the memcached protocol. The full definition for the different functions is the same for all the base functions (add, replace, prepend, append). For example, the function definition for memcached_set() is:

```
memcached_return
   memcached_set (memcached_st *ptr,
                  const char *key,
                  size_t key_length,
                  const char *value,
                  size_t value_length,
                  time_t expiration,
                  uint32_t flags);
```

The ptr is the memcached_st structure. The key and key_length define the key name and length, and value and value_length the corresponding value and length. You can also set the expiration and optional flags. For more information, see Controlling libmemcached Behaviors.

This table outlines the remainder of the set-related libmemcached functions and the equivalent core functions supported by the memcached protocol.

libmemcached Function	Equivalent Core Function
memcached_set(memc, key, key_length, value, value_length, expiration, flags)	Generic set() operation.
memcached_add(memc, key, key_length, value, value_length, expiration, flags)	Generic add() function.
memcached_replace(memc, key, key_length, value, value_length, expiration, flags)	Generic replace().
memcached_prepend(memc, key, key_length, value, value_length, expiration, flags)	Prepends the specified value before the current value of the specified key.
memcached_append(memc, key, key_length, value, value_length, expiration, flags)	Appends the specified value after the current value of the specified key.
memcached_cas(memc, key, key_length, value, value_length, expiration, flags, cas)	Overwrites the data for a given key as long as the corresponding cas value is still the same within the server.
memcached_set_by_key(memc, master_key, master_key_length, key, key_length, value, value_length, expiration, flags)	Similar to the generic set(), but has the option of an additional master key that can be used to identify an individual server.
memcached_add_by_key(memc, master_key, master_key_length, key, key_length, value, value_length, expiration, flags)	Similar to the generic add(), but has the option of an additional master key that can be used to identify an individual server.
memcached_replace_by_key(memc, master_key, master_key_length, key,	Similar to the generic replace(), but has the option of an additional master key that can be used to identify an individual server.

libmemcached Function	Equivalent Core Function
key_length, value, value_length, expiration, flags)	
memcached_prepend_by_key(memc, master_key, master_key_length, key, key_length, value, value_length, expiration, flags)	Similar to the memcached_prepend(), but has the option of an additional master key that can be used to identify an individual server.
memcached_append_by_key(memc, master_key, master_key_length, key, key_length, value, value_length, expiration, flags)	Similar to the memcached_append(), but has the option of an additional master key that can be used to identify an individual server.
memcached_cas_by_key(memc, master_key, master_key_length, key, key_length, value, value_length, expiration, flags)	Similar to the memcached_cas(), but has the option of an additional master key that can be used to identify an individual server.

The by_key methods add two further arguments that define the master key, to be used and applied during the hashing stage for selecting the servers. You can see this in the following definition:

```
memcached_return
   memcached_set_by_key(memcached_st *ptr,
                        const char *master_key,
                        size_t master_key_length,
                        const char *key,
                        size_t key_length,
                        const char *value,
                        size_t value_length,
                        time_t expiration,
                        uint32_t flags);
```

All the functions return a value of type memcached_return, which you can compare against the MEMCACHED_SUCCESS constant.

libmemcached Get Functions

The libmemcached functions provide both direct access to a single item, and a multiple-key request mechanism that provides much faster responses when fetching a large number of keys simultaneously.

The main get-style function, which is equivalent to the generic get() is memcached_get(). This function returns a string pointer, pointing to the value associated with the specified key.

```
char *memcached_get (memcached_st *ptr,
                     const char *key, size_t key_length,
                     size_t *value_length,
                     uint32_t *flags,
                     memcached_return *error);
```

A multi-key get, memcached_mget(), is also available. Using a multiple key get operation is much quicker to do in one block than retrieving the key values with individual calls to memcached_get(). To start the multi-key get, call memcached_mget():

```
memcached_return
    memcached_mget (memcached_st *ptr,
                    char **keys, size_t *key_length,
                    unsigned int number_of_keys);
```

The return value is the success of the operation. The keys parameter should be an array of strings containing the keys, and key_length an array containing the length of each corresponding key. number_of_keys is the number of keys supplied in the array.

To fetch the individual values, use memcached_fetch() to get each corresponding value.

```
char *memcached_fetch (memcached_st *ptr,
                       const char *key, size_t *key_length,
                       size_t *value_length,
                       uint32_t *flags,
                       memcached_return *error);
```

The function returns the key value, with the key, key_length and value_length parameters being populated with the corresponding key and length information. The function returns NULL when there are no more values to be returned. A full example, including the populating of the key data and the return of the information is provided here.

```
#include <stdio.h>
#include <sstring.h>
#include <unistd.h>
#include <libmemcached/memcached.h>

int main(int argc, char *argv[])
{
  memcached_server_st *servers = NULL;
  memcached_st *memc;
  memcached_return rc;
  char *keys[]= {"huey", "dewey", "louie"};
  size_t key_length[3];
  char *values[]= {"red", "blue", "green"};
  size_t value_length[3];
  unsigned int x;
  uint32_t flags;

  char return_key[MEMCACHED_MAX_KEY];
  size_t return_key_length;
  char *return_value;
  size_t return_value_length;

  memc= memcached_create(NULL);

  servers= memcached_server_list_append(servers, "localhost", 11211, &rc);
  rc= memcached_server_push(memc, servers);

  if (rc == MEMCACHED_SUCCESS)
    fprintf(stderr,"Added server successfully\n");
  else
    fprintf(stderr,"Couldn't add server: %s\n",memcached_strerror(memc, rc));

  for(x= 0; x < 3; x++)
    {
      key_length[x] = strlen(keys[x]);
      value_length[x] = strlen(values[x]);

      rc= memcached_set(memc, keys[x], key_length[x], values[x],
                        value_length[x], (time_t)0, (uint32_t)0);
      if (rc == MEMCACHED_SUCCESS)
        fprintf(stderr,"Key %s stored successfully\n",keys[x]);
      else
        fprintf(stderr,"Couldn't store key: %s\n",memcached_strerror(memc, rc));
    }

  rc= memcached_mget(memc, keys, key_length, 3);
  if (rc == MEMCACHED_SUCCESS)
    {
      while ((return_value= memcached_fetch(memc, return_key, &return_key_length,
                                            &return_value_length, &flags, &rc)) != NULL)
        {
          if (rc == MEMCACHED_SUCCESS)
            {
              fprintf(stderr,"Key %s returned %s\n",return_key, return_value);
            }
        }
    }

  return 0;
}
```

Running the above application produces the following output:

```
shell> memc_multi_fetch
Added server successfully
Key huey stored successfully
Key dewey stored successfully
Key louie stored successfully
Key huey returned red
Key dewey returned blue
Key louie returned green
```

Controlling libmemcached Behaviors

The behavior of libmemcached can be modified by setting one or more behavior flags. These can either be set globally, or they can be applied during the call to individual functions. Some behaviors also accept an additional setting, such as the hashing mechanism used when selecting servers.

To set global behaviors:

```
memcached_return
   memcached_behavior_set (memcached_st *ptr,
                           memcached_behavior flag,
                           uint64_t data);
```

To get the current behavior setting:

```
uint64_t
   memcached_behavior_get (memcached_st *ptr,
                           memcached_behavior flag);
```

The following table describes libmemcached behavior flags.

Behavior	Description
MEMCACHED_BEHAVIOR_NO_BLOCK	Caused libmemcached to use asynchronous I/O.
MEMCACHED_BEHAVIOR_TCP_NODELAY	Turns on no-delay for network sockets.
MEMCACHED_BEHAVIOR_HASH	Without a value, sets the default hashing algorithm for keys to use MD5. Other valid values include MEMCACHED_HASH_DEFAULT, MEMCACHED_HASH_MD5, MEMCACHED_HASH_CRC, MEMCACHED_HASH_FNV1_64, MEMCACHED_HASH_FNV1A_64, MEMCACHED_HASH_FNV1_32, and MEMCACHED_HASH_FNV1A_32.
MEMCACHED_BEHAVIOR_DISTRIBUTION	Changes the method of selecting the server used to store a given value. The default method

Behavior	Description
	is MEMCACHED_DISTRIBUTION_MODULA. You can enable consistent hashing by setting MEMCACHED_DISTRIBUTION_CONSISTENT. MEMCACHED_DISTRIBUTION_CONSISTENT is an alias for the value MEMCACHED_DISTRIBUTION_CONSISTENT_KETAMA.
MEMCACHED_BEHAVIOR_CACHE_LOOKUPS	Cache the lookups made to the DNS service. This can improve the performance if you are using names instead of IP addresses for individual hosts.
MEMCACHED_BEHAVIOR_SUPPORT_CAS	Support CAS operations. By default, this is disabled because it imposes a performance penalty.
MEMCACHED_BEHAVIOR_KETAMA	Sets the default distribution to MEMCACHED_DISTRIBUTION_CONSISTENT_KETAMA and the hash to MEMCACHED_HASH_MD5.
MEMCACHED_BEHAVIOR_POLL_TIMEOUT	Modify the timeout value used by poll(). Supply a signed int pointer for the timeout value.
MEMCACHED_BEHAVIOR_BUFFER_REQUESTS	Buffers IO requests instead of them being sent. A get operation, or closing the connection causes the data to be flushed.
MEMCACHED_BEHAVIOR_VERIFY_KEY	Forces libmemcached to verify that a specified key is valid.
MEMCACHED_BEHAVIOR_SORT_HOSTS	If set, hosts added to the list of configured hosts for a memcached_st structure are placed into the host list in sorted order. This breaks consistent hashing if that behavior has been enabled.
MEMCACHED_BEHAVIOR_CONNECT_TIMEOUT	In nonblocking mode this changes the value of the timeout during socket connection.

libmemcached Command-Line Utilities

In addition to the main C library interface, libmemcached also includes a number of command-line utilities that can be useful when working with and debugging memcached applications.

All of the command-line tools accept a number of arguments, the most critical of which is servers, which specifies the list of servers to connect to when returning information.

The main tools are:

- memcat: Display the value for each ID given on the command line:

  ```
  shell> memcat --servers=localhost hwkey
  Hello world
  ```

- memcp: Copy the contents of a file into the cache, using the file name as the key:

  ```
  shell> echo "Hello World" > hwkey
  shell> memcp --servers=localhost hwkey
  shell> memcat --servers=localhost hwkey
  Hello world
  ```

- memrm: Remove an item from the cache:

```
shell> memcat --servers=localhost hwkey
Hello world
shell> memrm --servers=localhost hwkey
shell> memcat --servers=localhost hwkey
```

- memslap: Test the load on one or more memcached servers, simulating get/set and multiple client operations. For example, you can simulate the load of 100 clients performing get operations:

```
shell> memslap --servers=localhost --concurrency=100 --flush --test=get
memslap --servers=localhost --concurrency=100 --flush --test=get Threads connecting to servers 100
 Took 13.571 seconds to read data
```

- memflush: Flush (empty) the contents of the memcached cache.

```
shell> memflush --servers=localhost
```

15.6.3.4 Using MySQL and memcached with Perl

The Cache::Memcached module provides a native interface to the Memcache protocol, and provides support for the core functions offered by memcached. Install the module using your operating system's package management system, or using CPAN:

```
root-shell> perl -MCPAN -e 'install Cache::Memcached'
```

To use memcached from Perl through the Cache::Memcached module, first create a new Cache::Memcached object that defines the list of servers and other parameters for the connection. The only argument is a hash containing the options for the cache interface. For example, to create a new instance that uses three memcached servers:

```
use Cache::Memcached;

my $cache = new Cache::Memcached {
    'servers' => [
        '192.168.0.100:11211',
        '192.168.0.101:11211',
        '192.168.0.102:11211',
 ],
};
```

Note

When using the Cache::Memcached interface with multiple servers, the API automatically performs certain operations across all the servers in the group. For example, getting statistical information through Cache::Memcached returns a hash that contains data on a host-by-host basis, as well as generalized statistics for all the servers in the group.

You can set additional properties on the cache object instance when it is created by specifying the option as part of the option hash. Alternatively, you can use a corresponding method on the instance:

- servers or method set_servers(): Specifies the list of the servers to be used. The servers list should be a reference to an array of servers, with each element as the address and port number combination (separated by a colon). You can also specify a local connection through a Unix socket (for example /tmp/sock/memcached). To specify the server with a weight (indicating how much more frequently the server should be used during hashing), specify an array reference with the memcached server instance and a weight number. Higher numbers give higher priority.

- compress_threshold or method set_compress_threshold(): Specifies the threshold when values are compressed. Values larger than the specified number are automatically compressed (using zlib) during storage and retrieval.
- no_rehash or method set_norehash(): Disables finding a new server if the original choice is unavailable.
- readonly or method set_readonly(): Disables writes to the memcached servers.

Once the Cache::Memcached object instance has been configured, you can use the set() and get() methods to store and retrieve information from the memcached servers. Objects stored in the cache are automatically serialized and deserialized using the Storable module.

The Cache::Memcached interface supports the following methods for storing/retrieving data, and relate to the generic methods as shown in the table.

Cache::Memcached Function	**Equivalent Generic Method**
get()	Generic get().
get_multi(keys)	Gets multiple keys from memcache using just one query. Returns a hash reference of key/value pairs.
set()	Generic set().
add()	Generic add().
replace()	Generic replace().
delete()	Generic delete().
incr()	Generic incr().
decr()	Generic decr().

Below is a complete example for using memcached with Perl and the Cache::Memcached module:

```
#!/usr/bin/perl

use Cache::Memcached;
use DBI;
use Data::Dumper;

# Configure the memcached server

my $cache = new Cache::Memcached {
    'servers' => [
                   'localhost:11211',
                   ],
    };

# Get the film name from the command line
# memcached keys must not contain spaces, so create
# a key name by replacing spaces with underscores

my $filmname = shift or die "Must specify the film name\n";
my $filmkey = $filmname;
$filmkey =~ s/ /_/;

# Load the data from the cache

my $filmdata = $cache->get($filmkey);

# If the data wasn't in the cache, then we load it from the database
```

```
if (!defined($filmdata))
{
    $filmdata = load_filmdata($filmname);

    if (defined($filmdata))
    {

# Set the data into the cache, using the key

 if ($cache->set($filmkey,$filmdata))
        {
            print STDERR "Film data loaded from database and cached\n";
        }
        else
        {
            print STDERR "Couldn't store to cache\n";
 }
    }
    else
    {
      die "Couldn't find $filmname\n";
    }
}
else
{
    print STDERR "Film data loaded from Memcached\n";
}

sub load_filmdata
{
    my ($filmname) = @_;

    my $dsn = "DBI:mysql:database=sakila;host=localhost;port=3306";

    $dbh = DBI->connect($dsn, 'sakila','password');

    my ($filmbase) = $dbh->selectrow_hashref(sprintf('select * from film where title = %s',
                                                     $dbh->quote($filmname)));

    if (!defined($filmname))
    {
      return (undef);
    }

    $filmbase->{stars} =
 $dbh->selectall_arrayref(sprintf('select concat(first_name," ",last_name) ' .
                                         'from film_actor left join (actor) ' .
                                         'on (film_actor.actor_id = actor.actor_id) ' .
                                         ' where film_id=%s',
                                         $dbh->quote($filmbase->{film_id})));

    return($filmbase);
}
```

The example uses the Sakila database, obtaining film data from the database and writing a composite record of the film and actors to memcached. When calling it for a film does not exist, you get this result:

```
shell> memcached-sakila.pl "ROCK INSTINCT"
Film data loaded from database and cached
```

When accessing a film that has already been added to the cache:

```
shell> memcached-sakila.pl "ROCK INSTINCT"
Film data loaded from Memcached
```

15.6.3.5 Using MySQL and memcached with Python

The Python memcache module interfaces to memcached servers, and is written in pure Python (that is, without using one of the C APIs). You can download and install a copy from Python Memcached.

To install, download the package and then run the Python installer:

```
python setup.py install
running install
running bdist_egg
running egg_info
creating python_memcached.egg-info
...
removing 'build/bdist.linux-x86_64/egg' (and everything under it)
Processing python_memcached-1.43-py2.4.egg
creating /usr/lib64/python2.4/site-packages/python_memcached-1.43-py2.4.egg
Extracting python_memcached-1.43-py2.4.egg to /usr/lib64/python2.4/site-packages
Adding python-memcached 1.43 to easy-install.pth file

Installed /usr/lib64/python2.4/site-packages/python_memcached-1.43-py2.4.egg
Processing dependencies for python-memcached==1.43
Finished processing dependencies for python-memcached==1.43
```

Once installed, the memcache module provides a class-based interface to your memcached servers. When you store Python data structures as memcached items, they are automatically serialized (turned into string values) using the Python cPickle or pickle modules.

To create a new memcache interface, import the memcache module and create a new instance of the memcache.Client class. For example, if the memcached daemon is running on localhost using the default port:

```
import memcache
memc = memcache.Client(['127.0.0.1:11211'])
```

The first argument is an array of strings containing the server and port number for each memcached instance to use. To enable debugging, set the optional debug parameter to 1.

By default, the hashing mechanism used to divide the items among multiple servers is crc32. To change the function used, set the value of memcache.serverHashFunction to the alternate function to use. For example:

```
from zlib import adler32
memcache.serverHashFunction = adler32
```

Once you have defined the servers to use within the memcache instance, the core functions provide the same functionality as in the generic interface specification. The following table provides a summary of the supported functions:

Python memcache Function	Equivalent Generic Function
get()	Generic get().
get_multi(keys)	Gets multiple values from the supplied array of keys. Returns a hash reference of key/value pairs.
set()	Generic set().
set_multi(dict [, expiry [, key_prefix]])	Sets multiple key/value pairs from the supplied dict.
add()	Generic add().

Python memcache Function	Equivalent Generic Function
replace()	Generic replace().
prepend(key, value [, expiry])	Prepends the supplied value to the value of the existing key.
append(key, value [, expiry[)	Appends the supplied value to the value of the existing key.
delete()	Generic delete().
delete_multi(keys [, expiry [, key_prefix]])	Deletes all the keys from the hash matching each string in the array keys.
incr()	Generic incr().
decr()	Generic decr().

Note

Within the Python memcache module, all the *_multi()functions support an optional key_prefix parameter. If supplied, then the string is used as a prefix to all key lookups. For example, if you call:

```
memc.get_multi(['a','b'], key_prefix='users:')
```

The function retrieves the keys users:a and users:b from the servers.

Here is an example showing the storage and retrieval of information to a memcache instance, loading the raw data from MySQL:

```
import sys
import MySQLdb
import memcache

memc = memcache.Client(['127.0.0.1:11211'], debug=1);

try:
    conn = MySQLdb.connect (host = "localhost",
                            user = "sakila",
                            passwd = "password",
                            db = "sakila")
except MySQLdb.Error, e:
     print "Error %d: %s" % (e.args[0], e.args[1])
     sys.exit (1)

popularfilms = memc.get('top5films')

if not popularfilms:
    cursor = conn.cursor()
    cursor.execute('select film_id,title from film order by rental_rate desc limit 5')
    rows = cursor.fetchall()
    memc.set('top5films',rows,60)
    print "Updated memcached with MySQL data"
else:
    print "Loaded data from memcached"
    for row in popularfilms:
        print "%s, %s" % (row[0], row[1])
```

When executed for the first time, the data is loaded from the MySQL database and stored to the memcached server.

```
shell> python memc_python.py
Updated memcached with MySQL data
```

Because the data is automatically serialized using cPickle/pickle, when you load the data back from memcached, you can use the object directly. In the example above, the information stored to memcached is in the form of rows from a Python DB cursor. When accessing the information (within the 60 second expiry time), the data is loaded from memcached and dumped:

```

shell> python memc_python.py
Loaded data from memcached
2, ACE GOLDFINGER
7, AIRPLANE SIERRA
8, AIRPORT POLLOCK
10, ALADDIN CALENDAR
13, ALI FOREVER
```

The serialization and deserialization happens automatically. Because serialization of Python data may be incompatible with other interfaces and languages, you can change the serialization module used during initialization. For example, you might use JSON format when you store complex data structures using a script written in one language, and access them in a script written in a different language.

15.6.3.6 Using MySQL and memcached with PHP

PHP provides support for the Memcache functions through a PECL extension. To enable the PHP memcache extensions, build PHP using the --enable-memcache option to configure when building from source.

If you are installing on a Red Hat-based server, you can install the php-pecl-memcache RPM:

```

root-shell> yum --install php-pecl-memcache
```

On Debian-based distributions, use the php-memcache package.

To set global runtime configuration options, specify the configuration option values within your php.ini file. The following table provides the name, default value, and a description for each global runtime configuration option.

Configuration option	Default	Description
memcache.allow_failover	1	Specifies whether another server in the list should be queried if the first server selected fails.
memcache.max_failover_attempts	20	Specifies the number of servers to try before returning a failure.
memcache.chunk_size	8192	Defines the size of network chunks used to exchange data with the memcached server.
memcache.default_port	11211	Defines the default port to use when communicating with the memcached servers.
memcache.hash_strategy	standard	Specifies which hash strategy to use. Set to consistent to enable servers to be added or removed from the pool without causing the keys to be remapped to other servers. When set to standard, an older (modula) strategy is used that potentially uses different servers for storage.

Configuration option	Default	Description
memcache.hash_function	crc32	Specifies which function to use when mapping keys to servers. crc32 uses the standard CRC32 hash. fnv uses the FNV-1a hashing algorithm.

To create a connection to a memcached server, create a new Memcache object and then specify the connection options. For example:

```
<?php

$cache = new Memcache;
$cache->connect('localhost',11211);
?>
```

This opens an immediate connection to the specified server.

To use multiple memcached servers, you need to add servers to the memcache object using addServer():

```
bool Memcache::addServer ( string $host [, int $port [, bool $persistent
                 [, int $weight [, int $timeout [, int $retry_interval
                 [, bool $status [, callback $failure_callback
                 ]]]]]]] )
```

The server management mechanism within the php-memcache module is a critical part of the interface as it controls the main interface to the memcached instances and how the different instances are selected through the hashing mechanism.

To create a simple connection to two memcached instances:

```
<?php

$cache = new Memcache;
$cache->addServer('192.168.0.100',11211);
$cache->addServer('192.168.0.101',11211);
?>
```

In this scenario, the instance connection is not explicitly opened, but only opened when you try to store or retrieve a value. To enable persistent connections to memcached instances, set the $persistent argument to true. This is the default setting, and causes the connections to remain open.

To help control the distribution of keys to different instances, use the global memcache.hash_strategy setting. This sets the hashing mechanism used to select. You can also add another weight to each server, which effectively increases the number of times the instance entry appears in the instance list, therefore increasing the likelihood of the instance being chosen over other instances. To set the weight, set the value of the $weight argument to more than one.

The functions for setting and retrieving information are identical to the generic functional interface offered by memcached, as shown in this table:

PECL memcache Function	Generic Function
get()	Generic get().
set()	Generic set().
add()	Generic add().
replace()	Generic replace().

PECL memcache Function	Generic Function
delete()	Generic delete().
increment()	Generic incr().
decrement()	Generic decr().

A full example of the PECL memcache interface is provided below. The code loads film data from the Sakila database when the user provides a film name. The data stored into the memcached instance is recorded as a mysqli result row, and the API automatically serializes the information for you.

```
<?php

$memc = new Memcache;
$memc->addServer('localhost','11211');

if(empty($_POST['film'])) {
?>
  <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
    <head>
      <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
      <title>Simple Memcache Lookup</title>
    </head>
    <body>
      <form method="post">
        <p><b>Film</b>: <input type="text" size="20" name="film"></p>
        <input type="submit">
      </form>
      <hr/>
<?php

} else {

    echo "Loading data...\n";

    $film   = htmlspecialchars($_POST['film'], ENT_QUOTES, 'UTF-8');
    $mfilms = $memc->get($film);

    if ($mfilms) {

        printf("<p>Film data for %s loaded from memcache</p>", $mfilms['title']);

        foreach (array_keys($mfilms) as $key) {
            printf("<p><b>%s</b>: %s</p>", $key, $mfilms[$key]);
        }

    } else {

        $mysqli = mysqli('localhost','sakila','password','sakila');

        if (mysqli_connect_error()) {
            sprintf("Database error: (%d) %s", mysqli_connect_errno(), mysqli_connect_error());
            exit;
        }

        $sql = sprintf('SELECT * FROM film WHERE title="%s"', $mysqli->real_escape_string($film));

        $result = $mysqli->query($sql);

        if (!$result) {
            sprintf("Database error: (%d) %s", $mysqli->errno, $mysqli->error);
            exit;
        }
        $row = $result->fetch_assoc();

        $memc->set($row['title'], $row);

        printf("<p>Loaded (%s) from MySQL</p>", htmlspecialchars($row['title'], ENT_QUOTES, 'UTF-8'));
    }
}
?>
  </body>
</html>
```

With PHP, the connections to the memcached instances are kept open as long as the PHP and associated Apache instance remain running. When adding or removing servers from the list in a running instance (for example, when starting another script that mentions additional servers), the connections are shared, but the script only selects among the instances explicitly configured within the script.

To ensure that changes to the server list within a script do not cause problems, make sure to use the consistent hashing mechanism.

15.6.3.7 Using MySQL and memcached with Ruby

There are a number of different modules for interfacing to memcached within Ruby. The Ruby-MemCache client library provides a native interface to memcached that does not require any external libraries, such as libmemcached. You can obtain the installer package from http://www.deveiate.org/projects/RMemCache.

To install, extract the package and then run install.rb:

```
shell> install.rb
```

If you have RubyGems, you can install the Ruby-MemCache gem:

```
shell> gem install Ruby-MemCache
Bulk updating Gem source index for: http://gems.rubyforge.org
Install required dependency io-reactor? [Yn]  y
Successfully installed Ruby-MemCache-0.0.1
Successfully installed io-reactor-0.05
Installing ri documentation for io-reactor-0.05...
Installing RDoc documentation for io-reactor-0.05...
```

To use a memcached instance from within Ruby, create a new instance of the MemCache object.

```
require 'memcache'
memc = MemCache::new '192.168.0.100:11211'
```

You can add a weight to each server to increase the likelihood of the server being selected during hashing by appending the weight count to the server host name/port string:

```
require 'memcache'
memc = MemCache::new '192.168.0.100:11211:3'
```

To add servers to an existing list, you can append them directly to the MemCache object:

```
memc += ["192.168.0.101:11211"]
```

To set data into the cache, you can just assign a value to a key within the new cache object, which works just like a standard Ruby hash object:

```
memc["key"] = "value"
```

Or to retrieve the value:

```
print memc["key"]
```

For more explicit actions, you can use the method interface, which mimics the main memcached API functions, as summarized in the following table:

Ruby MemCache Method	Equivalent memcached API Functions
get()	Generic get().
get_hash(keys)	Get the values of multiple keys, returning the information as a hash of the keys and their values.
set()	Generic set().
set_many(pairs)	Set the values of the keys and values in the hash pairs.
add()	Generic add().
replace()	Generic replace().
delete()	Generic delete().
incr()	Generic incr().
decr()	Generic decr().

15.6.3.8 Using MySQL and memcached with Java

The com.danga.MemCached class within Java provides a native interface to memcached instances. You can obtain the client from https://github.com/gwhalin/Memcached-Java-Client/downloads. The Java class uses hashes that are compatible with libmemcached, so you can mix and match Java and libmemcached applications accessing the same memcached instances. The serialization between Java and other interfaces are not compatible. If this is a problem, use JSON or a similar nonbinary serialization format.

On most systems, you can download the package and use the jar directly.

To use the com.danga.MemCached interface, you create a MemCachedClient instance and then configure the list of servers by configuring the SockIOPool. Through the pool specification you set up the server list, weighting, and the connection parameters to optimized the connections between your client and the memcached instances that you configure.

Generally, you can configure the memcached interface once within a single class, then use this interface throughout the rest of your application.

For example, to create a basic interface, first configure the MemCachedClient and base SockIOPool settings:

```
public class MyClass {

    protected static MemCachedClient mcc = new MemCachedClient();

    static {

        String[] servers =
            {
                "localhost:11211",
            };

        Integer[] weights = { 1 };
        SockIOPool pool = SockIOPool.getInstance();

        pool.setServers( servers );
        pool.setWeights( weights );
```

In the above sample, the list of servers is configured by creating an array of the memcached instances to use. You can then configure individual weights for each server.

The remainder of the properties for the connection are optional, but you can set the connection numbers (initial connections, minimum connections, maximum connections, and the idle timeout) by setting the pool parameters:

```
pool.setInitConn( 5 );
pool.setMinConn( 5 );
pool.setMaxConn( 250 );
pool.setMaxIdle( 1000 * 60 * 60 * 6
```

Once the parameters have been configured, initialize the connection pool:

```
pool.initialize();
```

The pool, and the connection to your memcached instances should now be ready to use.

To set the hashing algorithm used to select the server used when storing a given key, use pool.setHashingAlg():

```
pool.setHashingAlg( SockIOPool.NEW_COMPAT_HASH );
```

Valid values are NEW_COMPAT_HASH, OLD_COMPAT_HASH and NATIVE_HASH are also basic modula hashing algorithms. For a consistent hashing algorithm, use CONSISTENT_HASH. These constants are equivalent to the corresponding hash settings within libmemcached.

The following table outlines the Java com.danga.MemCached methods and the equivalent generic methods in the memcached interface specification.

Java com.danga.MemCached Method	Equivalent Generic Method
get()	Generic get().
getMulti(keys)	Get the values of multiple keys, returning the information as Hash map using java.lang.String for the keys and java.lang.Object for the corresponding values.
set()	Generic set().
add()	Generic add().
replace()	Generic replace().
delete()	Generic delete().
incr()	Generic incr().
decr()	Generic decr().

15.6.3.9 Using the memcached TCP Text Protocol

Communicating with a memcached server can be achieved through either the TCP or UDP protocols. When using the TCP protocol, you can use a simple text based interface for the exchange of information.

When communicating with memcached, you can connect to the server using the port configured for the server. You can open a connection with the server without requiring authorization or login. As soon as you have connected, you can start to send commands to the server. When you have finished, you can terminate the connection without sending any specific disconnection command. Clients are encouraged to keep their connections open to decrease latency and improve performance.

Data is sent to the memcached server in two forms:

- Text lines, which are used to send commands to the server, and receive responses from the server.
- Unstructured data, which is used to receive or send the value information for a given key. Data is returned to the client in exactly the format it was provided.

Both text lines (commands and responses) and unstructured data are always terminated with the string \r\n. Because the data being stored may contain this sequence, the length of the data (returned by the client before the unstructured data is transmitted should be used to determine the end of the data.

Commands to the server are structured according to their operation:

- **Storage commands**: set, add, replace, append, prepend, cas

  Storage commands to the server take the form:

  ```
  command key [flags] [exptime] length [noreply]
  ```

  Or when using compare and swap (cas):

  ```
  cas key [flags] [exptime] length [casunique] [noreply]
  ```

  Where:

  - command: The command name.
    - set: Store value against key
    - add: Store this value against key if the key does not already exist
    - replace: Store this value against key if the key already exists
    - append: Append the supplied value to the end of the value for the specified key. The flags and exptime arguments should not be used.
    - prepend: Append value currently in the cache to the end of the supplied value for the specified key. The flags and exptime arguments should not be used.
    - cas: Set the specified key to the supplied value, only if the supplied casunique matches. This is effectively the equivalent of change the information if nobody has updated it since I last fetched it.
  - key: The key. All data is stored using a the specific key. The key cannot contain control characters or whitespace, and can be up to 250 characters in size.
  - flags: The flags for the operation (as an integer). Flags in memcached are transparent. The memcached server ignores the contents of the flags. They can be used by the client to indicate any type of information. In memcached 1.2.0 and lower the value is a 16-bit integer value. In memcached 1.2.1 and higher the value is a 32-bit integer.
  - exptime: The expiry time, or zero for no expiry.

  - length: The length of the supplied value block in bytes, excluding the terminating \r\n characters.

  - casunique: A unique 64-bit value of an existing entry. This is used to compare against the existing value. Use the value returned by the gets command when issuing cas updates.

  - noreply: Tells the server not to reply to the command.

  For example, to store the value abcdef into the key xyzkey, you would use:

  ```
  set xyzkey 0 0 6\r\nabcdef\r\n
  ```

  The return value from the server is one line, specifying the status or error information. For more information, see Table 15.3, “memcached Protocol Responses”.

- **Retrieval commands**: get, gets

  Retrieval commands take the form:

  ```
  get key1 [key2 .... keyn]
  gets key1 [key2 ... keyn]
  ```

  You can supply multiple keys to the commands, with each requested key separated by whitespace.

  The server responds with an information line of the form:

  ```
  VALUE key flags bytes [casunique]
  ```

  Where:

  - key: The key name.

  - flags: The value of the flag integer supplied to the memcached server when the value was stored.

  - bytes: The size (excluding the terminating \r\n character sequence) of the stored value.

  - casunique: The unique 64-bit integer that identifies the item.

  The information line is immediately followed by the value data block. For example:

  ```
  get xyzkey\r\n
  VALUE xyzkey 0 6\r\n
  abcdef\r\n
  ```

  If you have requested multiple keys, an information line and data block is returned for each key found. If a requested key does not exist in the cache, no information is returned.

- **Delete commands**: delete

  Deletion commands take the form:

  ```
  delete key [time] [noreply]
  ```

  Where:

  - key: The key name.

  - time: The time in seconds (or a specific Unix time) for which the client wishes the server to refuse add or replace commands on this key. All add, replace, get, and gets commands fail during this period. set operations succeed. After this period, the key is deleted permanently and all commands are accepted.

    If not supplied, the value is assumed to be zero (delete immediately).

  - noreply: Tells the server not to reply to the command.

  Responses to the command are either DELETED to indicate that the key was successfully removed, or NOT_FOUND to indicate that the specified key could not be found.

- **Increment/Decrement**: incr, decr

  The increment and decrement commands change the value of a key within the server without performing a separate get/set sequence. The operations assume that the currently stored value is a 64-bit integer. If the stored value is not a 64-bit integer, then the value is assumed to be zero before the increment or decrement operation is applied.

  Increment and decrement commands take the form:

  ```
  incr key value [noreply]
  decr key value [noreply]
  ```

  Where:

  - key: The key name.

  - value: An integer to be used as the increment or decrement value.

  - noreply: Tells the server not to reply to the command.

  The response is:

  - NOT_FOUND: The specified key could not be located.

  - value: The new value associated with the specified key.

  Values are assumed to be unsigned. For decr operations, the value is never decremented below 0. For incr operations, the value wraps around the 64-bit maximum.

- **Statistics commands**: stats

  The stats command provides detailed statistical information about the current status of the memcached instance and the data it is storing.

  Statistics commands take the form:

  ```
  STAT [name] [value]
  ```

  Where:

  - name: The optional name of the statistics to return. If not specified, the general statistics are returned.

  - value: A specific value to be used when performing certain statistics operations.

The return value is a list of statistics data, formatted as follows:

```
STAT name value
```

The statistics are terminated with a single line, END.

For more information, see Section 15.6.4, “Getting memcached Statistics”.

For reference, a list of the different commands supported and their formats is provided below.

Table 15.2 memcached Command Reference

Command	Command Formats
set	set key flags exptime length, set key flags exptime length noreply
add	add key flags exptime length, add key flags exptime length noreply
replace	replace key flags exptime length, replace key flags exptime length noreply
append	append key length, append key length noreply
prepend	prepend key length, prepend key length noreply
cas	cas key flags exptime length casunique, cas key flags exptime length casunique noreply
get	get key1 [key2 ... keyn]
gets	
delete	delete key, delete key noreply, delete key expiry, delete key expiry noreply
incr	incr key, incr key noreply, incr key value, incr key value noreply
decr	decr key, decr key noreply, decr key value, decr key value noreply
stat	stat, stat name, stat name value

When sending a command to the server, the response from the server is one of the settings in the following table. All response values from the server are terminated by \r\n:

Table 15.3 memcached Protocol Responses

String	Description
STORED	Value has successfully been stored.
NOT_STORED	The value was not stored, but not because of an error. For commands where you are adding a or updating a value if it exists (such as add and replace), or where the item has already been set to be deleted.
EXISTS	When using a cas command, the item you are trying to store already exists and has been modified since you last checked it.
NOT_FOUND	The item you are trying to store, update or delete does not exist or has already been deleted.
ERROR	You submitted a nonexistent command name.

String	Description
CLIENT_ERROR errorstring	There was an error in the input line, the detail is contained in errorstring.
SERVER_ERROR errorstring	There was an error in the server that prevents it from returning the information. In extreme conditions, the server may disconnect the client after this error occurs.
VALUE keys flags length	The requested key has been found, and the stored key, flags and data block are returned, of the specified length.
DELETED	The requested key was deleted from the server.
STAT name value	A line of statistics data.
END	The end of the statistics data.

15.6.4 Getting memcached Statistics

The memcached system has a built-in statistics system that collects information about the data being stored into the cache, cache hit ratios, and detailed information on the memory usage and distribution of information through the slab allocation used to store individual items. Statistics are provided at both a basic level that provide the core statistics, and more specific statistics for specific areas of the memcached server.

This information can be useful to ensure that you are getting the correct level of cache and memory usage, and that your slab allocation and configuration properties are set at an optimal level.

The stats interface is available through the standard memcached protocol, so the reports can be accessed by using telnet to connect to the memcached. The supplied memcached-tool includes support for obtaining the Section 15.6.4.2, “memcached Slabs Statistics” and Section 15.6.4.1, “memcached General Statistics” information. For more information, see Section 15.6.4.6, “Using memcached-tool”.

Alternatively, most of the language API interfaces provide a function for obtaining the statistics from the server.

For example, to get the basic stats using telnet:

```
shell> telnet localhost 11211
Trying ::1...
Connected to localhost.
Escape character is '^]'.
stats
STAT pid 23599
STAT uptime 675
STAT time 1211439587
STAT version 1.2.5
STAT pointer_size 32
STAT rusage_user 1.404992
STAT rusage_system 4.694685
STAT curr_items 32
STAT total_items 56361
STAT bytes 2642
STAT curr_connections 53
STAT total_connections 438
STAT connection_structures 55
STAT cmd_get 113482
STAT cmd_set 80519
STAT get_hits 78926
STAT get_misses 34556
STAT evictions 0
STAT bytes_read 6379783
STAT bytes_written 4860179
STAT limit_maxbytes 67108864
STAT threads 1
END
```

When using Perl and the Cache::Memcached module, the stats() function returns information about all the servers currently configured in the connection object, and total statistics for all the memcached servers as a whole.

For example, the following Perl script obtains the stats and dumps the hash reference that is returned:

```
use Cache::Memcached;
use Data::Dumper;

my $memc = new Cache::Memcached;
$memc->set_servers(\@ARGV);

print Dumper($memc->stats());
```

When executed on the same memcached as used in the Telnet example above we get a hash reference with the host by host and total statistics:

```
$VAR1 = {
    'hosts' => {
           'localhost:11211' => {
                      'misc' => {
                            'bytes' => '2421',
                            'curr_connections' => '3',
                            'connection_structures' => '56',
                            'pointer_size' => '32',
                            'time' => '1211440166',
                            'total_items' => '410956',
                            'cmd_set' => '588167',
                            'bytes_written' => '35715151',
                            'evictions' => '0',
                            'curr_items' => '31',
                            'pid' => '23599',
                            'limit_maxbytes' => '67108864',
                            'uptime' => '1254',
                            'rusage_user' => '9.857805',
                            'cmd_get' => '838451',
                            'rusage_system' => '34.096988',
                            'version' => '1.2.5',
                            'get_hits' => '581511',
                            'bytes_read' => '46665716',
                            'threads' => '1',
                            'total_connections' => '3104',
                            'get_misses' => '256940'
                          },
                      'sizes' => {
                             '128' => '16',
                             '64' => '15'
                           }
                    }
         },
    'self' => {},
    'total' => {
           'cmd_get' => 838451,
           'bytes' => 2421,
           'get_hits' => 581511,
           'connection_structures' => 56,
           'bytes_read' => 46665716,
           'total_items' => 410956,
           'total_connections' => 3104,
           'cmd_set' => 588167,
           'bytes_written' => 35715151,
           'curr_items' => 31,
           'get_misses' => 256940
         }
        };
```

The statistics are divided up into a number of distinct sections, and then can be requested by adding the type to the stats command. Each statistics output is covered in more detail in the following sections.

- General statistics, see Section 15.6.4.1, “memcached General Statistics”.
- Slab statistics (slabs), see Section 15.6.4.2, “memcached Slabs Statistics”.
- Item statistics (items), see Section 15.6.4.3, “memcached Item Statistics”.
- Size statistics (sizes), see Section 15.6.4.4, “memcached Size Statistics”.
- Detailed status (detail), see Section 15.6.4.5, “memcached Detail Statistics”.

15.6.4.1 memcached General Statistics

The output of the general statistics provides an overview of the performance and use of the memcached instance. The statistics returned by the command and their meaning is shown in the following table.

The following terms are used to define the value type for each statistics value:

- 32u: 32-bit unsigned integer
- 64u: 64-bit unsigned integer
- 32u:32u: Two 32-bit unsigned integers separated by a colon
- String: Character string

Statistic	Data type	Description	Version
pid	32u	Process ID of the memcached instance.	
uptime	32u	Uptime (in seconds) for this memcached instance.	
time	32u	Current time (as epoch).	
version	string	Version string of this instance.	
pointer_size	string	Size of pointers for this host specified in bits (32 or 64).	
rusage_user	32u:32u	Total user time for this instance (seconds:microseconds).	
rusage_system	32u:32u	Total system time for this instance (seconds:microseconds).	
curr_items	32u	Current number of items stored by this instance.	
total_items	32u	Total number of items stored during the life of this instance.	
bytes	64u	Current number of bytes used by this server to store items.	
curr_connections	32u	Current number of open connections.	
total_connections	32u	Total number of connections opened since the server started running.	
connection_structures	32u	Number of connection structures allocated by the server.	

Statistic	Data type	Description	Version
cmd_get	64u	Total number of retrieval requests (get operations).	
cmd_set	64u	Total number of storage requests (set operations).	
get_hits	64u	Number of keys that have been requested and found present.	
get_misses	64u	Number of items that have been requested and not found.	
delete_hits	64u	Number of keys that have been deleted and found present.	1.3.x
delete_misses	64u	Number of items that have been delete and not found.	1.3.x
incr_hits	64u	Number of keys that have been incremented and found present.	1.3.x
incr_misses	64u	Number of items that have been incremented and not found.	1.3.x
decr_hits	64u	Number of keys that have been decremented and found present.	1.3.x
decr_misses	64u	Number of items that have been decremented and not found.	1.3.x
cas_hits	64u	Number of keys that have been compared and swapped and found present.	1.3.x
cas_misses	64u	Number of items that have been compared and swapped and not found.	1.3.x
cas_badvalue	64u	Number of keys that have been compared and swapped, but the comparison (original) value did not match the supplied value.	1.3.x
evictions	64u	Number of valid items removed from cache to free memory for new items.	
bytes_read	64u	Total number of bytes read by this server from network.	
bytes_written	64u	Total number of bytes sent by this server to network.	
limit_maxbytes	32u	Number of bytes this server is permitted to use for storage.	
threads	32u	Number of worker threads requested.	
conn_yields	64u	Number of yields for connections (related to the -R option).	1.4.0

The most useful statistics from those given here are the number of cache hits, misses, and evictions.

A large number of get_misses may just be an indication that the cache is still being populated with information. The number should, over time, decrease in comparison to the number of cache get_hits. If, however, you have a large number of cache misses compared to cache hits after an extended period of execution, it may be an indication that the size of the cache is too small and you either need to increase the total memory size, or increase the number of the memcached instances to improve the hit ratio.

A large number of evictions from the cache, particularly in comparison to the number of items stored is a sign that your cache is too small to hold the amount of information that you regularly want to keep cached. Instead of items being retained in the cache, items are being evicted to make way for new items keeping the turnover of items in the cache high, reducing the efficiency of the cache.

15.6.4.2 memcached Slabs Statistics

To get the slabs statistics, use the stats slabs command, or the API equivalent.

The slab statistics provide you with information about the slabs that have created and allocated for storing information within the cache. You get information both on each individual slab-class and total statistics for the whole slab.

```
STAT 1:chunk_size 104
STAT 1:chunks_per_page 10082
STAT 1:total_pages 1
STAT 1:total_chunks 10082
STAT 1:used_chunks 10081
STAT 1:free_chunks 1
STAT 1:free_chunks_end 10079
STAT 9:chunk_size 696
STAT 9:chunks_per_page 1506
STAT 9:total_pages 63
STAT 9:total_chunks 94878
STAT 9:used_chunks 94878
STAT 9:free_chunks 0
STAT 9:free_chunks_end 0
STAT active_slabs 2
STAT total_malloced 67083616
END
```

Individual stats for each slab class are prefixed with the slab ID. A unique ID is given to each allocated slab from the smallest size up to the largest. The prefix number indicates the slab class number in relation to the calculated chunk from the specified growth factor. Hence in the example, 1 is the first chunk size and 9 is the 9th chunk allocated size.

The parameters returned for each chunk size and a description of each parameter are provided in the following table.

Statistic	Description	Version
chunk_size	Space allocated to each chunk within this slab class.	
chunks_per_page	Number of chunks within a single page for this slab class.	
total_pages	Number of pages allocated to this slab class.	
total_chunks	Number of chunks allocated to the slab class.	
used_chunks	Number of chunks allocated to an item..	
free_chunks	Number of chunks not yet allocated to items.	
free_chunks_end	Number of free chunks at the end of the last allocated page.	
get_hits	Number of get hits to this chunk	1.3.x
cmd_set	Number of set commands on this chunk	1.3.x
delete_hits	Number of delete hits to this chunk	1.3.x
incr_hits	Number of increment hits to this chunk	1.3.x
decr_hits	Number of decrement hits to this chunk	1.3.x
cas_hits	Number of CAS hits to this chunk	1.3.x
cas_badval	Number of CAS hits on this chunk where the existing value did not match	1.3.x
mem_requested	The true amount of memory of memory requested within this chunk	1.4.1

The following additional statistics cover the information for the entire server, rather than on a chunk by chunk basis:

Statistic	Description	Version
active_slabs	Total number of slab classes allocated.	
total_malloced	Total amount of memory allocated to slab pages.	

The key values in the slab statistics are the chunk_size, and the corresponding total_chunks and used_chunks parameters. These given an indication of the size usage of the chunks within the system. Remember that one key/value pair is placed into a chunk of a suitable size.

From these stats, you can get an idea of your size and chunk allocation and distribution. If you store many items with a number of largely different sizes, consider adjusting the chunk size growth factor to increase in larger steps to prevent chunk and memory wastage. A good indication of a bad growth factor is a high number of different slab classes, but with relatively few chunks actually in use within each slab. Increasing the growth factor creates fewer slab classes and therefore makes better use of the allocated pages.

15.6.4.3 memcached Item Statistics

To get the items statistics, use the stats items command, or the API equivalent.

The items statistics give information about the individual items allocated within a given slab class.

```
STAT items:2:number 1
STAT items:2:age 452
STAT items:2:evicted 0
STAT items:2:evicted_nonzero 0
STAT items:2:evicted_time 2
STAT items:2:outofmemory 0
STAT items:2:tailrepairs 0
...
STAT items:27:number 1
STAT items:27:age 452
STAT items:27:evicted 0
STAT items:27:evicted_nonzero 0
STAT items:27:evicted_time 2
STAT items:27:outofmemory 0
STAT items:27:tailrepairs 0
```

The prefix number against each statistics relates to the corresponding chunk size, as returned by the stats slabs statistics. The result is a display of the number of items stored within each chunk within each slab size, and specific statistics about their age, eviction counts, and out of memory counts. A summary of the statistics is given in the following table.

Statistic	Description	
number	The number of items currently stored in this slab class.	
age	The age of the oldest item within the slab class, in seconds.	
evicted	The number of items evicted to make way for new entries.	
evicted_time	The time of the last evicted entry	
evicted_nonzero	The time of the last evicted non-zero entry	1.4.0
outofmemory	The number of items for this slab class that have triggered an out of memory error (only value when the -M command line option is in effect).	
tailrepairs	Number of times the entries for a particular ID need repairing	

Item level statistics can be used to determine how many items are stored within a given slab and their freshness and recycle rate. You can use this to help identify whether there are certain slab classes that are triggering a much larger number of evictions that others.

15.6.4.4 memcached Size Statistics

To get size statistics, use the `stats sizes` command, or the API equivalent.

The size statistics provide information about the sizes and number of items of each size within the cache. The information is returned as two columns, the first column is the size of the item (rounded up to the nearest 32 byte boundary), and the second column is the count of the number of items of that size within the cache:

```
96 35
128 38
160 807
192 804
224 410
256 222
288 83
320 39
352 53
384 33
416 64
448 51
480 30
512 54
544 39
576 10065
```

Caution

Running this statistic locks up your cache as each item is read from the cache and its size calculated. On a large cache, this may take some time and prevent any set or get operations until the process completes.

The item size statistics are useful only to determine the sizes of the objects you are storing. Since the actual memory allocation is relevant only in terms of the chunk size and page size, the information is only useful during a careful debugging or diagnostic session.

15.6.4.5 memcached Detail Statistics

For `memcached` 1.3.x and higher, you can enable and obtain detailed statistics about the get, set, and del operations on theindividual keys stored in the cache, and determine whether the attempts hit (found) a particular key. These operations are only recorded while the detailed stats analysis is turned on.

To enable detailed statistics, you must send the `stats detail on` command to the `memcached` server:

```
$ telnet localhost 11211
Trying 127.0.0.1...
Connected to tiger.
Escape character is '^]'.
stats detail on
OK
```

Individual statistics are recorded for every `get`, `set` and `del` operation on a key, including keys that are not currently stored in the server. For example, if an attempt is made to obtain the value of key `abckey` and it does not exist, the `get` operating on the specified key are recorded while detailed statistics are in effect, even if the key is not currently stored. The hits, that is, the number of get or del operations for a key that exists in the server are also counted.

To turn detailed statistics off, send the stats detail off command to the memcached server:

```
$ telnet localhost 11211
Trying 127.0.0.1...
Connected to tiger.
Escape character is '^]'.
stats detail on
OK
```

To obtain the detailed statistics recorded during the process, send the stats detail dump command to the memcached server:

```
stats detail dump
PREFIX hykkey get 0 hit 0 set 1 del 0
PREFIX xyzkey get 0 hit 0 set 1 del 0
PREFIX yukkey get 1 hit 0 set 0 del 0
PREFIX abckey get 3 hit 3 set 1 del 0
END
```

You can use the detailed statistics information to determine whether your memcached clients are using a large number of keys that do not exist in the server by comparing the hit and get or del counts. Because the information is recorded by key, you can also determine whether the failures or operations are clustered around specific keys.

15.6.4.6 Using memcached-tool

The memcached-tool, located within the scripts directory within the memcached source directory. The tool provides convenient access to some reports and statistics from any memcached instance.

The basic format of the command is:

```
shell> ./memcached-tool hostname:port [command]
```

The default output produces a list of the slab allocations and usage. For example:

```
shell> memcached-tool localhost:11211 display
  #  Item_Size  Max_age   Pages   Count   Full?  Evicted Evict_Time OOM
  1      80B        93s       1      20      no        0        0    0
  2     104B        93s       1      16      no        0        0    0
  3     136B      1335s       1      28      no        0        0    0
  4     176B      1335s       1      24      no        0        0    0
  5     224B      1335s       1      32      no        0        0    0
  6     280B      1335s       1      34      no        0        0    0
  7     352B      1335s       1      36      no        0        0    0
  8     440B      1335s       1      46      no        0        0    0
  9     552B      1335s       1      58      no        0        0    0
 10     696B      1335s       1      66      no        0        0    0
 11     872B      1335s       1      89      no        0        0    0
 12     1.1K      1335s       1     112      no        0        0    0
 13     1.3K      1335s       1     145      no        0        0    0
 14     1.7K      1335s       1     123      no        0        0    0
 15     2.1K      1335s       1     198      no        0        0    0
 16     2.6K      1335s       1     199      no        0        0    0
 17     3.3K      1335s       1     229      no        0        0    0
 18     4.1K      1335s       1     248     yes       36        2    0
 19     5.2K      1335s       2     328      no        0        0    0
 20     6.4K      1335s       2     316     yes      387        1    0
 21     8.1K      1335s       3     381     yes      492        1    0
 22    10.1K      1335s       3     303     yes      598        2    0
 23    12.6K      1335s       5     405     yes      605        1    0
 24    15.8K      1335s       6     384     yes      766        2    0
 25    19.7K      1335s       7     357     yes      908      170    0
 26    24.6K      1336s       7     287     yes     1012        1    0
 27    30.8K      1336s       7     231     yes     1193      169    0
 28    38.5K      1336s       4     104     yes     1323      169    0
 29    48.1K      1336s       1      21     yes     1287        1    0
 30    60.2K      1336s       1      17     yes     1093      169    0
 31    75.2K      1337s       1      13     yes      713      168    0
 32    94.0K      1337s       1      10     yes      278      168    0
 33   117.5K      1336s       1       3      no        0        0    0
```

This output is the same if you specify the `command` as `display`:

```
shell> memcached-tool localhost:11211 display
  #  Item_Size  Max_age   Pages   Count   Full?  Evicted Evict_Time OOM
  1      80B        93s       1      20      no        0        0    0
  2     104B        93s       1      16      no        0        0    0
...
```

The output shows a summarized version of the output from the `slabs` statistics. The columns provided in the output are shown below:

- `#`: The slab number
- `Item_Size`: The size of the slab
- `Max_age`: The age of the oldest item in the slab
- `Pages`: The number of pages allocated to the slab
- `Count`: The number of items in this slab
- `Full?`: Whether the slab is fully populated
- `Evicted`: The number of objects evicted from this slab
- `Evict_Time`: The time (in seconds) since the last eviction
- `OOM`: The number of items that have triggered an out of memory error

You can also obtain a dump of the general statistics for the server using the `stats` command:

```
shell> memcached-tool localhost:11211 stats
#localhost:11211   Field       Value
         accepting_conns           1
                   bytes         162
              bytes_read         485
           bytes_written        6820
              cas_badval           0
                cas_hits           0
              cas_misses           0
               cmd_flush           0
                 cmd_get           4
                 cmd_set           2
             conn_yields           0
   connection_structures          11
        curr_connections          10
              curr_items           2
               decr_hits           0
             decr_misses           1
             delete_hits           0
           delete_misses           0
               evictions           0
                get_hits           4
              get_misses           0
               incr_hits           0
             incr_misses           2
          limit_maxbytes    67108864
     listen_disabled_num           0
                     pid       12981
            pointer_size          32
           rusage_system    0.013911
             rusage_user    0.011876
                 threads           4
                    time  1255518565
       total_connections          20
             total_items           2
                  uptime         880
                 version       1.4.2
```

15.6.5 memcached FAQ

Questions

- 16.6.5.1: [2092] Can memcached be run on a Windows environment?
- 16.6.5.2: [2092] What is the maximum size of an object you can store in memcached? Is that configurable?
- 16.6.5.3: [2092] Is it true memcached will be much more effective with db-read-intensive applications than with db-write-intensive applications?
- 16.6.5.4: [2092] Is there any overhead in not using persistent connections? If persistent is always recommended, what are the downsides (for example, locking up)?
- 16.6.5.5: [2092] How is an event such as a crash of one of the memcached servers handled by the memcached client?
- 16.6.5.6: [2092] What is a recommended hardware configuration for a memcached server?
- 16.6.5.7: [2093] Is memcached more effective for video and audio as opposed to textual read/writes?
- 16.6.5.8: [2093] Can memcached work with ASPX?
- 16.6.5.9: [2093] How expensive is it to establish a memcache connection? Should those connections be pooled?
- 16.6.5.10: [2093] How is the data handled when the memcached server is down?
- 16.6.5.11: [2093] How are auto-increment columns in the MySQL database coordinated across multiple instances of memcached?
- 16.6.5.12: [2093] Is compression available?
- 16.6.5.13: [2094] Can we implement different types of memcached as different nodes in the same server, so can there be deterministic and non-deterministic in the same server?
- 16.6.5.14: [2094] What are best practices for testing an implementation, to ensure that it improves performance, and to measure the impact of memcached configuration changes? And would you recommend keeping the configuration very simple to start?

Questions and Answers

16.6.5.1: Can memcached be run on a Windows environment?

No. Currently memcached is available only on the Unix/Linux platform. There is an unofficial port available, see http://www.codeplex.com/memcachedproviders.

16.6.5.2: What is the maximum size of an object you can store in memcached? Is that configurable?

The default maximum object size is 1MB. In memcached 1.4.2 and later, you can change the maximum size of an object using the `-I` command line option.

For versions before this, to increase this size, you have to re-compile memcached. You can modify the value of the `POWER_BLOCK` within the `slabs.c` file within the source.

In memcached 1.4.2 and higher, you can configure the maximum supported object size by using the `-I` command-line option. For example, to increase the maximum object size to 5MB:

```
$ memcached -I 5m
```

If an object is larger than the maximum object size, you must manually split it. memcached is very simple: you give it a key and some data, it tries to cache it in RAM. If you try to store more than the default maximum size, the value is just truncated for speed reasons.

16.6.5.3: Is it true memcached will be much more effective with db-read-intensive applications than with db-write-intensive applications?

Yes. memcached plays no role in database writes, it is a method of caching data already read from the database in RAM.

16.6.5.4: Is there any overhead in not using persistent connections? If persistent is always recommended, what are the downsides (for example, locking up)?

If you don't use persistent connections when communicating with memcached, there will be a small increase in the latency of opening the connection each time. The effect is comparable to use nonpersistent connections with MySQL.

In general, the chance of locking or other issues with persistent connections is minimal, because there is very little locking within memcached. If there is a problem, eventually your request will time out and return no result, so your application will need to load from MySQL again.

16.6.5.5: How is an event such as a crash of one of the memcached servers handled by the memcached client?

There is no automatic handling of this. If your client fails to get a response from a server, code a fallback mechanism to load the data from the MySQL database.

The client APIs all provide the ability to add and remove memcached instances on the fly. If within your application you notice that memcached server is no longer responding, you can remove the server from the list of servers, and keys will automatically be redistributed to another memcached server in the list. If retaining the cache content on all your servers is important, make sure you use an API that supports a consistent hashing algorithm. For more information, see Section 15.6.2.4, “memcached Hashing/Distribution Types”.

16.6.5.6: What is a recommended hardware configuration for a memcached server?

memcached has a very low processing overhead. All that is required is spare physical RAM capacity. A memcached server does not require a dedicated machine. If you have web, application, or database servers that have spare RAM capacity, then use them with memcached.

To build and deploy a dedicated memcached server, use a relatively low-power CPU, lots of RAM, and one or more Gigabit Ethernet interfaces.

16.6.5.7: Is memcached more effective for video and audio as opposed to textual read/writes?

memcached works equally well for all kinds of data. To memcached, any value you store is just a stream of data. Remember, though, that the maximum size of an object you can store in memcached is 1MB, but can be configured to be larger by using the `-I` option in memcached 1.4.2 and later, or by modifying the source in versions before 1.4.2. If you plan on using memcached with audio and video content, you will probably want to increase the maximum object size. Also remember that memcached is a solution for caching information for reading. It shouldn't be used for writes, except when updating the information in the cache.

16.6.5.8: Can memcached work with ASPX?

There are ports and interfaces for many languages and environments. ASPX relies on an underlying language such as C# or VisualBasic, and if you are using ASP.NET then there is a C# memcached library. For more information, see https://sourceforge.net/projects/memcacheddotnet/.

16.6.5.9: How expensive is it to establish a memcache connection? Should those connections be pooled?

Opening the connection is relatively inexpensive, because there is no security, authentication or other handshake taking place before you can start sending requests and getting results. Most APIs support a persistent connection to a memcached instance to reduce the latency. Connection pooling would depend on the API you are using, but if you are communicating directly over TCP/IP, then connection pooling would provide some small performance benefit.

16.6.5.10: How is the data handled when the memcached server is down?

The behavior is entirely application dependent. Most applications fall back to loading the data from the database (just as if they were updating the memcached information). If you are using multiple memcached servers, you might also remove a downed server from the list to prevent it from affecting performance. Otherwise, the client will still attempt to communicate with the memcached server that corresponds to the key you are trying to load.

16.6.5.11: How are auto-increment columns in the MySQL database coordinated across multiple instances of memcached?

They aren't. There is no relationship between MySQL and memcached unless your application (or, if you are using the MySQL UDFs for memcached, your database definition) creates one.

If you are storing information based on an auto-increment key into multiple instances of memcached, the information is only stored on one of the memcached instances anyway. The client uses the key value to determine which memcached instance to store the information. It doesn't store the same information across all the instances, as that would be a waste of cache memory.

16.6.5.12: Is compression available?

Yes. Most of the client APIs support some sort of compression, and some even allow you to specify the threshold at which a value is deemed appropriate for compression during storage.

16.6.5.13: Can we implement different types of `memcached` as different nodes in the same server, so can there be deterministic and non-deterministic in the same server?

Yes. You can run multiple instances of `memcached` on a single server, and in your client configuration you choose the list of servers you want to use.

16.6.5.14: What are best practices for testing an implementation, to ensure that it improves performance, and to measure the impact of `memcached` configuration changes? And would you recommend keeping the configuration very simple to start?

The best way to test the performance is to start up a `memcached` instance. First, modify your application so that it stores the data just before the data is about to be used or displayed into `memcached`. Since the APIs handle the serialization of the data, it should just be a one-line modification to your code. Then, modify the start of the process that would normally load that information from MySQL with the code that requests the data from `memcached`. If the data cannot be loaded from `memcached`, default to the MySQL process.

All of the changes required will probably amount to just a few lines of code. To get the best benefit, make sure you cache entire objects (for example, all the components of a web page, blog post, discussion thread, and so on), rather than using `memcached` as a simple cache of individual rows of MySQL tables.

Keeping the configuration simple at the start, or even over the long term, is easy with `memcached`. Once you have the basic structure up and running, often the only ongoing change is to add more servers into the list of servers used by your applications. You don't need to manage the `memcached` servers, and there is no complex configuration; just add more servers to the list and let the client API and the `memcached` servers make the decisions.

15.7 MySQL Proxy

The MySQL Proxy is an application that communicates over the network using the MySQL network protocol and provides communication between one or more MySQL servers and one or more MySQL clients. Because MySQL Proxy uses the MySQL network protocol, it can be used without modification with any MySQL-compatible client that uses the protocol. This includes the `mysql` command-line client, any clients that uses the MySQL client libraries, and any connector that supports the MySQL network protocol.

In the most basic configuration, MySQL Proxy simply interposes itself between the server and clients, passing queries from the clients to the MySQL Server and returning the responses from the MySQL Server to the appropriate client. In more advanced configurations, the MySQL Proxy can also monitor and alter the communication between the client and the server. Query interception enables you to add profiling, and interception of the exchanges is scriptable using the Lua scripting language.

By intercepting the queries from the client, the proxy can insert additional queries into the list of queries sent to the server, and remove the additional results when they are returned by the server. Using this functionality you can return the results from the original query to the client while adding informational statements to each query, for example, to monitor their execution time or progress, and separately log the results.

The proxy enables you to perform additional monitoring, filtering, or manipulation of queries without requiring you to make any modifications to the client and without the client even being aware that it is communicating with anything but a genuine MySQL server.

This documentation covers MySQL Proxy 0.8.2. And MySQL Proxy contains third-party code. For license information on third-party code, see Appendix A, *Licenses for Third-Party Components*.

> **Warning**
>
> MySQL Proxy is currently an Alpha release and should not be used within production environments.

Important

MySQL Proxy is compatible with MySQL 5.0 or later. Testing has not been performed with Version 4.1. Please provide feedback on your experiences using the MySQL Proxy Forum.

For release notes detailing the changes in each release of MySQL Proxy, see MySQL Proxy Release Notes.

15.7.1 MySQL Proxy Supported Platforms

MySQL Proxy is currently available as a precompiled binary for the following platforms:

- Linux (including Red Hat, Fedora, Debian, SuSE) and derivatives
- Mac OS X
- FreeBSD
- IBM AIX
- Sun Solaris
- Microsoft Windows (including Microsoft Windows XP, Microsoft Windows Vista, Microsoft Windows Server 2003, Microsoft Windows Server 2008)

Note

You must have the .NET Framework 1.1 or higher installed.

Other Unix/Linux platforms not listed should be compatible by using the source package and building MySQL Proxy locally.

System requirements for the MySQL Proxy application are the same as the main MySQL server. Currently MySQL Proxy is compatible only with MySQL 5.0.1 and later. MySQL Proxy is provided as a standalone, statically linked binary. You need not have MySQL or Lua installed.

15.7.2 Installing MySQL Proxy

You have three choices for installing MySQL Proxy:

- Precompiled binaries are available for a number of different platforms. See Section 15.7.2.1, “Installing MySQL Proxy from a Binary Distribution”.
- You can install from the source code to build on an environment not supported by the binary distributions. See Section 15.7.2.2, “Installing MySQL Proxy from a Source Distribution”.
- The latest version of the MySQL Proxy source code is available through a development repository is the best way to stay up to date with the latest fixes and revisions. See Section 15.7.2.3, “Installing MySQL Proxy from the Bazaar Repository”.

15.7.2.1 Installing MySQL Proxy from a Binary Distribution

If you download a binary package, you must extract and copy the package contents to your desired installation directory. The package contains files required by MySQL Proxy, including additional Lua scripts and other components required for execution.

To install, unpack the archive into the desired directory, then modify your `PATH` environment variable so that you can use the `mysql-proxy` command directly:

```
shell> cd /usr/local
shell> tar zxf mysql-proxy-0.8.2-platform.tar.gz
shell> PATH=$PATH:/usr/local/mysql-proxy-0.8.2-platform/sbin
```

To update the path globally on a system, you might need administrator privileges to modify the appropriate `/etc/profile`, `/etc/bashrc`, or other system configuration file.

On Windows, you can update the `PATH` environment variable using this procedure:

1. On the Windows desktop, right-click the My Computer icon, and select Properties.

2. Next select the Advanced tab from the System Properties menu that appears, and click the Environment Variables button.

3. Under **System Variables**, select Path, then click the Edit button. The Edit System Variable dialogue should appear.

The Microsoft Visual C++ runtime libraries are a requirement for running MySQL Proxy as of version 0.8.2. Users that do not have these libraries must download and install the Microsoft Visual C++ 2008 Service Pack 1 Redistributable Package MFC Security Update. Use the following link to obtain the package:

```
http://www.microsoft.com/download/en/details.aspx?id=26368
```

15.7.2.2 Installing MySQL Proxy from a Source Distribution

You can download a source package and compile the MySQL Proxy yourself. To build from source, you must have the following prerequisite components installed:

- `libevent` 1.x or higher (1.3b or later is preferred).

- `lua` 5.1.x or higher.

- `glib2` 2.6.0 or higher.

- `pkg-config`.

- `libtool` 1.5 or higher.

- MySQL 5.0.x or higher developer files.

> **Note**
>
> On some operating systems, you might need to manually build the required components to get the latest version. If you have trouble compiling MySQL Proxy, consider using a binary distributions instead.

After verifying that the prerequisite components are installed, configure and build MySQL Proxy:

```
shell> tar zxf mysql-proxy-0.8.2.tar.gz
shell> cd mysql-proxy-0.8.2
shell> ./configure
shell> make
```

To test the build, use the `check` target to `make`:

```
shell> make check
```

The tests try to connect to `localhost` using the `root` user. To provide a password, set the `MYSQL_PASSWORD` environment variable:

```
shell> MYSQL_PASSWORD=root_pwd make check
```

You can install using the `install` target:

```
shell> make install
```

By default, `mysql-proxy` is installed into `/usr/local/sbin/mysql-proxy`. The Lua example scripts are installed into `/usr/local/share`.

15.7.2.3 Installing MySQL Proxy from the Bazaar Repository

The MySQL Proxy source is available through a public Bazaar repository and is the quickest way to get the latest releases and fixes.

A build from the Bazaar repository requires that the following prerequisite components be installed:

- Bazaar 1.10.0 or later.
- `libtool` 1.5 or higher.
- `autoconf` 2.56 or higher.
- `automake` 1.10 or higher.
- `libevent` 1.x or higher (1.3b or later is preferred).
- `lua` 5.1.x or higher.
- `glib2` 2.4.0 or higher.
- `pkg-config`.
- MySQL 5.0.x or higher developer files.

The `mysql-proxy` source is hosted on Launchpad. To check out a local copy of the Bazaar repository, use `bzr`:

```
shell> bzr branch lp:mysql-proxy
```

The preceding command downloads a complete version of the Bazaar repository for `mysql-proxy`. The main source files are located within the `trunk` subdirectory. The configuration scripts must be generated before you can configure and build `mysql-proxy`. The `autogen.sh` script generates the required configuration scripts for you:

```
shell> sh ./autogen.sh
```

The `autogen.sh` script creates the standard `configure` script, which you then use to configure and build with `make`:

```
shell> ./configure
shell> make
shell> make install
```

To create a standalone source distribution, identical to the source distribution available for download, use this command:

```
shell> make distcheck
```

The preceding command creates the file `mysql-proxy-0.8.2.tar.gz` (with the corresponding current version) within the current directory.

15.7.2.4 Setting Up MySQL Proxy as a Windows Service

The MySQL distribution on Windows includes the `mysql-proxy-svc.exe` command that enables a MySQL Proxy instance to be managed by the Windows service control manager. You can control the service, including automatically starting and stopping it during boot, reboot and shutdown, without separately running the MySQL Proxy application.

To set up a MySQL Proxy service, use the `sc` command to create a new service using the MySQL Proxy service command. Specify the MySQL Proxy options on the `sc` command line, and identify the service with a unique name. For example, to configure a new MySQL Proxy instance that will automatically start when your system boots, redirecting queries to the local MySQL server:

```
C:\> sc create "Proxy" DisplayName= "MySQL Proxy" start= "auto" »
  binPath= "C:\Program Files\MySQL\mysql-proxy-0.8.2\bin\mysql-proxy-svc.exe »
  --proxy-backend-addresses=127.0.0.1:3306"
```

> **Note**
>
> The space following the equal sign after each property is required; failure to include it results in an error.

The preceding command creates a new service called `Proxy`. You can start and stop the service using the `net start|stop` command with the service name. The service is not automatically started after it is created. To start the service:

```
C:\> net start proxy
The MySQL Proxy service is starting.
The MySQL Proxy service was started successfully.
```

You can specify additional command-line options to the `sc` command. You can also set up multiple MySQL Proxy services on the same machine (providing they are configured to listen on different ports and/or IP addresses.

You can delete a service that you have created:

```
C:\> sc delete proxy
```

For more information on creating services using `sc`, see How to create a Windows service by using Sc.exe.

15.7.3 MySQL Proxy Command Options

To start MySQL Proxy, you can run it directly from the command line:

```
shell> mysql-proxy
```

For most situations, you specify at least the host name or address and the port number of the backend MySQL server to which the MySQL Proxy should pass queries.

You can specify options to `mysql-proxy` either on the command line, or by using a configuration file and the `--defaults-file` command-line option to specify the file location.

If you use a configuration file, format it as follows:

- Specify the options within a `[mysql-proxy]` configuration group. For example:

  ```
  [mysql-proxy]
  admin-address = host:port
  ```

- Specify all configuration options in the form of a configuration name and the value to set.

- For options that are a simple toggle on the command line (for example, `--proxy-skip-profiling`), use `true` or `false`. For example, the following is invalid:

  ```
  [mysql-proxy]
  proxy-skip-profiling
  ```

  But this is valid:

  ```
  [mysql-proxy]
  proxy-skip-profiling = true
  ```

- Give the configuration file Unix permissions of `0660` (readable and writable by user and group, no access for others).

Failure to adhere to any of these requirements causes `mysql-proxy` to generate an error during startup.

The following tables list the supported configuration file and command-line options.

Table 15.4 `mysql-proxy` Help Options

Format	Option File	Description
--help		Show help options
--help-admin		Show admin module options
--help-all		Show all help options
--help-proxy		Show proxy module options

Table 15.5 `mysql-proxy` Admin Options

Format	Option File	Description
--admin-address=host:port	admin-address=host:port	The admin module listening host and port
--admin-lua-script=file_name	admin-lua-script=file_name	Script to execute by the admin module
--admin-password=password	admin-password=password	Authentication password for admin module

Format	Option File	Description
--admin-username=user_name	admin-username=user_name	Authentication user name for admin module
--proxy-address=host:port	proxy-address=host:port	The listening proxy server host and port

Table 15.6 `mysql-proxy` Proxy Options

Format	Option File	Description	Removed
--no-proxy	no-proxy	Do not start the proxy module	
--proxy-backend-addresses=host:port	proxy-backend-addresses=host:port	The MySQL server host and port	
--proxy-fix-bug-25371	proxy-fix-bug-25371	Enable the fix for Bug #25371 for older libmysql versions	0.8.1
--proxy-lua-script=file_name	proxy-lua-script=file_name	Filename for Lua script for proxy operations	
--proxy-pool-no-change-user	proxy-pool-no-change-user	Do not use the protocol CHANGE_USER command to reset the connection when coming from the connection pool	
--proxy-read-only-backend-addresses=host:port	proxy-read-only-backend-addresses=host:port	The MySQL server host and port (read only)	
--proxy-skip-profiling	proxy-skip-profiling	Disable query profiling	

Table 15.7 `mysql-proxy` Applications Options

Format	Option File	Description
--basedir=dir_name	basedir=dir_name	The base directory prefix for paths in the configuration
--daemon	daemon	Start in daemon mode
--defaults-file=file_name		The configuration file to use
--event-threads=count	event-threads=count	The number of event-handling threads
--keepalive	keepalive	Try to restart the proxy if a crash occurs
--log-backtrace-on-crash	log-backtrace-on-crash	Try to invoke the debugger and generate a backtrace on crash
--log-file=file_name	log-file=file_name	The file where error messages are logged
--log-level=level	log-level=level	The logging level
--log-use-syslog	log-use-syslog	Log errors to syslog
--lua-cpath=dir_name	lua-cpath=dir_name	Set the LUA_CPATH
--lua-path=dir_name	lua-path=dir_name	Set the LUA_PATH
--max-open-files=count	max-open-files=count	The maximum number of open files to support
--pid-file=file_name	pid-file=file_name	File in which to store the process ID

Format	Option File	Description
--plugin-dir=dir_name	plugin-dir=dir_name	Directory containing plugin files
--plugins=plugin,...	plugins=plugin,...	List of plugins to load
--user=user_name	user=user_name	The user to use when running mysql-proxy
--version		Show version information

Except as noted in the following details, all of the options can be used within the configuration file by supplying the option and the corresponding value. For example:

```
[mysql-proxy]
log-file = /var/log/mysql-proxy.log
log-level = message
```

- `--help`, `-h`

Command-Line Format	--help
	-h

  Show available help options.

- `--help-admin`

Command-Line Format	--help-admin

  Show options for the admin module.

- `--help-all`

Command-Line Format	--help-all

  Show all help options.

- `--help-proxy`

Command-Line Format	--help-proxy

  Show options for the proxy module.

- `--admin-address=*host*:*port*`

Command-Line Format	--admin-address=host:port	
Option-File Format	admin-address=host:port	
	Permitted Values	
	Type	string
	Default	:4041

  The host name (or IP address) and port for the administration port. The default is `localhost:4041`.

- `--admin-lua-script=*file_name*`

Command-Line Format	--admin-lua-script=file_name

Option-File Format	admin-lua-script=file_name	
	Permitted Values	
	Type	file name
	Default	

The script to use for the proxy administration module.

- --admin-password=*password*

Command-Line Format	--admin-password=password	
Option-File Format	admin-password=password	
	Permitted Values	
	Type	string
	Default	

The password to use to authenticate users wanting to connect to the MySQL Proxy administration module. This module uses the MySQL protocol to request a user name and password for connections.

- --admin-username=*user_name*

Command-Line Format	--admin-username=user_name	
Option-File Format	admin-username=user_name	
	Permitted Values	
	Type	string
	Default	root

The user name to use to authenticate users wanting to connect to the MySQL Proxy administration module. This module uses the MySQL protocol to request a user name and password for connections. The default user name is root.

- --basedir=*dir_name*

Command-Line Format	--basedir=dir_name	
Option-File Format	basedir=dir_name	
	Permitted Values	
	Type	directory name

The base directory to use as a prefix for all other file name configuration options. The base name should be an absolute (not relative) directory. If you specify a relative directory, mysql-proxy generates an error during startup.

- --daemon

Command-Line Format	--daemon
Option-File Format	daemon

Starts the proxy in daemon mode.

- --defaults-file=*file_name*

Command-Line Format	`--defaults-file=file_name`

The file to read for configuration options. If not specified, MySQL Proxy takes options only from the command line.

- `--event-threads=`*`count`*

Command-Line Format	`--event-threads=count`	
Option-File Format	`event-threads=count`	
	Permitted Values	
	Type	`numeric`
	Default	`1`

The number of event threads to reserve to handle incoming requests.

- `--keepalive`

Command-Line Format	`--keepalive`
Option-File Format	`keepalive`

Create a process surrounding the main `mysql-proxy` process that attempts to restart the main `mysql-proxy` process in the event of a crash or other failure.

> **Note**
>
> The `--keepalive` option is not available on Microsoft Windows. When running as a service, `mysql-proxy` automatically restarts.

- `--log-backtrace-on-crash`

Command-Line Format	`--log-backtrace-on-crash`
Option-File Format	`log-backtrace-on-crash`

Log a backtrace to the error log and try to initialize the debugger in the event of a failure.

- `--log-file=`*`file_name`*

Command-Line Format	`--log-file=file_name`	
Option-File Format	`log-file=file_name`	
	Permitted Values	
	Type	`file name`

The file to use to record log information. If this option is not given, `mysql-proxy` logs to the standard error output.

- `--log-level=`*`level`*

Command-Line Format	`--log-level=level`
Option-File Format	`log-level=level`
	Permitted Values

	Type	enumeration
	Default	critical
	Valid Values	critical
		error
		warning
		info
		message
		debug

The log level to use when outputting error messages. Messages with that level (or lower) are output. For example, `message` level also outputs message with `info`, `warning`, and `error` levels.

- `--log-use-syslog`

Command-Line Format	`--log-use-syslog`
Option-File Format	`log-use-syslog`

Log errors to the syslog (Unix/Linux only).

- `--lua-cpath=`*`dir_name`*

Command-Line Format	`--lua-cpath=dir_name`	
Option-File Format	`lua-cpath=dir_name`	
	Permitted Values	
	Type	directory name

The `LUA_CPATH` to use when loading compiled modules or libraries for Lua scripts.

- `--lua-path=`*`dir_name`*

Command-Line Format	`--lua-path=dir_name`	
Option-File Format	`lua-path=dir_name`	
	Permitted Values	
	Type	directory name

The `LUA_CPATH` to use when loading modules for Lua.

- `--max-open-files=`*`count`*

Command-Line Format	`--max-open-files=count`	
Option-File Format	`max-open-files=count`	
	Permitted Values	
	Type	numeric

The maximum number of open files and sockets supported by the `mysql-proxy` process. Certain scripts might require a higher value.

- `--no-proxy`

Command-Line Format	--no-proxy	
Option-File Format	no-proxy	

Disable the proxy module.

- --plugin-dir=*dir_name*

Command-Line Format	--plugin-dir=dir_name	
Option-File Format	plugin-dir=dir_name	
	Permitted Values	
	Type	directory name

The directory to use when loading plugins for mysql-proxy.

- --plugins=*plugin*

Command-Line Format	--plugins=plugin,...	
Option-File Format	plugins=plugin,...	
	Permitted Values	
	Type	string

Loads a plugin.

When using this option on the command line, you can specify the option multiple times to specify multiple plugins. For example:

```
shell> mysql-proxy --plugins=proxy --plugins=admin
```

When using the option within the configuration file, you should separate multiple plugins by commas. The equivalent of the preceding example would be:

```
...
plugins=proxy,admin
```

- --proxy-address=*host*:*port*, -P *host*:*port*

Command-Line Format	--proxy-address=host:port	
	-P host:port	
Option-File Format	proxy-address=host:port	
	Permitted Values	
	Type	string
	Default	:4040

The listening host name (or IP address) and port of the proxy server. The default is :4040 (all IPs on port 4040).

- --proxy-read-only-backend-addresses=*host*:*port*, -r *host*:*port*

Command-Line Format	--proxy-read-only-backend-addresses=host:port

	-r host:port	
Option-File Format	proxy-read-only-backend-addresses=host:port	
	Permitted Values	
	Type	string

The listening host name (or IP address) and port of the proxy server for read-only connections. The default is for this information not to be set.

Note

Setting this value only configures the servers within the corresponding internal structure (see `proxy.global.backends` [2113]). You can determine the backend type by checking the `type` field for each connection.

You should therefore only use this option in combination with a script designed to make use of the different backend types.

When using this option on the command line, you can specify the option and the server multiple times to specify multiple backends. For example:

```
shell> mysql-proxy --proxy-read-only-backend-addresses=192.168.0.1:3306 --proxy-read-only-backend-addresses=
```

When using the option within the configuration file, you should separate multiple servers by commas. The equivalent of the preceding example would be:

```
...
proxy-read-only-backend-addresses = 192.168.0.1:3306,192.168.0.2:3306
```

- `--proxy-backend-addresses=host:port`, `-b host:port`

Command-Line Format	--proxy-backend-addresses=host:port	
	-b host:port	
Option-File Format	proxy-backend-addresses=host:port	
	Permitted Values	
	Type	string
	Default	127.0.0.1:3306

The host name (or IP address) and port of the MySQL server to connect to. You can specify multiple backend servers by supplying multiple options. Clients are connected to each backend server in round-robin fashion. For example, if you specify two servers A and B, the first client connection will go to server A; the second client connection to server B and the third client connection to server A.

When using this option on the command line, you can specify the option and the server multiple times to specify multiple backends. For example:

```
shell> mysql-proxy --proxy-backend-addresses 192.168.0.1:3306 --proxy-backend-addresses 192.168.0.2:3306
```

When using the option within the configuration file, you should separate multiple servers by commas. The equivalent of the preceding example would be:

```
...
proxy-backend-addresses = 192.168.0.1:3306,192.168.0.2:3306
```

- `--proxy-pool-no-change-user`

Command-Line Format	`--proxy-pool-no-change-user`
Option-File Format	`proxy-pool-no-change-user`

Disable use of the MySQL protocol `CHANGE_USER` command when reusing a connection from the pool of connections specified by the `proxy-backend-addresses` list.

- `--proxy-skip-profiling`

Command-Line Format	`--proxy-skip-profiling`
Option-File Format	`proxy-skip-profiling`

Disable query profiling (statistics time tracking). The default is for tracking to be enabled.

- `--proxy-fix-bug-25371`

Removed	0.8.1
Command-Line Format	`--proxy-fix-bug-25371`
Option-File Format	`proxy-fix-bug-25371`

Enable a workaround for an issue when connecting to a MySQL server later than 5.1.12 when using a MySQL client library of any earlier version.

This option was removed in `mysql-proxy` 0.8.1. Now, `mysql-proxy` returns an error message at the protocol level if it sees a `COM_CHANGE_USER` being sent to a server that has a version from 5.1.14 to 5.1.17.

- `--proxy-lua-script=`*`file_name`*, `-s `*`file_name`*

Command-Line Format	`--proxy-lua-script=file_name`	
	`-s file_name`	
Option-File Format	`proxy-lua-script=file_name`	
	Permitted Values	
	Type	`file name`

The Lua script file to be loaded. Note that the script file is not physically loaded and parsed until a connection is made. Also note that the specified Lua script is reloaded for each connection; if the content of the Lua script changes while `mysql-proxy` is running, the updated content is automatically used when a new connection is made.

- `--pid-file=`*`file_name`*

Command-Line Format	`--pid-file=file_name`	
Option-File Format	`pid-file=file_name`	
	Permitted Values	
	Type	`file name`

The name of the file in which to store the process ID.

- `--user=user_name`

Command-Line Format	`--user=user_name`	
Option-File Format	`user=user_name`	
	Permitted Values	
	Type	`string`

Run `mysql-proxy` as the specified `user`.

- `--version`, `-V`

Command-Line Format	`--version`
	`-V`

Show the version number.

The most common usage is as a simple proxy service (that is, without additional scripting). For basic proxy operation, you must specify at least one `proxy-backend-addresses` option to specify the MySQL server to connect to by default:

```
shell> mysql-proxy --proxy-backend-addresses=MySQL.example.com:3306
```

The default proxy port is `4040`, so you can connect to your MySQL server through the proxy by specifying the host name and port details:

```
shell> mysql --host=localhost --port=4040
```

If your server requires authentication information, this will be passed through natively without alteration by `mysql-proxy`, so you must also specify the required authentication information:

```
shell> mysql --host=localhost --port=4040 \
   --user=user_name --password=password
```

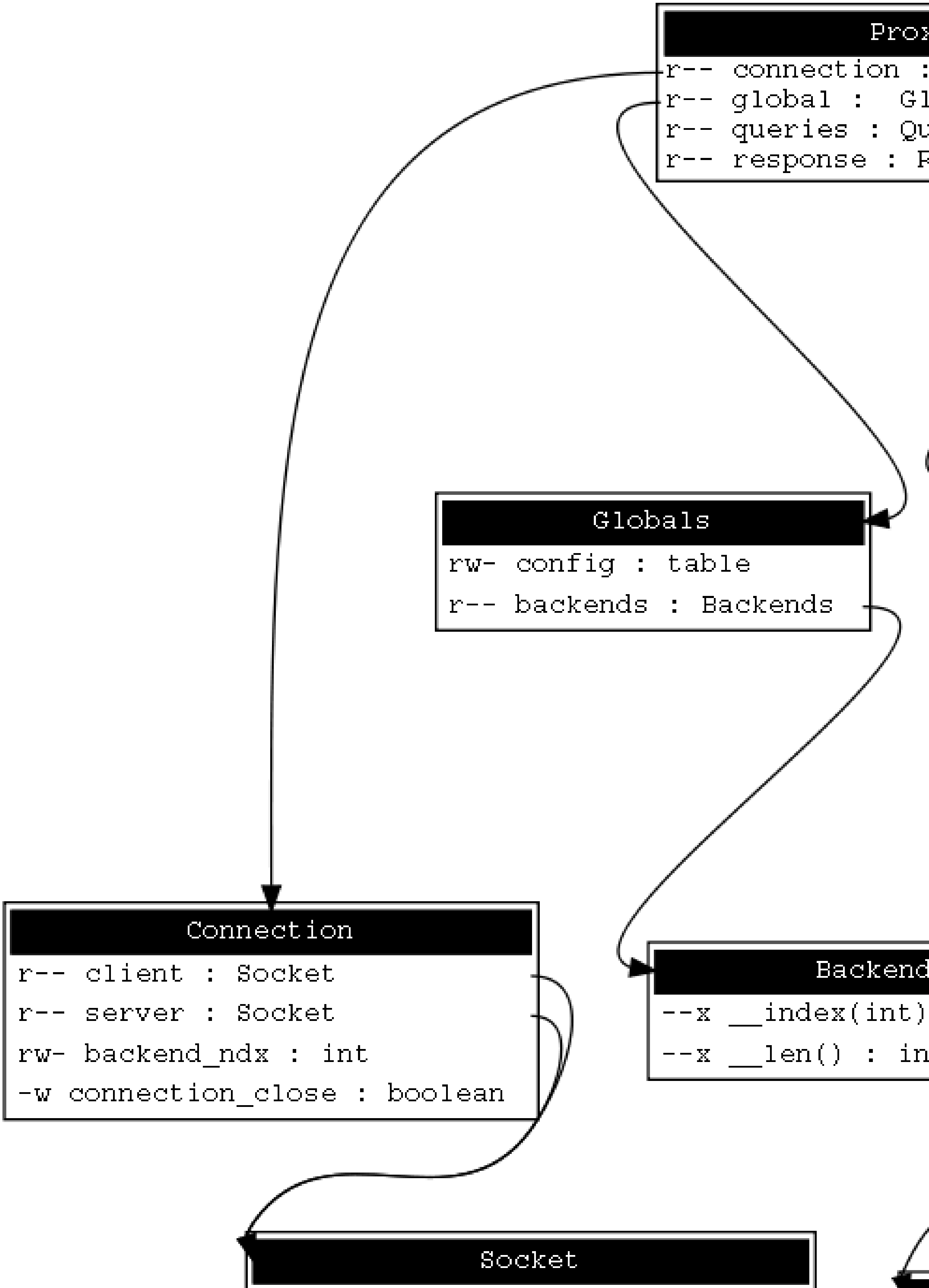
You can also connect to a read-only port (which filters out `UPDATE` and `INSERT` queries) by connecting to the read-only port. By default the host name is the default, and the port is `4042`, but you can alter the host/port information by using the `--proxy-read-only-backend-addresses` command-line option.

For more detailed information on how to use these command-line options, and `mysql-proxy` in general in combination with Lua scripts, see Section 15.7.5, “Using MySQL Proxy”.

15.7.4 MySQL Proxy Scripting

You can control how MySQL Proxy manipulates and works with the queries and results that are passed on to the MySQL server through the use of the embedded Lua scripting language. You can find out more about the Lua programming language from the Lua Web site.

The following diagram shows an overview of the classes exposed by MySQL Proxy.


Prox
r-- connection :
r-- global : Gl
r-- queries : Qu
r-- response : R
Globals
rw- config : table
r-- backends : Backends
Connection
r-- client : Socket
r-- server : Socket
rw- backend_ndx : int
-w connection_close : boolean
Backend
--x __index(int)
--x __len() : in
Socket

The primary interaction between MySQL Proxy and the server is provided by defining one or more functions through an Lua script. A number of functions are supported, according to different events and operations in the communication sequence between a client and one or more backend MySQL servers:

- `connect_server()`: This function is called each time a connection is made to MySQL Proxy from a client. You can use this function during load-balancing to intercept the original connection and decide which server the client should ultimately be attached to. If you do not define a special solution, a simple round-robin style distribution is used by default.
- `read_handshake()`: This function is called when the initial handshake information is returned by the server. You can capture the handshake information returned and provide additional checks before the authorization exchange takes place.
- `read_auth()`: This function is called when the authorization packet (user name, password, default database) are submitted by the client to the server for authentication.
- `read_auth_result()`: This function is called when the server returns an authorization packet to the client indicating whether the authorization succeeded.
- `read_query()`: This function is called each time a query is sent by the client to the server. You can use this to edit and manipulate the original query, including adding new queries before and after the original statement. You can also use this function to return information directly to the client, bypassing the server, which can be useful to filter unwanted queries or queries that exceed known limits.
- `read_query_result()`: This function is called each time a result is returned from the server, providing you have manually injected queries into the query queue. If you have not explicitly injected queries within the `read_query()` function, this function is not triggered. You can use this to edit the result set, or to remove or filter the result sets generated from additional queries you injected into the queue when using `read_query()`.

The following table lists MySQL proxy and server communication functions, the supplied information, and the direction of information flow when the function is triggered.

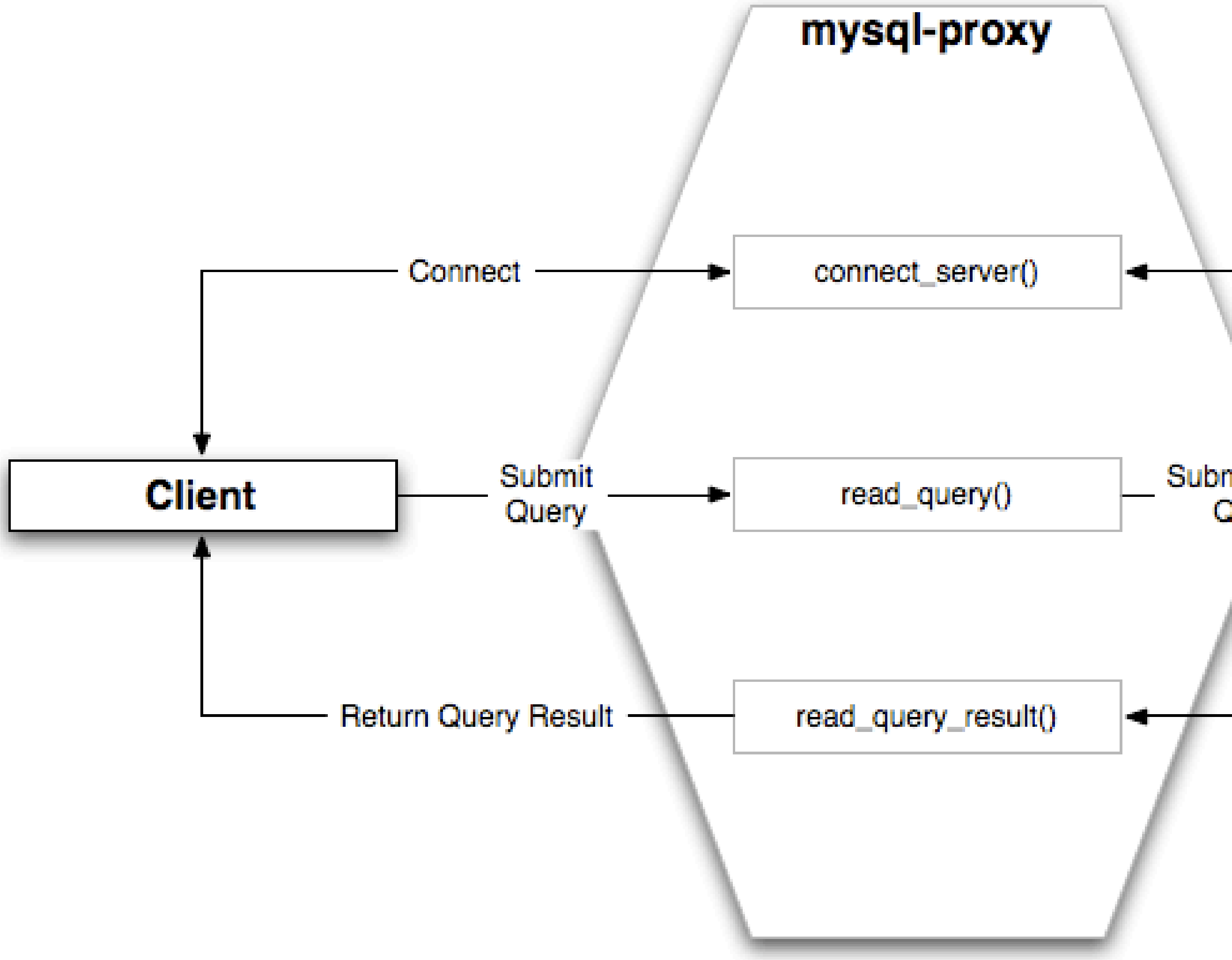
Function	Supplied Information	Direction
`connect_server()`	None	Client to Server
`read_handshake()`	None	Server to Client
`read_auth()`	None	Client to Server
`read_auth_result()`	None	Server to Client
`read_query()`	Query	Client to Server
`read_query_result()`	Query result	Server to Client

By default, all functions return a result that indicates whether the data should be passed on to the client or server (depending on the direction of the information being transferred). This return value can be overridden by explicitly returning a constant indicating that a particular response should be sent. For example, it is possible to construct result set information by hand within `read_query()` and to return the result set directly to the client without ever sending the original query to the server.

In addition to these functions, a number of built-in structures provide control over how MySQL Proxy forwards queries and returns the results by providing a simplified interface to elements such as the list of queries and the groups of result sets that are returned.

15.7.4.1 Proxy Scripting Sequence During Query Injection

The following figure gives an example of how the proxy might be used when injecting queries into the query queue. Because the proxy sits between the client and MySQL server, what the proxy sends to the server, and the information that the proxy ultimately returns to the client, need not match or correlate. Once the client has connected to the proxy, the sequence shown in the following diagram occurs for each individual query sent by the client.



1. When the client submits one query to the proxy, the `read_query()` function within the proxy is triggered. The function adds the query to the query queue.

2. Once manipulation by `read_query()` has completed, the queries are submitted, sequentially, to the MySQL server.

3. The MySQL server returns the results from each query, one result set for each query submitted. The `read_query_result()` function is triggered for each result set, and each invocation can decide which result set to return to the client

For example, you can queue additional queries into the global query queue to be processed by the server. This can be used to add statistical information by adding queries before and after the original query, changing the original query:

```
SELECT * FROM City;
```

Into a sequence of queries:

```
SELECT NOW();
SELECT * FROM City;
SELECT NOW();
```

You can also modify the original statement; for example, to add `EXPLAIN` to each statement executed to get information on how the statement was processed, again altering our original SQL statement into a number of statements:

```
SELECT * FROM City;
EXPLAIN SELECT * FROM City;
```

In both of these examples, the client would have received more result sets than expected. Regardless of how you manipulate the incoming query and the returned result, the number of queries returned by the proxy must match the number of original queries sent by the client.

You could adjust the client to handle the multiple result sets sent by the proxy, but in most cases you will want the existence of the proxy to remain transparent. To ensure that the number of queries and result sets match, you can use the MySQL Proxy `read_query_result()` to extract the additional result set information and return only the result set the client originally requested back to the client. You can achieve this by giving each query that you add to the query queue a unique ID, then filter out queries that do not match the original query ID when processing them with `read_query_result()`.

15.7.4.2 Internal Structures

There are a number of internal structures within the scripting element of MySQL Proxy. The primary structure is `proxy` and this provides an interface to the many common structures used throughout the script, such as connection lists and configured backend servers. Other structures, such as the incoming packet from the client and result sets are only available within the context of one of the scriptable functions.

The following table describes common attributes of the MySQL `proxy` scripting element.

Attribute	Description
`connection`	A structure containing the active client connections. For a list of attributes, see `proxy.connection` [2112].
`servers`	A structure containing the list of configured backend servers. For a list of attributes, see `proxy.global.backends` [2113].
`queries`	A structure containing the queue of queries that will be sent to the server during a single client query. For a list of attributes, see `proxy.queries` [2114].
`PROXY_VERSION`	The version number of MySQL Proxy, encoded in hex. You can use this to check that the version number supports a particular option from within the Lua script. Note that the value is encoded as a hex value, so to check the version is at least 0.5.1 you compare against `0x00501`.

`proxy.connection`

The `proxy.connection` object is read only, and provides information about the current connection, and is split into a `client` and `server` tables. This enables you to examine information about both the incoming client connections to the proxy (`client`), and to the backend servers (`server`).

The following table describes the client and server attributes of the `proxy.connection` object.

Attribute	Description
client.default_db	Default database requested by the client
client.username	User name used to authenticate
client.scrambled_password	The scrambled version of the password used to authenticate
client.dst.name	The combined address:port of the Proxy port used by this client (should match the --proxy-address configuration parameter)
client.dst.address	The IP address of the of the Proxy port used by this client
client.dst.port	The port number of the of the Proxy port used by this client
client.src.name	The combined address:port of the client (originating) TCP/IP endpoint
client.src.address	The IP address of the client (originating) TCP/IP port
client.src.port	The port of the client (originating) TCP/IP endpoint
server.scramble_buffer	The scramble buffer used to scramble the password
server.mysqld_version	The MySQL version number of the server
server.thread_id	The ID of the thread handling the connection to the current server
server.dst.name	The combined address:port for the backend server for the current connection (i.e. the connection to the MySQL server)
server.dst.address	The address for the backend server
server.dst.port	The port for the backend server
server.src.name	The combined address:port for the TCP/IP endpoint used by the Proxy to connect to the backend server
server.src.address	The address of the endpoint for the proxy-side connection to the MySQL server
server.src.port	The port of the endpoint for the proxy-side connection to the MySQL server

proxy.global.backends

The proxy.global.backends table is partially writable and contains an array of all the configured backend servers and the server metadata (IP address, status, etc.). You can determine the array index of the current connection using proxy.connection["backend_ndx"] which is the index into this table of the backend server being used by the active connection.

The attributes for each entry within the proxy.global.backends table are shown in the following table.

Attribute	Description
dst.name	The combined address:port of the backend server.
dst.address	The IP address of the backend server.
dst.port	The port of the backend server.
connected_clients	The number of clients currently connected.
state	The status of the backend server. See Backend State/Type Constants [2117].
type	The type of the backend server. You can use this to identify whether the backed was configured as a standard read/write backend, or a read-only backend. You can compare this value to the proxy.BACKEND_TYPE_RW and proxy.BACKEND_TYPE_RO.

proxy.queries

The proxy.queries object is a queue representing the list of queries to be sent to the server. The queue is not populated automatically, but if you do not explicitly populate the queue, queries are passed on to the backend server verbatim. Also, if you do not populate the query queue by hand, the read_query_result() function is not triggered.

The following functions are supported for populating the proxy.queries object.

Function	Description
append(id,packet,[options])	Appends a query to the end of the query queue. The id is an integer identifier that you can use to recognize the query results when they are returned by the server. The packet should be a properly formatted query packet. The optional options should be a table containing the options specific to this packet.
prepend(id,packet)	Prepends a query to the query queue. The id is an identifier that you can use to recognize the query results when they are returned by the server. The packet should be a properly formatted query packet.
reset()	Empties the query queue.
len()	Returns the number of query packets in the queue.

For example, you could append a query packet to the proxy.queries queue by using the append():

```
proxy.queries:append(1,packet)
```

The optional third argument to append() should contain the options for the packet. To have access to the result set through the read_query_result() function, set the resultset_is_needed flag to true:

```
proxy.queries:append( 1, packet, { resultset_is_needed = true } )
```

If that flag is false (the default), proxy will:

- Send the result set to the client as soon as it is received
- Reduce memory usage (because the result set is not stored internally for processing)
- Reduce latency of returning results to the client
- Pass data from server to client unaltered

The default mode is therefore quicker and useful if you only want to monitor the queries sent, and the basic statistics.

To perform any kind of manipulation on the returned data, you must set the flag to true, which will:

- Store the result set so that it can be processed.
- Enable modification of the result set before it is returned to the client.
- Enable you to discard the result set instead of returning it to the client.

proxy.response

The proxy.response structure is used when you want to return your own MySQL response, instead of forwarding a packet that you have received a backend server. The structure holds the response type information, an optional error message, and the result set (rows/columns) to return.

The following table describes the attributes of the `proxy.response` structure.

Attribute	Description
`type`	The type of the response. The type must be either `MYSQLD_PACKET_OK` or `MYSQLD_PACKET_ERR`. If the `MYSQLD_PACKET_ERR`, you should set the value of the `mysql.response.errmsg` with a suitable error message.
`errmsg`	A string containing the error message that will be returned to the client.
`resultset`	A structure containing the result set information (columns and rows), identical to what would be returned when returning a results from a `SELECT` query.

When using `proxy.response` you either set `proxy.response.type` to `proxy.MYSQLD_PACKET_OK` and then build `resultset` to contain the results to return, or set `proxy.response.type` to `proxy.MYSQLD_PACKET_ERR` and set the `proxy.response.errmsg` to a string with the error message. To send the completed result set or error message, you should return the `proxy.PROXY_SEND_RESULT` to trigger the return of the packet information.

An example of this can be seen in the `tutorial-resultset.lua` script within the MySQL Proxy package:

```
if string.lower(command) == "show" and string.lower(option) == "querycounter" then
        ---
        -- proxy.PROXY_SEND_RESULT requires
        --
        -- proxy.response.type to be either
        -- * proxy.MYSQLD_PACKET_OK or
        -- * proxy.MYSQLD_PACKET_ERR
        --
        -- for proxy.MYSQLD_PACKET_OK you need a resultset
        -- * fields
        -- * rows
        --
        -- for proxy.MYSQLD_PACKET_ERR
        -- * errmsg
        proxy.response.type = proxy.MYSQLD_PACKET_OK
        proxy.response.resultset = {
                fields = {
                        { type = proxy.MYSQL_TYPE_LONG, name = "global_query_counter", },
                        { type = proxy.MYSQL_TYPE_LONG, name = "query_counter", },
                },
                rows = {
                        { proxy.global.query_counter, query_counter }
                }
        }

        -- we have our result, send it back
        return proxy.PROXY_SEND_RESULT
elseif string.lower(command) == "show" and string.lower(option) == "myerror" then
        proxy.response.type = proxy.MYSQLD_PACKET_ERR
        proxy.response.errmsg = "my first error"

        return proxy.PROXY_SEND_RESULT
```

`proxy.response.resultset`

The `proxy.response.resultset` structure should be populated with the rows and columns of data to return. The structure contains the information about the entire result set, with the individual elements of the data shown in the following table.

The following table describes the attributes of the proxy.response.resultset structure.

Attribute	Description
fields	The definition of the columns being returned. This should be a dictionary structure with the type specifying the MySQL data type, and the name specifying the column name. Columns should be listed in the order of the column data that will be returned.
flags	A number of flags related to the result set. Valid flags include auto_commit (whether an automatic commit was triggered), no_good_index_used (the query executed without using an appropriate index), and no_index_used (the query executed without using any index).
rows	The actual row data. The information should be returned as an array of arrays. Each inner array should contain the column data, with the outer array making up the entire result set.
warning_count	The number of warnings for this result set.
affected_rows	The number of rows affected by the original statement.
insert_id	The last insert ID for an auto-incremented column in a table.
query_status	The status of the query operation. You can use the MYSQLD_PACKET_OK or MYSQLD_PACKET_ERR constants to populate this parameter.

For an example showing how to use this structure, see proxy.response [2114].

Proxy Return State Constants

The following constants are used internally by the proxy to specify the response to send to the client or server. All constants are exposed as values within the main proxy table.

Constant	Description
PROXY_SEND_QUERY	Causes the proxy to send the current contents of the queries queue to the server.
PROXY_SEND_RESULT	Causes the proxy to send a result set back to the client.
PROXY_IGNORE_RESULT	Causes the proxy to drop the result set (nothing is returned to the client).

As constants, these entities are available without qualification in the Lua scripts. For example, at the end of the read_query_result() you might return PROXY_IGNORE_RESULT:

```
return proxy.PROXY_IGNORE_RESULT
```

Packet State Constants

The following states describe the status of a network packet. These items are entries within the main proxy table.

Constant	Description
MYSQLD_PACKET_OK	The packet is OK
MYSQLD_PACKET_ERR	The packet contains error information
MYSQLD_PACKET_RAW	The packet contains raw data

Backend State/Type Constants

The following constants are used either to define the status or type of the backend MySQL server to which the proxy is connected. These items are entries within the main `proxy` table.

Constant	Description
`BACKEND_STATE_UNKNOWN`	The current status is unknown
`BACKEND_STATE_UP`	The backend is known to be up (available)
`BACKEND_STATE_DOWN`	The backend is known to be down (unavailable)
`BACKEND_TYPE_UNKNOWN`	Backend type is unknown
`BACKEND_TYPE_RW`	Backend is available for read/write
`BACKEND_TYPE_RO`	Backend is available only for read-only use

Server Command Constants

The values described in the table below are used in the packets exchanged between the client and server to identify the information in the rest of the packet. These items are entries within the main `proxy` table. The packet type is defined as the first character in the sent packet. For example, when intercepting packets from the client to edit or monitor a query, you would check that the first byte of the packet was of type `proxy.COM_QUERY`.

Constant	Description
`COM_SLEEP`	Sleep
`COM_QUIT`	Quit
`COM_INIT_DB`	Initialize database
`COM_QUERY`	Query
`COM_FIELD_LIST`	Field List
`COM_CREATE_DB`	Create database
`COM_DROP_DB`	Drop database
`COM_REFRESH`	Refresh
`COM_SHUTDOWN`	Shutdown
`COM_STATISTICS`	Statistics
`COM_PROCESS_INFO`	Process List
`COM_CONNECT`	Connect
`COM_PROCESS_KILL`	Kill
`COM_DEBUG`	Debug
`COM_PING`	Ping
`COM_TIME`	Time
`COM_DELAYED_INSERT`	Delayed insert
`COM_CHANGE_USER`	Change user
`COM_BINLOG_DUMP`	Binlog dump
`COM_TABLE_DUMP`	Table dump
`COM_CONNECT_OUT`	Connect out
`COM_REGISTER_SLAVE`	Register slave

Constant	Description
COM_STMT_PREPARE	Prepare server-side statement
COM_STMT_EXECUTE	Execute server-side statement
COM_STMT_SEND_LONG_DATA	Long data
COM_STMT_CLOSE	Close server-side statement
COM_STMT_RESET	Reset statement
COM_SET_OPTION	Set option
COM_STMT_FETCH	Fetch statement
COM_DAEMON	Daemon (MySQL 5.1 only)
COM_ERROR	Error

MySQL Type Constants

These constants are used to identify the field types in the query result data returned to clients from the result of a query. These items are entries within the main `proxy` table.

Constant	Field Type
MYSQL_TYPE_DECIMAL	Decimal
MYSQL_TYPE_NEWDECIMAL	Decimal (MySQL 5.0 or later)
MYSQL_TYPE_TINY	Tiny
MYSQL_TYPE_SHORT	Short
MYSQL_TYPE_LONG	Long
MYSQL_TYPE_FLOAT	Float
MYSQL_TYPE_DOUBLE	Double
MYSQL_TYPE_NULL	Null
MYSQL_TYPE_TIMESTAMP	Timestamp
MYSQL_TYPE_LONGLONG	Long long
MYSQL_TYPE_INT24	Integer
MYSQL_TYPE_DATE	Date
MYSQL_TYPE_TIME	Time
MYSQL_TYPE_DATETIME	Datetime
MYSQL_TYPE_YEAR	Year
MYSQL_TYPE_NEWDATE	Date (MySQL 5.0 or later)
MYSQL_TYPE_ENUM	Enumeration
MYSQL_TYPE_SET	Set
MYSQL_TYPE_TINY_BLOB	Tiny Blob
MYSQL_TYPE_MEDIUM_BLOB	Medium Blob
MYSQL_TYPE_LONG_BLOB	Long Blob
MYSQL_TYPE_BLOB	Blob
MYSQL_TYPE_VAR_STRING	Varstring
MYSQL_TYPE_STRING	String

Constant	Field Type
MYSQL_TYPE_TINY	Tiny (compatible with MYSQL_TYPE_CHAR)
MYSQL_TYPE_ENUM	Enumeration (compatible with MYSQL_TYPE_INTERVAL)
MYSQL_TYPE_GEOMETRY	Geometry
MYSQL_TYPE_BIT	Bit

15.7.4.3 Capturing a Connection with connect_server()

When the proxy accepts a connection from a MySQL client, the connect_server() function is called.

There are no arguments to the function, but you can use and if necessary manipulate the information in the proxy.connection table, which is unique to each client session.

For example, if you have multiple backend servers, you can specify which server that connection should use by setting the value of proxy.connection.backend_ndx to a valid server number. The following code chooses between two servers based on whether the current time in minutes is odd or even:

```
function connect_server()
        print("--> a client really wants to talk to a server")
        if (tonumber(os.date("%M")) % 2 == 0) then
                proxy.connection.backend_ndx = 2
                print("Choosing backend 2")
        else
                proxy.connection.backend_ndx = 1
                print("Choosing backend 1")
        end
        print("Using " .. proxy.global.backends[proxy.connection.backend_ndx].dst.name)
end
```

This example also displays the IP address/port combination by accessing the information from the internal proxy.global.backends table.

15.7.4.4 Examining the Handshake with read_handshake()

Handshake information is sent by the server to the client after the initial connection (through connect_server()) has been made. The handshake information contains details about the MySQL version, the ID of the thread that will handle the connection information, and the IP address of the client and server. This information is exposed through the proxy.connection structure.

- proxy.connection.server.mysqld_version: The version of the MySQL server.
- proxy.connection.server.thread_id: The thread ID.
- proxy.connection.server.scramble_buffer: The password scramble buffer.
- proxy.connection.server.dst.name: The IP address of the server.
- proxy.connection.client.src.name: The IP address of the client.

For example, you can print out the handshake data and refuse clients by IP address with the following function:

```
function read_handshake()
        print("<-- let's send him some information about us")
        print("    mysqld-version: " .. proxy.connection.server.mysqld_version)
        print("    thread-id     : " .. proxy.connection.server.thread_id)
        print("    scramble-buf  : " .. string.format("%q",proxy.connection.server.scramble_buffer))
        print("    server-addr   : " .. proxy.connection.server.dst.name)
        print("    client-addr   : " .. proxy.connection.client.dst.name)

        if not proxy.connection.client.src.name:match("^127.0.0.1:") then
                proxy.response.type = proxy.MYSQLD_PACKET_ERR
                proxy.response.errmsg = "only local connects are allowed"

                print("we don't like this client");

                return proxy.PROXY_SEND_RESULT
        end
end
```

Note that you must return an error packet to the client by using `proxy.PROXY_SEND_RESULT`.

15.7.4.5 Examining the Authentication Credentials with `read_auth()`

The `read_auth()` function is triggered when an authentication handshake is initiated by the client. In the execution sequence, `read_auth()` occurs immediately after `read_handshake()`, so the server selection has already been made, but the connection and authorization information has not yet been provided to the backend server.

You can obtain the authentication information by examining the `proxy.connection.client` structure. For more information, see `proxy.connection` [2112].

For example, you can print the user name and password supplied during authorization using:

```
function read_auth()
        print("    username      : " .. proxy.connection.client.username)
        print("    password      : " .. string.format("%q", proxy.connection.client.scrambled_password))
end
```

You can interrupt the authentication process within this function and return an error packet back to the client by constructing a new packet and returning `proxy.PROXY_SEND_RESULT`:

```
proxy.response.type = proxy.MYSQLD_PACKET_ERR
proxy.response.errmsg = "Logins are not allowed"
return proxy.PROXY_SEND_RESULT
```

15.7.4.6 Accessing Authentication Information with `read_auth_result()`

The return packet from the server during authentication is captured by `read_auth_result()`. The only argument to this function is the authentication packet returned by the server. As the packet is a raw MySQL network protocol packet, you must access the first byte to identify the packet type and contents. The `MYSQLD_PACKET_ERR` and `MYSQLD_PACKET_OK` constants can be used to identify whether the authentication was successful:

```
function read_auth_result(auth)
        local state = auth.packet:byte()

        if state == proxy.MYSQLD_PACKET_OK then
                print("<-- auth ok");
        elseif state == proxy.MYSQLD_PACKET_ERR then
                print("<-- auth failed");
        else
                print("<-- auth ... don't know: " .. string.format("%q", auth.packet));
        end
end
```

If a long-password capable client tries to authenticate to a server that supports long passwords, but the user password provided is actually short, `read_auth_result()` will be called twice. The first time, `auth.packet:byte()` will equal 254, indicating that the client should try again using the old password protocol. The second time time `read_auth_result()/` is called, `auth.packet:byte()` will indicate whether the authentication actually succeeded.

15.7.4.7 Manipulating Queries with `read_query()`

The `read_query()` function is called once for each query submitted by the client and accepts a single argument, the query packet that was provided. To access the content of the packet, you must parse the packet contents manually.

For example, you can intercept a query packet and print out the contents using the following function definition:

```
function read_query( packet )
        if packet:byte() == proxy.COM_QUERY then
                print("we got a normal query: " .. packet:sub(2))
        end
end
```

This example checks the first byte of the packet to determine the type. If the type is `COM_QUERY` (see Server Command Constants [2117]), we extract the query from the packet and print it. The structure of the packet type supplied is important. In the case of a `COM_QUERY` packet, the remaining contents of the packet are the text of the query string. In this example, no changes have been made to the query or the list of queries that will ultimately be sent to the MySQL server.

To modify a query, or add new queries, you must populate the query queue (`proxy.queries`), then execute the queries that you have placed into the queue. If you do not modify the original query or the queue, the query received from the client is sent to the MySQL server verbatim.

When adding queries to the queue, you should follow these guidelines:

- The packets inserted into the queue must be valid query packets. For each packet, you must set the initial byte to the packet type. If you are appending a query, you can append the query statement to the rest of the packet.

- Once you add a query to the queue, the queue is used as the source for queries sent to the server. If you add a query to the queue to add more information, you must also add the original query to the queue or it will not be executed.

- Once the queue has been populated, you must set the return value from `read_query()` to indicate whether the query queue should be sent to the server.

- When you add queries to the queue, you should add an ID. The ID you specify is returned with the result set so that you identify each query and corresponding result set. The ID has no other purpose than as an identifier for correlating the query and result set. When operating in a passive mode, during profiling for example, you identify the original query and the corresponding result set so that the results expected by the client can be returned correctly.

- Unless your client is designed to cope with more result sets than queries, you should ensure that the number of queries from the client match the number of results sets returned to the client. Using the unique ID and removing result sets you inserted will help.

Normally, the `read_query()` and `read_query_result()` function are used in conjunction with each other to inject additional queries and remove the additional result sets. However, `read_query_result()` is only called if you populate the query queue within `read_query()`.

15.7.4.8 Manipulating Results with read_query_result()

The read_query_result() is called for each result set returned by the server only if you have manually injected queries into the query queue. If you have not manipulated the query queue, this function is not called. The function supports a single argument, the result packet, which provides a number of properties:

- id: The ID of the result set, which corresponds to the ID that was set when the query packet was submitted to the server when using append(id) on the query queue. You must have set the resultset_is_needed flag to append to intercept the result set before it is returned to the client. See proxy.queries [2114].
- query: The text of the original query.
- query_time: The number of microseconds required to receive the first row of a result set since the query was sent to the server.
- response_time: The number of microseconds required to receive the last row of the result set since the query was sent to the server.
- resultset: The content of the result set data.

By accessing the result information from the MySQL server, you can extract the results that match the queries that you injected, return different result sets (for example, from a modified query), and even create your own result sets.

The following Lua script, for example, will output the query, followed by the query time and response time (that is, the time to execute the query and the time to return the data for the query) for each query sent to the server:

```
function read_query( packet )
        if packet:byte() == proxy.COM_QUERY then
                print("we got a normal query: " .. packet:sub(2))

                proxy.queries:append(1, packet )

                return proxy.PROXY_SEND_QUERY
        end
end

function read_query_result(inj)
        print("query-time: " .. (inj.query_time / 1000) .. "ms")
        print("response-time: " .. (inj.response_time / 1000) .. "ms")
end
```

You can access the rows of returned results from the result set by accessing the rows property of the resultset property of the result that is exposed through read_query_result(). For example, you can iterate over the results showing the first column from each row using this Lua fragment:

```
for row in inj.resultset.rows do
        print("injected query returned: " .. row[1])
end
```

Just like read_query(), read_query_result() can return different values for each result according to the result returned. If you have injected additional queries into the query queue, for example, remove the results returned from those additional queries and return only the results from the query originally submitted by the client.

The following example injects additional SELECT NOW() statements into the query queue, giving them a different ID to the ID of the original query. Within read_query_result(), if the ID for the injected queries is identified, we display the result row, and return the `proxy.PROXY_IGNORE_RESULT` from the function so that the result is not returned to the client. If the result is from any other query, we print out the query time information for the query and return the default, which passes on the result set unchanged. We could also have explicitly returned `proxy.PROXY_IGNORE_RESULT` to the MySQL client.

```
function read_query( packet )
        if packet:byte() == proxy.COM_QUERY then
                proxy.queries:append(2, string.char(proxy.COM_QUERY) .. "SELECT NOW()", {resultset_is_neede
                proxy.queries:append(1, packet, {resultset_is_needed = true})
                proxy.queries:append(2, string.char(proxy.COM_QUERY) .. "SELECT NOW()", {resultset_is_neede

                return proxy.PROXY_SEND_QUERY
        end
end


function read_query_result(inj)
        if inj.id == 2 then
                for row in inj.resultset.rows do
                        print("injected query returned: " .. row[1])
                end
                return proxy.PROXY_IGNORE_RESULT
        else
                print("query-time: " .. (inj.query_time / 1000) .. "ms")
                print("response-time: " .. (inj.response_time / 1000) .. "ms")
        end
end
```

For further examples, see Section 15.7.5, “Using MySQL Proxy”.

15.7.5 Using MySQL Proxy

There are a number of different ways to use MySQL Proxy. At the most basic level, you can allow MySQL Proxy to pass queries from clients to a single server. To use MySQL Proxy in this mode, you just have to specify on the command line the backend server to which the proxy should connect:

```
shell> mysql-proxy --proxy-backend-addresses=sakila:3306
```

If you specify multiple backend MySQL servers, the proxy connects each client to each server in a round-robin fashion. Suppose that you have two MySQL servers, A and B. The first client to connect is connected to server A, the second to server B, the third to server A. For example:

```
shell> mysql-proxy \
     --proxy-backend-addresses=narcissus:3306 \
     --proxy-backend-addresses=nostromo:3306
```

When you specify multiple servers in this way, the proxy automatically identifies when a MySQL server has become unavailable and marks it accordingly. New connections are automatically attached to a server that is available, and a warning is reported to the standard output from `mysql-proxy`:

```
network-mysqld.c.367: connect(nostromo:3306) failed: Connection refused
network-mysqld-proxy.c.2405: connecting to backend (nostromo:3306) failed, marking it as down for ...
```

Lua scripts enable a finer level of control, both over the connections and their distribution and how queries and result sets are processed. When using an Lua script, you must specify the name of the script on the command line using the `--proxy-lua-script` option:

```
shell> mysql-proxy --proxy-lua-script=mc.lua --proxy-backend-addresses=sakila:3306
```

When you specify a script, the script is not executed until a connection is made. This means that faults with the script are not raised until the script is executed. Script faults will not affect the distribution of queries to backend MySQL servers.

Note

Because a script is not read until the connection is made, you can modify the contents of the Lua script file while the proxy is still running and the modified script is automatically used for the next connection. This ensures that MySQL Proxy remains available because it need not be restarted for the changes to take effect.

15.7.5.1 Using the Administration Interface

The `mysql-proxy` administration interface can be accessed using any MySQL client using the standard protocols. You can use the administration interface to gain information about the proxy server as a whole - standard connections to the proxy are isolated to operate as if you were connected directly to the backend MySQL server.

In `mysql-proxy` 0.8.0 and earlier, a rudimentary interface was built into the proxy. In later versions this was replaced so that you must specify an administration script to be used when users connect to the administration interface.

To use the administration interface, specify the user name and password required to connect to the admin service, using the `--admin-username` and `--admin-password` options. You must also specify the Lua script to be used as the interface to the administration service by using the `admin-lua-script` script option to point to a Lua script.

For example, you can create a basic interface to the internal components of the `mysql-proxy` system using the following script, written by Diego Medina:

```
--[[

   Copyright 2008, 2010, Oracle and/or its affiliates. All rights reserved.

   This program is free software; you can redistribute it and/or modify
   it under the terms of the GNU General Public License as published by
   the Free Software Foundation; version 2 of the License.

   This program is distributed in the hope that it will be useful,
   but WITHOUT ANY WARRANTY; without even the implied warranty of
   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
   GNU General Public License for more details.

   You should have received a copy of the GNU General Public License
   along with this program; if not, write to the Free Software
   Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

--]]

-- admin.lua

--[[

    See http://www.chriscalender.com/?p=41
    (Thanks to Chris Calender)
    See http://datacharmer.blogspot.com/2009/01/mysql-proxy-is-back.html
    (Thanks Giuseppe Maxia)

--]]

function set_error(errmsg)
```

```
    proxy.response = {
        type = proxy.MYSQLD_PACKET_ERR,
        errmsg = errmsg or "error"
    }
end

function read_query(packet)
    if packet:byte() ~= proxy.COM_QUERY then
        set_error("[admin] we only handle text-based queries (COM_QUERY)")
        return proxy.PROXY_SEND_RESULT
    end

    local query = packet:sub(2)
    local rows = { }
    local fields = { }

    -- try to match the string up to the first non-alphanum
    local f_s, f_e, command = string.find(packet, "^%s*(%w+)", 2)
    local option

    if f_e then
            -- if that match, take the next sub-string as option
            f_s, f_e, option = string.find(packet, "^%s+(%w+)", f_e + 1)
    end

    -- we got our commands, execute it
    if command == "show" and option == "querycounter" then
            ---
            -- proxy.PROXY_SEND_RESULT requires
            --
            -- proxy.response.type to be either
            -- * proxy.MYSQLD_PACKET_OK or
            -- * proxy.MYSQLD_PACKET_ERR
            --
            -- for proxy.MYSQLD_PACKET_OK you need a resultset
            -- * fields
            -- * rows
            --
            -- for proxy.MYSQLD_PACKET_ERR
            -- * errmsg
            proxy.response.type = proxy.MYSQLD_PACKET_OK
            proxy.response.resultset = {
                    fields = {
                            { type = proxy.MYSQL_TYPE_LONG, name = "query_counter", },
                    },
                    rows = {
                            { proxy.global.query_counter }
                    }
            }

            -- we have our result, send it back
            return proxy.PROXY_SEND_RESULT
    elseif command == "show" and option == "myerror" then
            proxy.response.type = proxy.MYSQLD_PACKET_ERR
            proxy.response.errmsg = "my first error"

            return proxy.PROXY_SEND_RESULT

    elseif string.sub(packet, 2):lower() == 'select help' then
            return show_process_help()

    elseif string.sub(packet, 2):lower() == 'show proxy processlist' then
            return show_process_table()

    elseif query == "SELECT * FROM backends" then
        fields = {
            { name = "backend_ndx",
              type = proxy.MYSQL_TYPE_LONG },

            { name = "address",
              type = proxy.MYSQL_TYPE_STRING },
            { name = "state",
              type = proxy.MYSQL_TYPE_STRING },
            { name = "type",
              type = proxy.MYSQL_TYPE_STRING },
        }

        for i = 1, #proxy.global.backends do
            local b = proxy.global.backends[i]

            rows[#rows + 1] = {
                i, b.dst.name, b.state, b.type
            }
        end
    else
        set_error()
        return proxy.PROXY_SEND_RESULT
    end

    proxy.response = {
        type = proxy.MYSQLD_PACKET_OK,
        resultset = {
            fields = fields,
            rows = rows
        }
    }
    return proxy.PROXY_SEND_RESULT
end


function make_dataset (header, dataset)
    proxy.response.type = proxy.MYSQLD_PACKET_OK

    proxy.response.resultset = {
        fields = {},
        rows = {}
    }
    for i,v in pairs (header) do
        table.insert(proxy.response.resultset.fields, {type = proxy.MYSQL_TYPE_STRING, name = v})
    end
    for i,v in pairs (dataset) do
        table.insert(proxy.response.resultset.rows, v )
    end
    return proxy.PROXY_SEND_RESULT
end

function show_process_table()
    local dataset = {}
    local header = { 'Id', 'IP Address', 'Time' }
    local rows = {}
    for t_i, t_v in pairs (proxy.global.process) do
        for s_i, s_v in pairs ( t_v ) do
            table.insert(rows, { t_i, s_v.ip, os.date('%c',s_v.ts) })
        end
    end
    return make_dataset(header,rows)
end

function show_process_help()
    local dataset = {}
    local header = { 'command',  'description' }
    local rows = {
        {'SELECT HELP',                 'This command.'},
        {'SHOW PROXY PROCESSLIST',      'Show all connections and their true IP Address.'},
    }
    return make_dataset(header,rows)
end

function dump_process_table()
    proxy.global.initialize_process_table()
    print('current contents of process table')
    for t_i, t_v in pairs (proxy.global.process) do
        print ('session id: ', t_i)
        for s_i, s_v in pairs ( t_v ) do
            print ( '\t', s_i, s_v.ip, s_v.ts )
        end
    end
    print ('---END PROCESS TABLE---')
end

--[[    Help

we use a simple string-match to split commands are word-boundaries

mysql> show querycounter

is split into
command = "show"
option  = "querycounter"

spaces are ignored, the case has to be as is.

mysql> show myerror

returns a error-packet

--]]
```

The script works in combination with a main proxy script, `reporter.lua`:

```

--[[

   Copyright 2008, 2010, Oracle and/or its affiliates. All rights reserved.

   This program is free software; you can redistribute it and/or modify
   it under the terms of the GNU General Public License as published by
   the Free Software Foundation; version 2 of the License.

   This program is distributed in the hope that it will be useful,
   but WITHOUT ANY WARRANTY; without even the implied warranty of
   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
   GNU General Public License for more details.

   You should have received a copy of the GNU General Public License
   along with this program; if not, write to the Free Software
   Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

--]]

-- reporter.lua

--[[

    See http://www.chriscalender.com/?p=41
    (Thanks to Chris Calender)
    See http://datacharmer.blogspot.com/2009/01/mysql-proxy-is-back.html
    (Thanks Giuseppe Maxia)

--]]
```

```
proxy.global.query_counter = proxy.global.query_counter or 0

function proxy.global.initialize_process_table()
    if proxy.global.process == nil then
        proxy.global.process = {}
    end
    if proxy.global.process[proxy.connection.server.thread_id] == nil then
        proxy.global.process[proxy.connection.server.thread_id] = {}
    end
end

function read_auth_result( auth )
    local state = auth.packet:byte()
    if state == proxy.MYSQLD_PACKET_OK then
        proxy.global.initialize_process_table()
        table.insert( proxy.global.process[proxy.connection.server.thread_id],
            { ip = proxy.connection.client.src.name, ts = os.time() } )
    end
end

function disconnect_client()
    local connection_id = proxy.connection.server.thread_id
    if connection_id then
        -- client has disconnected, set this to nil
        proxy.global.process[connection_id] = nil
    end
end


---
-- read_query() can return a resultset
--
-- You can use read_query() to return a result-set.
--
-- @param packet the mysql-packet sent by the client
--
-- @return
--   * nothing to pass on the packet as is,
--   * proxy.PROXY_SEND_QUERY to send the queries from the proxy.queries queue
--   * proxy.PROXY_SEND_RESULT to send your own result-set
--
function read_query( packet )
        -- a new query came in in this connection
        -- using proxy.global.* to make it available to the admin plugin
        proxy.global.query_counter = proxy.global.query_counter + 1

end
```

To use the script, save the first script to a file (`admin.lua` in the following example) and the other to `reporter.lua`, then run `mysql-proxy` specifying the admin script and a backend MySQL server:

```
shell> mysql-proxy --admin-lua-script=admin.lua --admin-password=password \ »
     --admin-username=root --proxy-backend-addresses=127.0.0.1:3306 -proxy-lua-script=reporter.lua
```

In a different window, connect to the MySQL server through the proxy:

```
shell> mysql --user=root --password=password --port=4040
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1798669
Server version: 5.0.70-log Gentoo Linux mysql-5.0.70-r1

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql>
```

In another different window, connect to the `mysql-proxy` admin service using the specified user name and password:

```
shell> mysql --user=root --password=password --port=4041 --host=localhost
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1
Server version: 5.0.99-agent-admin

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql>
```

To monitor the status of the proxy, ask for a list of the current active processes:

```
mysql> show proxy processlist;
+---------+--------------------+-------------------------+
| Id      | IP Address         | Time                    |
+---------+--------------------+-------------------------+
| 1798669 | 192.168.0.112:52592 | Wed Jan 20 16:58:00 2010 |
+---------+--------------------+-------------------------+
1 row in set (0.00 sec)

mysql>
```

For more information on the example, see MySQL Proxy Admin Example.

15.7.6 MySQL Proxy FAQ

Questions

- 16.7.6.1: [2130] In load balancing, how can I separate reads from writes?
- 16.7.6.2: [2131] How do I use a socket with MySQL Proxy? Proxy change logs mention that support for UNIX sockets has been added.
- 16.7.6.3: [2131] Can I use MySQL Proxy with all versions of MySQL?
- 16.7.6.4: [2131] Can I run MySQL Proxy as a daemon?
- 16.7.6.5: [2131] Do proxy applications run on a separate server? If not, what is the overhead incurred by Proxy on the DB server side?
- 16.7.6.6: [2131] With load balancing, what happens to transactions? Are all queries sent to the same server?
- 16.7.6.7: [2131] Is it possible to use MySQL Proxy with updating a Lucene index (or Solr) by making TCP calls to that server to update?
- 16.7.6.8: [2131] Is the system context switch expensive, how much overhead does the Lua script add?
- 16.7.6.9: [2131] How much latency does a proxy add to a connection?
- 16.7.6.10: [2131] Do you have to make one large script and call it at proxy startup, can I change scripts without stopping and restarting (interrupting) the proxy?
- 16.7.6.11: [2131] If MySQL Proxy has to live on same machine as MySQL, are there any tuning considerations to ensure both perform optimally?
- 16.7.6.12: [2132] I currently use SQL Relay for efficient connection pooling with a number of Apache processes connecting to a MySQL server. Can MySQL Proxy currently accomplish this? My goal is to minimize connection latency while keeping temporary tables available.

- 16.7.6.13: [2132] Are these reserved function names (for example, `error_result()`) that get automatically called?
- 16.7.6.14: [2132] As the script is re-read by MySQL Proxy, does it cache this or is it looking at the file system with each request?
- 16.7.6.15: [2132] Given that there is a `connect_server()` function, can a Lua script link up with multiple servers?
- 16.7.6.16: [2132] Is the MySQL Proxy an API?
- 16.7.6.17: [2132] The global namespace variable example with quotas does not persist after a reboot, is that correct?
- 16.7.6.18: [2132] Can MySQL Proxy handle SSL connections?
- 16.7.6.19: [2132] Could MySQL Proxy be used to capture passwords?
- 16.7.6.20: [2132] Are there tools for isolating problems? How can someone figure out whether a problem is in the client, the database, or the proxy?
- 16.7.6.21: [2132] Is MySQL Proxy similar to what is provided by Java connection pools?
- 16.7.6.22: [2133] So authentication with connection pooling has to be done at every connection? What is the authentication latency?
- 16.7.6.23: [2133] If you have multiple databases on the same box, can you use proxy to connect to databases on default port 3306?
- 16.7.6.24: [2133] What about caching the authorization information so clients connecting are given back-end connections that were established with identical authorization information, thus saving a few more round trips?
- 16.7.6.25: [2133] Is there any big web site using MySQL Proxy? For what purpose and what transaction rate have they achieved?
- 16.7.6.26: [2133] How does MySQL Proxy compare to DBSlayer?
- 16.7.6.27: [2133] I tried using MySQL Proxy without any Lua script to try a round-robin type load balancing. In this case, if the first database in the list is down, MySQL Proxy would not connect the client to the second database in the list.
- 16.7.6.28: [2133] Is it “safe” to use `LuaSocket` with proxy scripts?
- 16.7.6.29: [2133] How different is MySQL Proxy from DBCP (Database connection pooling) for Apache in terms of connection pooling?
- 16.7.6.30: [2133] MySQL Proxy can handle about 5000 connections, what is the limit on a MySQL server?
- 16.7.6.31: [2133] Would the Java-only connection pooling solution work for multiple web servers? With this, I would assume that you can pool across many web servers at once?

Questions and Answers

16.7.6.1: In load balancing, how can I separate reads from writes?

There is no automatic separation of queries that perform reads or writes to the different backend servers. However, you can specify to `mysql-proxy` that one or more of the “backend” MySQL servers are read only.

```
shell> mysql-proxy \
--proxy-backend-addresses=10.0.1.2:3306 \
--proxy-read-only-backend-addresses=10.0.1.3:3306 &
```

16.7.6.2: How do I use a socket with MySQL Proxy? Proxy change logs mention that support for UNIX sockets has been added.

Specify the path to the socket:

```
--proxy-backend-addresses=/path/to/socket
```

16.7.6.3: Can I use MySQL Proxy with all versions of MySQL?

MySQL Proxy is designed to work with MySQL 5.0 or higher, and supports the MySQL network protocol for 5.0 and higher.

16.7.6.4: Can I run MySQL Proxy as a daemon?

Use the `--daemon` option. To keep track of the process ID, the daemon can be started with the `--pid-file=file` option to save the PID to a known file name. On version 0.5.x, the Proxy cannot be started natively as a daemon.

16.7.6.5: Do proxy applications run on a separate server? If not, what is the overhead incurred by Proxy on the DB server side?

You can run the proxy on the application server, on its own box, or on the DB-server depending on the use case.

16.7.6.6: With load balancing, what happens to transactions? Are all queries sent to the same server?

Without any special customization the whole connection is sent to the same server. That keeps the whole connection state intact.

16.7.6.7: Is it possible to use MySQL Proxy with updating a Lucene index (or Solr) by making TCP calls to that server to update?

Yes, but it is not advised for now.

16.7.6.8: Is the system context switch expensive, how much overhead does the Lua script add?

Lua is fast and the overhead should be small enough for most applications. The raw packet overhead is around 400 microseconds.

16.7.6.9: How much latency does a proxy add to a connection?

In the range of 400 microseconds per request.

16.7.6.10: Do you have to make one large script and call it at proxy startup, can I change scripts without stopping and restarting (interrupting) the proxy?

You can just change the script and the proxy will reload it when a client connects.

16.7.6.11: If MySQL Proxy has to live on same machine as MySQL, are there any tuning considerations to ensure both perform optimally?

MySQL Proxy can live on any box: application, database, or its own box. MySQL Proxy uses comparatively little CPU or RAM, with negligible additional requirements or overhead.

16.7.6.12: I currently use SQL Relay for efficient connection pooling with a number of Apache processes connecting to a MySQL server. Can MySQL Proxy currently accomplish this? My goal is to minimize connection latency while keeping temporary tables available.

Yes.

16.7.6.13: Are these reserved function names (for example, `error_result()`) that get automatically called?

Only functions and values starting with `proxy.*` are provided by the proxy. All others are user provided.

16.7.6.14: As the script is re-read by MySQL Proxy, does it cache this or is it looking at the file system with each request?

It looks for the script at client-connect and reads it if it has changed, otherwise it uses the cached version.

16.7.6.15: Given that there is a `connect_server()` function, can a Lua script link up with multiple servers?

MySQL Proxy provides some tutorials in the source package; one is `examples/tutorial-keepalive.lua`.

16.7.6.16: Is the MySQL Proxy an API?

No, MySQL Proxy is an application that forwards packets from a client to a server using the MySQL network protocol. The MySQL Proxy provides a API allowing you to change its behavior.

16.7.6.17: The global namespace variable example with quotas does not persist after a reboot, is that correct?

Yes. If you restart the proxy, you lose the results, unless you save them in a file.

16.7.6.18: Can MySQL Proxy handle SSL connections?

No, being the man-in-the-middle, Proxy cannot handle encrypted sessions because it cannot share the SSL information.

16.7.6.19: Could MySQL Proxy be used to capture passwords?

The MySQL network protocol does not allow passwords to be sent in cleartext, all you could capture is the encrypted version.

16.7.6.20: Are there tools for isolating problems? How can someone figure out whether a problem is in the client, the database, or the proxy?

You can set a debug script in the proxy, which is an exceptionally good tool for this purpose. You can see very clearly which component is causing the problem, if you set the right breakpoints.

16.7.6.21: Is MySQL Proxy similar to what is provided by Java connection pools?

Yes and no. Java connection pools are specific to Java applications, MySQL Proxy works with any client API that talks the MySQL network protocol. Also, connection pools do not provide any functionality for intelligently examining the network packets and modifying the contents.

16.7.6.22: So authentication with connection pooling has to be done at every connection? What is the authentication latency?

You can skip the round-trip and use the connection as it was added to the pool. As long as the application cleans up the temporary tables it used. The overhead is (as always) around 400 microseconds.

16.7.6.23: If you have multiple databases on the same box, can you use proxy to connect to databases on default port 3306?

Yes, MySQL Proxy can listen on any port, provided that none of the MySQL servers are listening on the same port.

16.7.6.24: What about caching the authorization information so clients connecting are given back-end connections that were established with identical authorization information, thus saving a few more round trips?

There is an `--proxy-pool-no-change-user` option that provides this functionality.

16.7.6.25: Is there any big web site using MySQL Proxy? For what purpose and what transaction rate have they achieved?

Yes, gaiaonline. They have tested MySQL Proxy and seen it handle 2400 queries per second through the proxy.

16.7.6.26: How does MySQL Proxy compare to DBSlayer?

DBSlayer is a REST->MySQL tool, MySQL Proxy is transparent to your application. No change to the application is needed.

16.7.6.27: I tried using MySQL Proxy without any Lua script to try a round-robin type load balancing. In this case, if the first database in the list is down, MySQL Proxy would not connect the client to the second database in the list.

This issue is fixed in version 0.7.0.

16.7.6.28: Is it “safe” to use `LuaSocket` with proxy scripts?

You can, but it is not advised because it may block.

16.7.6.29: How different is MySQL Proxy from DBCP (Database connection pooling) for Apache in terms of connection pooling?

Connection Pooling is just one use case of the MySQL Proxy. You can use it for a lot more and it works in cases where you cannot use DBCP (for example, if you do not have Java).

16.7.6.30: MySQL Proxy can handle about 5000 connections, what is the limit on a MySQL server?

The server limit is given by the value of the `max_connections` system variable. The default value is version dependent.

16.7.6.31: Would the Java-only connection pooling solution work for multiple web servers? With this, I would assume that you can pool across many web servers at once?

Yes. But you can also start one proxy on each application server to get a similar behavior as you have it already.

Chapter 16 Replication

Table of Contents

Replication enables data from one MySQL database server (the master) to be replicated to one or more MySQL database servers (the slaves). Replication is asynchronous by default - slaves need not to connected permanently to receive updates from the master. This means that updates can occur over long-distance connections and even over temporary or intermittent connections such as a dial-up service. Depending on the configuration, you can replicate all databases, selected databases, or even selected tables within a database.

For answers to some questions often asked by those who are new to MySQL Replication, see Section B.13, “MySQL 5.7 FAQ: Replication”.

The target uses for replication in MySQL include:

- Scale-out solutions - spreading the load among multiple slaves to improve performance. In this environment, all writes and updates must take place on the master server. Reads, however, may take place on one or more slaves. This model can improve the performance of writes (since the master is dedicated to updates), while dramatically increasing read speed across an increasing number of slaves.

- Data security - because data is replicated to the slave, and the slave can pause the replication process, it is possible to run backup services on the slave without corrupting the corresponding master data.

- Analytics - live data can be created on the master, while the analysis of the information can take place on the slave without affecting the performance of the master.

- Long-distance data distribution - if a branch office would like to work with a copy of your main data, you can use replication to create a local copy of the data for their use without requiring permanent access to the master.

Replication in MySQL features support for one-way, asynchronous replication, in which one server acts as the master, while one or more other servers act as slaves. This is in contrast to the *synchronous* replication which is a characteristic of MySQL Cluster (see MySQL Cluster NDB 7.2). In MySQL 5.7, an interface to semisynchronous replication is supported in addition to the built-in asynchronous replication. With semisynchronous replication, a commit performed on the master side blocks before returning to the session that performed the transaction until at least one slave acknowledges that it has received and logged the events for the transaction. See Section 16.3.8, “Semisynchronous Replication” MySQL 5.7 also supports delayed replication such that a slave server deliberately lags behind the master by at least a specified amount of time. See Section 16.3.9, “Delayed Replication”.

There are a number of solutions available for setting up replication between two servers, but the best method to use depends on the presence of data and the engine types you are using. For more information on the available options, see Section 16.1.1, “How to Set Up Replication”.

There are two core types of replication format, Statement Based Replication (SBR), which replicates entire SQL statements, and Row Based Replication (RBR), which replicates only the changed rows. You may also use a third variety, Mixed Based Replication (MBR). For more information on the different replication formats, see Section 16.1.2, “Replication Formats”. In MySQL 5.7, statement-based format is the default.

MySQL 5.7 supports transactional replication based on *global transaction identifiers* (GTIDs). When using this type of replication, it is not necessary to work directly with log files or positions within these files, which greatly simplifies many common replication tasks. Because replication using GTIDs is entirely transactional, consistency between master and slave is guaranteed as long as all transactions committed on the master have also been applied on the slave. For more information about GTIDs and GTID-based replication, see Section 16.1.3, “Replication with Global Transaction Identifiers”.

Replication is controlled through a number of different options and variables. These control the core operation of the replication, timeouts, and the databases and filters that can be applied on databases and tables. For more information on the available options, see Section 16.1.4, “Replication and Binary Logging Options and Variables”.

You can use replication to solve a number of different problems, including problems with performance, supporting the backup of different databases, and as part of a larger solution to alleviate system failures. For information on how to address these issues, see Section 16.3, “Replication Solutions”.

For notes and tips on how different data types and statements are treated during replication, including details of replication features, version compatibility, upgrades, and problems and their resolution, including an FAQ, see Section 16.4, “Replication Notes and Tips”.

For detailed information on the implementation of replication, how replication works, the process and contents of the binary log, background threads and the rules used to decide how statements are recorded and replication, see Section 16.2, “Replication Implementation”.

16.1 Replication Configuration

Replication between servers in MySQL is based on the binary logging mechanism. The MySQL instance operating as the master (the source of the database changes) writes updates and changes as “events” to the binary log. The information in the binary log is stored in different logging formats according to the database changes being recorded. Slaves are configured to read the binary log from the master and to execute the events in the binary log on the slave's local database.

The master is “dumb” in this scenario. Once binary logging has been enabled, all statements are recorded in the binary log. Each slave receives a copy of the entire contents of the binary log. It is the responsibility of the slave to decide which statements in the binary log should be executed; you cannot configure the master to log only certain events. If you do not specify otherwise, all events in the master binary log are executed on the slave. If required, you can configure the slave to process only events that apply to particular databases or tables.

Each slave keeps a record of the binary log coordinates: The file name and position within the file that it has read and processed from the master. This means that multiple slaves can be connected to the master and executing different parts of the same binary log. Because the slaves control this process, individual slaves can be connected and disconnected from the server without affecting the master's operation. Also, because each slave remembers the position within the binary log, it is possible for slaves to be disconnected, reconnect and then “catch up” by continuing from the recorded position.

Both the master and each slave must be configured with a unique ID (using the `server-id` [2162] option). In addition, each slave must be configured with information about the master host name, log file name, and position within that file. These details can be controlled from within a MySQL session using the `CHANGE MASTER TO` statement on the slave. The details are stored within the slave's master info repository, which can be either a file or a table (see Section 16.2.2, “Replication Relay and Status Logs”).

This section describes the setup and configuration required for a replication environment, including step-by-step instructions for creating a new replication environment. The major components of this section are:

- For a guide to setting up two or more servers for replication, Section 16.1.1, “How to Set Up Replication”, deals with the configuration of the systems and provides methods for copying data between the master and slaves.

- Events in the binary log are recorded using a number of formats. These are referred to as statement-based replication (SBR) or row-based replication (RBR). A third type, mixed-format replication (MIXED), uses SBR or RBR replication automatically to take advantage of the benefits of both SBR and RBR formats when appropriate. The different formats are discussed in Section 16.1.2, “Replication Formats”.

- Detailed information on the different configuration options and variables that apply to replication is provided in Section 16.1.4, “Replication and Binary Logging Options and Variables”.

- Once started, the replication process should require little administration or monitoring. However, for advice on common tasks that you may want to execute, see Section 16.1.5, “Common Replication Administration Tasks”.

16.1.1 How to Set Up Replication

This section describes how to set up complete replication of a MySQL server. There are a number of different methods for setting up replication, and the exact method to use depends on how you are setting up replication, and whether you already have data within your master database.

There are some generic tasks that are common to all replication setups:

- On the master, you must enable binary logging and configure a unique server ID. This might require a server restart. See Section 16.1.1.1, “Setting the Replication Master Configuration”.

- On each slave that you want to connect to the master, you must configure a unique server ID. This might require a server restart. See Section 16.1.1.2, “Setting the Replication Slave Configuration”.

- You may want to create a separate user that will be used by your slaves to authenticate with the master to read the binary log for replication. The step is optional. See Section 16.1.1.3, “Creating a User for Replication”.

- Before creating a data snapshot or starting the replication process, you should record the position of the binary log on the master. You will need this information when configuring the slave so that the slave knows where within the binary log to start executing events. See Section 16.1.1.4, “Obtaining the Replication Master Binary Log Coordinates”.
- If you already have data on your master and you want to use it to synchronize your slave, you will need to create a data snapshot. You can create a snapshot using `mysqldump` (see Section 16.1.1.5, “Creating a Data Snapshot Using `mysqldump`”) or by copying the data files directly (see Section 16.1.1.6, “Creating a Data Snapshot Using Raw Data Files”).
- You will need to configure the slave with settings for connecting to the master, such as the host name, login credentials, and binary log file name and position. See Section 16.1.1.10, “Setting the Master Configuration on the Slave”.

Once you have configured the basic options, you will need to follow the instructions for your replication setup. A number of alternatives are provided:

- If you are establishing a new MySQL master and one or more slaves, you need only set up the configuration, as you have no data to exchange. For guidance on setting up replication in this situation, see Section 16.1.1.7, “Setting Up Replication with New Master and Slaves”.
- If you are already running a MySQL server, and therefore already have data that must be transferred to your slaves before replication starts, have not previously configured the binary log and are able to shut down your MySQL server for a short period during the process, see Section 16.1.1.8, “Setting Up Replication with Existing Data”.
- If you are adding slaves to an existing replication environment, you can set up the slaves without affecting the master. See Section 16.1.1.9, “Introducing Additional Slaves to an Existing Replication Environment”.

If you will be administering MySQL replication servers, we suggest that you read this entire chapter through and try all statements mentioned in Section 13.4.1, “SQL Statements for Controlling Master Servers”, and Section 13.4.2, “SQL Statements for Controlling Slave Servers”. You should also familiarize yourself with the replication startup options described in Section 16.1.4, “Replication and Binary Logging Options and Variables”.

Note

Note that certain steps within the setup process require the `SUPER` privilege. If you do not have this privilege, it might not be possible to enable replication.

16.1.1.1 Setting the Replication Master Configuration

On a replication master, you must enable binary logging and establish a unique server ID. If this has not already been done, this part of master setup requires a server restart.

Binary logging *must* be enabled on the master because the binary log is the basis for sending data changes from the master to its slaves. If binary logging is not enabled, replication will not be possible.

Each server within a replication group must be configured with a unique server ID. This ID is used to identify individual servers within the group, and must be a positive integer between 1 and (2^{32})–1. How you organize and select the numbers is entirely up to you.

To configure the binary log and server ID options, you will need to shut down your MySQL server and edit the `my.cnf` or `my.ini` file. Add the following options to the configuration file within the `[mysqld]` section. If these options already exist, but are commented out, uncomment the options and alter them according to your needs. For example, to enable binary logging using a log file name prefix of `mysql-bin`, and configure a server ID of 1, use these lines:

```
[mysqld]
log-bin=mysql-bin
server-id=1
```

After making the changes, restart the server.

Note

If you omit server-id [2162] (or set it explicitly to its default value of 0), a master refuses connections from all slaves.

Note

For the greatest possible durability and consistency in a replication setup using InnoDB with transactions, you should use innodb_flush_log_at_trx_commit=1 and sync_binlog=1 in the master my.cnf file.

Note

Ensure that the skip-networking option is not enabled on your replication master. If networking has been disabled, your slave will not able to communicate with the master and replication will fail.

16.1.1.2 Setting the Replication Slave Configuration

On a replication slave, you must establish a unique server ID. If this has not already been done, this part of slave setup requires a server restart.

If the slave server ID is not already set, or the current value conflicts with the value that you have chosen for the master server, you should shut down your slave server and edit the configuration to specify a unique server ID. For example:

```
[mysqld]
server-id=2
```

After making the changes, restart the server.

If you are setting up multiple slaves, each one must have a unique server-id [2162] value that differs from that of the master and from each of the other slaves. Think of server-id [2162] values as something similar to IP addresses: These IDs uniquely identify each server instance in the community of replication partners.

Note

If you omit server-id [2162] (or set it explicitly to its default value of 0), a slave refuses to connect to a master.

You do not have to enable binary logging on the slave for replication to be enabled. However, if you enable binary logging on the slave, you can use the binary log for data backups and crash recovery on the slave, and also use the slave as part of a more complex replication topology (for example, where the slave acts as a master to other slaves).

16.1.1.3 Creating a User for Replication

Each slave must connect to the master using a MySQL user name and password, so there must be a user account on the master that the slave can use to connect. Any account can be used for this operation, providing it has been granted the REPLICATION SLAVE privilege. You may wish to create a different account for each slave, or connect to the master using the same account for each slave.

You need not create an account specifically for replication. owever, you should be aware that the user name and password are stored in plain text in the master info repository file or table (see Section 16.2.2.2, “Slave Status Logs”). Therefore, you may want to create a separate account that has privileges only for the replication process, to minimize the possibility of compromise to other accounts.

To create a new account, use CREATE USER. To grant this account the privileges required for replication, use the GRANT statement. If you create an account solely for the purposes of replication, that account needs only the REPLICATION SLAVE privilege. For example, to set up a new user, repl, that can connect for replication from any host within the mydomain.com domain, issue these statements on the master:

```
mysql> CREATE USER 'repl'@'%.mydomain.com' IDENTIFIED BY 'slavepass';
mysql> GRANT REPLICATION SLAVE ON *.* TO 'repl'@'%.mydomain.com';
```

See Section 13.7.1, “Account Management Statements”, for more information on statements for manipulation of user accounts.

16.1.1.4 Obtaining the Replication Master Binary Log Coordinates

To configure replication on the slave you must determine the master's current coordinates within its binary log. You will need this information so that when the slave starts the replication process, it is able to start processing events from the binary log at the correct point.

If you have existing data on your master that you want to synchronize on your slaves before starting the replication process, you must stop processing statements on the master, and then obtain its current binary log coordinates and dump its data, before permitting the master to continue executing statements. If you do not stop the execution of statements, the data dump and the master status information that you use will not match and you will end up with inconsistent or corrupted databases on the slaves.

To obtain the master binary log coordinates, follow these steps:

1. Start a session on the master by connecting to it with the command-line client, and flush all tables and block write statements by executing the FLUSH TABLES WITH READ LOCK statement:

   ```
   mysql> FLUSH TABLES WITH READ LOCK;
   ```

   For InnoDB tables, note that FLUSH TABLES WITH READ LOCK also blocks COMMIT operations.

   Warning

   Leave the client from which you issued the FLUSH TABLES statement running so that the read lock remains in effect. If you exit the client, the lock is released.

2. In a different session on the master, use the SHOW MASTER STATUS statement to determine the current binary log file name and position:

   ```
   mysql > SHOW MASTER STATUS;
   +------------------+----------+--------------+------------------+
   | File             | Position | Binlog_Do_DB | Binlog_Ignore_DB |
   +------------------+----------+--------------+------------------+
   | mysql-bin.000003 | 73       | test         | manual,mysql     |
   +------------------+----------+--------------+------------------+
   ```

The `File` column shows the name of the log file and `Position` shows the position within the file. In this example, the binary log file is `mysql-bin.000003` and the position is 73. Record these values. You need them later when you are setting up the slave. They represent the replication coordinates at which the slave should begin processing new updates from the master.

If the master has been running previously without binary logging enabled, the log file name and position values displayed by `SHOW MASTER STATUS` or `mysqldump --master-data` will be empty. In that case, the values that you need to use later when specifying the slave's log file and position are the empty string (`''`) and `4`.

You now have the information you need to enable the slave to start reading from the binary log in the correct place to start replication.

If you have existing data that needs be to synchronized with the slave before you start replication, leave the client running so that the lock remains in place and then proceed to Section 16.1.1.5, “Creating a Data Snapshot Using `mysqldump`”, or Section 16.1.1.6, “Creating a Data Snapshot Using Raw Data Files”. The idea here is to prevent any further changes so that the data copied to the slaves is in synchrony with the master.

If you are setting up a brand new master and slave replication group, you can exit the first session to release the read lock.

16.1.1.5 Creating a Data Snapshot Using `mysqldump`

One way to create a snapshot of the data in an existing master database is to use the `mysqldump` tool to create a dump of all the databases you want to replicate. Once the data dump has been completed, you then import this data into the slave before starting the replication process.

The example shown here dumps all databases to a file named `dbdump.db`, and includes the `--master-data` option which automatically appends the `CHANGE MASTER TO` statement required on the slave to start the replication process:

```
shell> mysqldump --all-databases --master-data > dbdump.db
```

If you do not use `--master-data`, then it is necessary to lock all tables in a separate session manually (using `FLUSH TABLES WITH READ LOCK`) prior to running `mysqldump`, then exiting or running UNLOCK TABLES from the second session to release the locks. You must also obtain binary log position information matching the snapshot, using `SHOW MASTER STATUS`, and use this to issue the appropriate `CHANGE MASTER TO` statement when starting the slave.

When choosing databases to include in the dump, remember that you need to filter out databases on each slave that you do not want to include in the replication process.

To import the data, either copy the dump file to the slave, or access the file from the master when connecting remotely to the slave.

16.1.1.6 Creating a Data Snapshot Using Raw Data Files

If your database is large, copying the raw data files can be more efficient than using `mysqldump` and importing the file on each slave. This technique skips the overhead of updating indexes as the `INSERT` statements are replayed.

Using this method with tables in storage engines with complex caching or logging algorithms requires extra steps to produce a perfect “point in time” snapshot: the initial copy command might leave out cache information and logging updates, even if you have acquired a global read lock. How the storage engine responds to this depends on its crash recovery abilities.

This method also does not work reliably if the master and slave have different values for ft_stopword_file, ft_min_word_len, or ft_max_word_len and you are copying tables having full-text indexes.

If you use InnoDB tables, you can use the mysqlbackup command from the MySQL Enterprise Backup component to produce a consistent snapshot. This command records the log name and offset corresponding to the snapshot to be later used on the slave. MySQL Enterprise Backup is a commercial product that is included as part of a MySQL Enterprise subscription. See Section 23.2, “MySQL Enterprise Backup” for detailed information.

Otherwise, use the cold backup technique to obtain a reliable binary snapshot of InnoDB tables: copy all data files after doing a slow shutdown of the MySQL Server.

To create a raw data snapshot of MyISAM tables, you can use standard copy tools such as cp or copy, a remote copy tool such as scp or rsync, an archiving tool such as zip or tar, or a file system snapshot tool such as dump, providing that your MySQL data files exist on a single file system. If you are replicating only certain databases, copy only those files that relate to those tables. (For InnoDB, all tables in all databases are stored in the system tablespace files, unless you have the innodb_file_per_table option enabled.)

You might want to specifically exclude the following files from your archive:

- Files relating to the mysql database.
- The master info repository file, if used (see Section 16.2.2, “Replication Relay and Status Logs”).
- The master's binary log files.
- Any relay log files.

To get the most consistent results with a raw data snapshot, shut down the master server during the process, as follows:

1. Acquire a read lock and get the master's status. See Section 16.1.1.4, “Obtaining the Replication Master Binary Log Coordinates”.
2. In a separate session, shut down the master server:

   ```
   shell> mysqladmin shutdown
   ```

3. Make a copy of the MySQL data files. The following examples show common ways to do this. You need to choose only one of them:

   ```
   shell> tar cf /tmp/db.tar ./data
   shell> zip -r /tmp/db.zip ./data
   shell> rsync --recursive ./data /tmp/dbdata
   ```

4. Restart the master server.

If you are not using InnoDB tables, you can get a snapshot of the system from a master without shutting down the server as described in the following steps:

1. Acquire a read lock and get the master's status. See Section 16.1.1.4, “Obtaining the Replication Master Binary Log Coordinates”.
2. Make a copy of the MySQL data files. The following examples show common ways to do this. You need to choose only one of them:

```
shell> tar cf /tmp/db.tar ./data
shell> zip -r /tmp/db.zip ./data
shell> rsync --recursive ./data /tmp/dbdata
```

3. In the client where you acquired the read lock, release the lock:

   ```
   mysql> UNLOCK TABLES;
   ```

Once you have created the archive or copy of the database, copy the files to each slave before starting the slave replication process.

16.1.1.7 Setting Up Replication with New Master and Slaves

The easiest and most straightforward method for setting up replication is to use new master and slave servers.

You can also use this method if you are setting up new servers but have an existing dump of the databases from a different server that you want to load into your replication configuration. By loading the data into a new master, the data will be automatically replicated to the slaves.

To set up replication between a new master and slave:

1. Configure the MySQL master with the necessary configuration properties. See Section 16.1.1.1, “Setting the Replication Master Configuration”.

2. Start up the MySQL master.

3. Set up a user. See Section 16.1.1.3, “Creating a User for Replication”.

4. Obtain the master status information. See Section 16.1.1.4, “Obtaining the Replication Master Binary Log Coordinates”.

5. On the master, release the read lock:

   ```
   mysql> UNLOCK TABLES;
   ```

6. On the slave, edit the MySQL configuration. See Section 16.1.1.2, “Setting the Replication Slave Configuration”.

7. Start up the MySQL slave.

8. Execute a `CHANGE MASTER TO` statement to set the master replication server configuration. See Section 16.1.1.10, “Setting the Master Configuration on the Slave”.

Perform the slave setup steps on each slave.

Because there is no data to load or exchange on a new server configuration you do not need to copy or import any information.

If you are setting up a new replication environment using the data from a different existing database server, you will now need to run the dump file generated from that server on the new master. The database updates will automatically be propagated to the slaves:

```
shell> mysql -h master < fulldb.dump
```

16.1.1.8 Setting Up Replication with Existing Data

When setting up replication with existing data, you will need to decide how best to get the data from the master to the slave before starting the replication service.

The basic process for setting up replication with existing data is as follows:

1. With the MySQL master running, create a user to be used by the slave when connecting to the master during replication. See Section 16.1.1.3, “Creating a User for Replication”.

2. If you have not already configured the `server-id` [2162] and enabled binary logging on the master server, you will need to shut it down to configure these options. See Section 16.1.1.1, “Setting the Replication Master Configuration”.

   If you have to shut down your master server, this is a good opportunity to take a snapshot of its databases. You should obtain the master status (see Section 16.1.1.4, “Obtaining the Replication Master Binary Log Coordinates”) before taking down the master, updating the configuration and taking a snapshot. For information on how to create a snapshot using raw data files, see Section 16.1.1.6, “Creating a Data Snapshot Using Raw Data Files”.

3. If your master server is already correctly configured, obtain its status (see Section 16.1.1.4, “Obtaining the Replication Master Binary Log Coordinates”) and then use `mysqldump` to take a snapshot (see Section 16.1.1.5, “Creating a Data Snapshot Using `mysqldump`”) or take a raw snapshot of the live server using the guide in Section 16.1.1.6, “Creating a Data Snapshot Using Raw Data Files”.

4. Update the configuration of the slave. See Section 16.1.1.2, “Setting the Replication Slave Configuration”.

5. The next step depends on how you created the snapshot of data on the master.

   If you used `mysqldump`:

   a. Start the slave, using the `--skip-slave-start` option so that replication does not start.

   b. Import the dump file:

      ```
      shell> mysql < fulldb.dump
      ```

   If you created a snapshot using the raw data files:

   a. Extract the data files into your slave data directory. For example:

      ```
      shell> tar xvf dbdump.tar
      ```

      You may need to set permissions and ownership on the files so that the slave server can access and modify them.

   b. Start the slave, using the `--skip-slave-start` option so that replication does not start.

6. Configure the slave with the replication coordinates from the master. This tells the slave the binary log file and position within the file where replication needs to start. Also, configure the slave with the login credentials and host name of the master. For more information on the `CHANGE MASTER TO` statement required, see Section 16.1.1.10, “Setting the Master Configuration on the Slave”.

7. Start the slave threads:

   ```
   mysql> START SLAVE;
   ```

After you have performed this procedure, the slave should connect to the master and catch up on any updates that have occurred since the snapshot was taken.

If you have forgotten to set the `server-id` [2162] option for the master, slaves cannot connect to it.

If you have forgotten to set the server-id [2162] option for the slave, you get the following error in the slave's error log:

```
Warning: You should set server-id to a non-0 value if master_host
is set; we will force server id to 2, but this MySQL server will
not act as a slave.
```

You also find error messages in the slave's error log if it is not able to replicate for any other reason.

The slave uses information stored in its master info repository to keep track of how much of the master's binary log it has processed. The repository can be in the form of files or a table, as determined by the value set for --master-info-repository. When a slave runs with --master-info-repository=FILE, you can find in its data directory two files, named master.info and relay-log.info. If --master-info-repository=TABLE instead, this information is saved in the table master_slave_info in the mysql database. In either case, do *not* remove or edit the files or table unless you know exactly what you are doing and fully understand the implications. Even in that case, it is preferred that you use the CHANGE MASTER TO statement to change replication parameters. The slave can use the values specified in the statement to update the status files automatically. See Section 16.2.2, “Replication Relay and Status Logs”, for more information.

Note

The contents of the master info repository override some of the server options specified on the command line or in my.cnf. See Section 16.1.4, “Replication and Binary Logging Options and Variables”, for more details.

A single snapshot of the master suffices for multiple slaves. To set up additional slaves, use the same master snapshot and follow the slave portion of the procedure just described.

16.1.1.9 Introducing Additional Slaves to an Existing Replication Environment

To add another slave to an existing replication configuration, you can do so without stopping the master. Instead, set up the new slave by making a copy of an existing slave, except that you configure the new slave with a different server-id [2162] value.

To duplicate an existing slave:

1. Shut down the existing slave:

   ```
   shell> mysqladmin shutdown
   ```

2. Copy the data directory from the existing slave to the new slave. You can do this by creating an archive using tar or WinZip, or by performing a direct copy using a tool such as cp or rsync. Ensure that you also copy the log files and relay log files.

   A common problem that is encountered when adding new replication slaves is that the new slave fails with a series of warning and error messages like these:

   ```
   071118 16:44:10 [Warning] Neither --relay-log nor --relay-log-index were used; so
   replication may break when this MySQL server acts as a slave and has his hostname
   changed!! Please use '--relay-log=new_slave_hostname-relay-bin' to avoid this problem.
   071118 16:44:10 [ERROR] Failed to open the relay log './old_slave_hostname-relay-bin.003525'
   (relay_log_pos 22940879)
   071118 16:44:10 [ERROR] Could not find target log during relay log initialization
   071118 16:44:10 [ERROR] Failed to initialize the master info structure
   ```

This is due to the fact that, if the `--relay-log` option is not specified, the relay log files contain the host name as part of their file names. (This is also true of the relay log index file if the `--relay-log-index` option is not used. See Section 16.1.4, “Replication and Binary Logging Options and Variables”, for more information about these options.)

To avoid this problem, use the same value for `--relay-log` on the new slave that was used on the existing slave. (If this option was not set explicitly on the existing slave, use *`existing_slave_hostname`*`-relay-bin`.) If this is not feasible, copy the existing slave's relay log index file to the new slave and set the `--relay-log-index` option on the new slave to match what was used on the existing slave. (If this option was not set explicitly on the existing slave, use *`existing_slave_hostname`*`-relay-bin.index`.) Alternatively—if you have already tried to start the new slave (after following the remaining steps in this section) and have encountered errors like those described previously—then perform the following steps:

a. If you have not already done so, issue a `STOP SLAVE` on the new slave.

   If you have already started the existing slave again, issue a `STOP SLAVE` on the existing slave as well.

b. Copy the contents of the existing slave's relay log index file into the new slave's relay log index file, making sure to overwrite any content already in the file.

c. Proceed with the remaining steps in this section.

3. Copy the master info and relay log info repositories (see Section 16.2.2, “Replication Relay and Status Logs”) from the existing slave to the new slave. These hold the current log coordinates for the master's binary log and the slave's relay log.

4. Start the existing slave.

5. On the new slave, edit the configuration and give the new slave a unique `server-id` [2162] not used by the master or any of the existing slaves.

6. Start the new slave. The slave uses the information in its master info repository to start the replication process.

16.1.1.10 Setting the Master Configuration on the Slave

To set up the slave to communicate with the master for replication, you must tell the slave the necessary connection information. To do this, execute the following statement on the slave, replacing the option values with the actual values relevant to your system:

```
mysql> CHANGE MASTER TO
    ->     MASTER_HOST='master_host_name',
    ->     MASTER_USER='replication_user_name',
    ->     MASTER_PASSWORD='replication_password',
    ->     MASTER_LOG_FILE='recorded_log_file_name',
    ->     MASTER_LOG_POS=recorded_log_position;
```

Note

Replication cannot use Unix socket files. You must be able to connect to the master MySQL server using TCP/IP.

The `CHANGE MASTER TO` statement has other options as well. For example, it is possible to set up secure replication using SSL. For a full list of options, and information about the maximum permissible length for the string-valued options, see Section 13.4.2.1, “`CHANGE MASTER TO` Syntax”.

16.1.2 Replication Formats

Replication works because events written to the binary log are read from the master and then processed on the slave. The events are recorded within the binary log in different formats according to the type of event. The different replication formats used correspond to the binary logging format used when the events were recorded in the master's binary log. The correlation between binary logging formats and the terms used during replication are:

- Replication capabilities in MySQL originally were based on propagation of SQL statements from master to slave. This is called *statement-based replication* (often abbreviated as *SBR*), which corresponds to the standard statement-based binary logging format. In older versions of MySQL (5.1.4 and earlier), binary logging and replication used this format exclusively.
- Row-based binary logging logs changes in individual table rows. When used with MySQL replication, this is known as *row-based replication* (often abbreviated as *RBR*). In row-based replication, the master writes *events* to the binary log that indicate how individual table rows are changed.
- The server can change the binary logging format in real time according to the type of event using *mixed-format logging*.

  When the mixed format is in effect, statement-based logging is used by default, but automatically switches to row-based logging in particular cases as described later. Replication using the mixed format is often referred to as *mixed-based replication* or *mixed-format replication*. For more information, see Section 5.2.4.3, “Mixed Binary Logging Format”.

In MySQL 5.7, statement-based format is the default.

When using `MIXED` format, the binary logging format is determined in part by the storage engine being used and the statement being executed. For more information on mixed-format logging and the rules governing the support of different logging formats, see Section 5.2.4.3, “Mixed Binary Logging Format”.

The logging format in a running MySQL server is controlled by setting the `binlog_format` server system variable. This variable can be set with session or global scope. The rules governing when and how the new setting takes effect are the same as for other MySQL server system variables—setting the variable for the current session lasts only until the end of that session, and the change is not visible to other sessions; setting the variable globally requires a restart of the server to take effect. For more information, see Section 13.7.4, “`SET` Syntax”.

There are conditions under which you cannot change the binary logging format at runtime or doing so causes replication to fail. See Section 5.2.4.2, “Setting The Binary Log Format”.

You must have the `SUPER` privilege to set either the global or session `binlog_format` value.

The statement-based and row-based replication formats have different issues and limitations. For a comparison of their relative advantages and disadvantages, see Section 16.1.2.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”.

With statement-based replication, you may encounter issues with replicating stored routines or triggers. You can avoid these issues by using row-based replication instead. For more information, see Section 18.7, “Binary Logging of Stored Programs”.

16.1.2.1 Advantages and Disadvantages of Statement-Based and Row-Based Replication

Each binary logging format has advantages and disadvantages. For most users, the mixed replication format should provide the best combination of data integrity and performance. If, however, you want to take advantage of the features specific to the statement-based or row-based replication format when performing certain tasks, you can use the information in this section, which provides a summary of their relative advantages and disadvantages, to determine which is best for your needs.

- Advantages of statement-based replication
- Disadvantages of statement-based replication
- Advantages of row-based replication
- Disadvantages of row-based replication

Advantages of statement-based replication

- Proven technology that has existed in MySQL since 3.23.
- Less data written to log files. When updates or deletes affect many rows, this results in *much* less storage space required for log files. This also means that taking and restoring from backups can be accomplished more quickly.
- Log files contain all statements that made any changes, so they can be used to audit the database.

Disadvantages of statement-based replication

- **Statements that are unsafe for SBR.**
  Not all statements which modify data (such as `INSERT` `DELETE`, `UPDATE`, and `REPLACE` statements) can be replicated using statement-based replication. Any nondeterministic behavior is difficult to replicate when using statement-based replication. Examples of such DML (Data Modification Language) statements include the following:

  - A statement that depends on a UDF or stored program that is nondeterministic, since the value returned by such a UDF or stored program or depends on factors other than the parameters supplied to it. (Row-based replication, however, simply replicates the value returned by the UDF or stored program, so its effect on table rows and data is the same on both the master and slave.) See Section 16.4.1.11, “Replication of Invoked Features”, for more information.
  - `DELETE` and `UPDATE` statements that use a `LIMIT` clause without an `ORDER BY` are nondeterministic. See Section 16.4.1.16, “Replication and `LIMIT`”.
  - Statements using any of the following functions cannot be replicated properly using statement-based replication:

    - `LOAD_FILE()`
    - `UUID()`, `UUID_SHORT()`
    - `USER()`
    - `FOUND_ROWS()`
    - `SYSDATE()` (unless both the master and the slave are started with the `--sysdate-is-now` option)
    - `GET_LOCK()`
    - `IS_FREE_LOCK()`
    - `IS_USED_LOCK()`

  - MASTER_POS_WAIT()
  - RAND()
  - RELEASE_LOCK()
  - SLEEP()
  - VERSION()

  However, all other functions are replicated correctly using statement-based replication, including NOW() and so forth.

  For more information, see Section 16.4.1.15, “Replication and System Functions”.

  Statements that cannot be replicated correctly using statement-based replication are logged with a warning like the one shown here:

```
[Warning] Statement is not safe to log in statement format.
```

  A similar warning is also issued to the client in such cases. The client can display it using SHOW WARNINGS.

- INSERT ... SELECT requires a greater number of row-level locks than with row-based replication.
- UPDATE statements that require a table scan (because no index is used in the WHERE clause) must lock a greater number of rows than with row-based replication.
- For InnoDB: An INSERT statement that uses AUTO_INCREMENT blocks other nonconflicting INSERT statements.
- For complex statements, the statement must be evaluated and executed on the slave before the rows are updated or inserted. With row-based replication, the slave only has to modify the affected rows, not execute the full statement.
- If there is an error in evaluation on the slave, particularly when executing complex statements, statement-based replication may slowly increase the margin of error across the affected rows over time. See Section 16.4.1.26, “Slave Errors During Replication”.
- Stored functions execute with the same NOW() value as the calling statement. However, this is not true of stored procedures.
- Deterministic UDFs must be applied on the slaves.
- Table definitions must be (nearly) identical on master and slave. See Section 16.4.1.9, “Replication with Differing Table Definitions on Master and Slave”, for more information.

Advantages of row-based replication

- All changes can be replicated. This is the safest form of replication.

  The mysql database is not replicated. The mysql database is instead seen as a node-specific database. Row-based replication is not supported on tables in this database. Instead, statements that would normally update this information—such as GRANT, REVOKE and the manipulation of triggers, stored routines (including stored procedures), and views—are all replicated to slaves using statement-based replication.

For statements such as CREATE TABLE ... SELECT, a CREATE statement is generated from the table definition and replicated using statement-based format, while the row insertions are replicated using row-based format.

- The technology is the same as in most other database management systems; knowledge about other systems transfers to MySQL.
- Fewer row locks are required on the master, which thus achieves higher concurrency, for the following types of statements:
  - INSERT ... SELECT
  - INSERT statements with AUTO_INCREMENT
  - UPDATE or DELETE statements with WHERE clauses that do not use keys or do not change most of the examined rows.
- Fewer row locks are required on the slave for any INSERT, UPDATE, or DELETE statement.

Disadvantages of row-based replication

- RBR tends to generate more data that must be logged. To replicate a DML statement (such as an UPDATE or DELETE statement), statement-based replication writes only the statement to the binary log. By contrast, row-based replication writes each changed row to the binary log. If the statement changes many rows, row-based replication may write significantly more data to the binary log; this is true even for statements that are rolled back. This also means that taking and restoring from backup can require more time. In addition, the binary log is locked for a longer time to write the data, which may cause concurrency problems.
- Deterministic UDFs that generate large BLOB values take longer to replicate with row-based replication than with statement-based replication. This is because the BLOB column value is logged, rather than the statement generating the data.
- You cannot examine the logs to see what statements were executed, nor can you see on the slave what statements were received from the master and executed.

  However, you can see what data was changed using mysqlbinlog with the options --base64-output=DECODE-ROWS and --verbose.
- For tables using the MyISAM storage engine, a stronger lock is required on the slave for INSERT statements when applying them as row-based events to the binary log than when applying them as statements. This means that concurrent inserts on MyISAM tables are not supported when using row-based replication.

16.1.2.2 Usage of Row-Based Logging and Replication

Major changes in the replication environment and in the behavior of applications can result from using row-based logging (RBL) or row-based replication (RBR) rather than statement-based logging or replication. This section describes a number of issues known to exist when using row-based logging or replication, and discusses some best practices for taking advantage of row-based logging and replication.

For additional information, see Section 16.1.2, “Replication Formats”, and Section 16.1.2.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”.

- **RBL, RBR, and temporary tables.** As noted in Section 16.4.1.22, “Replication and Temporary Tables”, temporary tables are not replicated when using row-based format. When mixed format is in effect, “safe” statements involving temporary tables are logged using statement-based format. For more information, see Section 16.1.2.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”.

Temporary tables are not replicated when using row-based format because there is no need. In addition, because temporary tables can be read only from the thread which created them, there is seldom if ever any benefit obtained from replicating them, even when using statement-based format.

In MySQL 5.7, you can switch from statement-based to row-based binary logging mode even when temporary tables have been created. However, while using the row-based format, the MySQL server cannot determine the logging mode that was in effect when a given temporary table was created. For this reason, the server in such cases logs a `DROP TEMPORARY TABLE IF EXISTS` statement for each temporary table that still exists for a given client session when that session ends. While this means that it is possible that an unnecessary `DROP TEMPORARY TABLE` statement might be logged in some cases, the statement is harmless, and does not cause an error even if the table does not exist, due to the presence of the `IF NOT EXISTS` option.

Nontransactional DML statements involving temporary tablesare allowed when using `binlog_format=ROW`, as long as any nontransactional tables affected by the statements are temporary tables (Bug #14272672).

- **RBL and synchronization of nontransactional tables.** When many rows are affected, the set of changes is split into several events; when the statement commits, all of these events are written to the binary log. When executing on the slave, a table lock is taken on all tables involved, and then the rows are applied in batch mode. (This may or may not be effective, depending on the engine used for the slave's copy of the table.)

- **Latency and binary log size.** Because RBL writes changes for each row to the binary log, its size can increase quite rapidly. In a replication environment, this can significantly increase the time required to make changes on the slave that match those on the master. You should be aware of the potential for this delay in your applications.

- **Reading the binary log.** `mysqlbinlog` displays row-based events in the binary log using the `BINLOG` statement (see Section 13.7.6.1, “`BINLOG` Syntax”). This statement displays an event in printable form, but as a base 64-encoded string the meaning of which is not evident. When invoked with the `--base64-output=DECODE-ROWS` and `--verbose` options, `mysqlbinlog` formats the contents of the binary log in a manner that is easily human readable. This is helpful when binary log events were written in row-based format if you want to read or recover from a replication or database failure using the contents of the binary log. For more information, see Section 4.6.7.2, “`mysqlbinlog` Row Event Display”.

- **Binary log execution errors and `slave_exec_mode`.** If `slave_exec_mode` is `IDEMPOTENT`, a failure to apply changes from RBL because the original row cannot be found does not trigger an error or cause replication to fail. This means that it is possible that updates are not applied on the slave, so that the master and slave are no longer synchronized. Latency issues and use of nontransactional tables with RBR when `slave_exec_mode` is `IDEMPOTENT` can cause the master and slave to diverge even further. For more information about `slave_exec_mode`, see Section 5.1.4, “Server System Variables”.

> **Note**
>
> `slave_exec_mode=IDEMPOTENT` is generally useful only for circular replication or multi-master replication with MySQL Cluster, for which `IDEMPOTENT` is the default value.
>
> For other scenarios, setting `slave_exec_mode` to `STRICT` is normally sufficient; this is the default value for storage engines other than `NDB`.

> The NDBCLUSTER storage engine is currently not supported in MySQL 5.7. MySQL Cluster users wishing to upgrade from MySQL 5.0 should instead migrate to MySQL Cluster NDB 7.1 or later. For more information about MySQL Cluster NDB 7.1, see MySQL Cluster NDB 6.1 - 7.1; for more information about MySQL Cluster NDB 7.2, see MySQL Cluster NDB 7.2.

- **Lack of binary log checksums.** RBL uses no checksums. This means that network, disk, and other errors may not be identified when processing the binary log. To ensure that data is transmitted without network corruption, you may want to consider using SSL, which adds another layer of checksumming, for replication connections. The `CHANGE MASTER TO` statement has options to enable replication over SSL. See also Section 13.4.2.1, "`CHANGE MASTER TO` Syntax", for general information about setting up MySQL with SSL.

- **Filtering based on server ID not supported.** A common practice is to filter out changes on some slaves by using a `WHERE` clause that includes the relation `@@server_id <> id_value` clause with `UPDATE` and `DELETE` statements, a simple example of such a clause being `WHERE @@server_id <> 1`. However, this does not work correctly with row-based logging. If you must use the `server_id` system variable for statement filtering, you must also use `--binlog_format=STATEMENT`.

  In MySQL 5.7, you can do filtering based on server ID by using the `IGNORE_SERVER_IDS` option for the `CHANGE MASTER TO` statement. This option works with the statement-based and row-based logging formats.

- **Database-level replication options.** The effects of the `--replicate-do-db`, `--replicate-ignore-db`, and `--replicate-rewrite-db` options differ considerably depending on whether row-based or statement-based logging is used. Because of this, it is recommended to avoid database-level options and instead use table-level options such as `--replicate-do-table` and `--replicate-ignore-table`. For more information about these options and the impact that your choice of replication format has on how they operate, see Section 16.1.4, "Replication and Binary Logging Options and Variables".

- **RBL, nontransactional tables, and stopped slaves.** When using row-based logging, if the slave server is stopped while a slave thread is updating a nontransactional table, the slave database may reaches an inconsistent state. For this reason, it is recommended that you use a transactional storage engine such as `InnoDB` for all tables replicated using the row-based format.

  Use of `STOP SLAVE` or `STOP SLAVE SQL_THREAD` prior to shutting down the slave MySQL server helps prevent such issues from occurring, and is always recommended regardless of the logging format or storage engines employed.

16.1.2.3 Determination of Safe and Unsafe Statements in Binary Logging

When speaking of the "safeness" of a statement in MySQL Replication, we are referring to whether a statement and its effects can be replicated correctly using statement-based format. If this is true of the statement, we refer to the statement as *safe*; otherwise, we refer to it as *unsafe*.

In general, a statement is safe if it deterministic, and unsafe if it is not. However, certain nondeterministic functions are *not* considered unsafe (see Nondeterministic functions not considered unsafe, later in this section). In addition, statements using results from floating-point math functions—which are hardware-dependent—are always considered unsafe (see Section 16.4.1.12, "Replication and Floating-Point Values").

Handling of safe and unsafe statements. A statement is treated differently depending on whether the statement is considered safe, and with respect to the binary logging format (that is, the current value of `binlog_format`).

- No distinction is made in the treatment of safe and unsafe statements when the binary logging mode is `ROW`.
- If the binary logging format is `MIXED`, statements flagged as unsafe are logged using the row-based format; statements regarded as safe are logged using the statement-based format.
- If the binary logging format is `STATEMENT`, statements flagged as being unsafe generate a warning to this effect. (Safe statements are logged normally.)

Each statement flagged as unsafe generates a warning. Formerly, in cases where a great many such statements were executed on the master, this could lead to very large error log files, sometimes even filling up an entire disk unexpectedly. To guard against this, MySQL 5.7 provides a warning suppression mechanism, which behaves as follows: Whenever the 50 most recent `ER_BINLOG_UNSAFE_STATEMENT` warnings have been generated more than 50 times in any 50-second period, warning suppression is enabled. When activated, this causes such warnings not to be written to the error log; instead, for each 50 warnings of this type, a note `The last warning was repeated N times in last S seconds` is written to the error log. This continues as long as the 50 most recent such warnings were issued in 50 seconds or less; once the rate has decreased below this threshold, the warnings are once again logged normally. Warning suppression has no effect on how the safety of statements for statement-based logging is determined, nor on how warnings are sent to the client (MySQL clients still receive one warning for each such statement).

For more information, see Section 16.1.2, “Replication Formats”.

Statements considered unsafe.
Statements having the following characteristics are considered unsafe:

- **Statements containing system functions that may return a different value on slave.** These functions include `FOUND_ROWS()`, `GET_LOCK()`, `IS_FREE_LOCK()`, `IS_USED_LOCK()`, `LOAD_FILE()`, `MASTER_POS_WAIT()`, `PASSWORD()`, `RAND()`, `RELEASE_LOCK()`, `ROW_COUNT()`, `SESSION_USER()`, `SLEEP()`, `SYSDATE()`, `SYSTEM_USER()`, `USER()`, `UUID()`, and `UUID_SHORT()`.

  Nondeterministic functions not considered unsafe. Although these functions are not deterministic, they are treated as safe for purposes of logging and replication: `CONNECTION_ID()`, `CURDATE()`, `CURRENT_DATE()`, `CURRENT_TIME()`, `CURRENT_TIMESTAMP()`, `CURTIME()`,, `LAST_INSERT_ID()`, `LOCALTIME()`, `LOCALTIMESTAMP()`, `NOW()`, `UNIX_TIMESTAMP()`, `UTC_DATE()`, `UTC_TIME()`, and `UTC_TIMESTAMP()`.

  For more information, see Section 16.4.1.15, “Replication and System Functions”.

- **References to system variables.** Most system variables are not replicated correctly using the statement-based format. For exceptions, see Section 5.2.4.3, “Mixed Binary Logging Format”.

  See Section 16.4.1.34, “Replication and Variables”.

- **UDFs.** Since we have no control over what a UDF does, we must assume that it is executing unsafe statements.

- **Fulltext plugin.** Since this plugin may behave differently on different MySQL servers (and we have no control over this), we must assume that statements depending on it may have different results. For this reason, all statements relying on the fulltext plugin are treated as unsafe in MySQKL 5.7.1 and later. (Bug #11756280, Bug #48183)

- **Trigger or stored program updates a table having an `AUTO_INCREMENT` column.** This is unsafe because the order in which the rows are updated may differ on the master and the slave.

In addition, an INSERT into a table that has a composite primary key containing an AUTO_INCREMENT column that is not the first column of this composite key is unsafe.

For more information, see Section 16.4.1.1, “Replication and AUTO_INCREMENT”.

- **INSERT ... ON DUPLICATE KEY UPDATE statements on tables with multiple primary or unique keys.** When executed against a table that contains more than one primary or unique key, this statement is considered unsafe, being sensitive to the order in which the storage engine checks the keys, which is not deterministic, and on which the choice of rows updated by the MySQL Server depends.

  An INSERT ... ON DUPLICATE KEY UPDATE statement against a table having more than one unique or primary key is marked as unsafe for statement-based replication. (Bug #11765650, Bug #58637)

- **Updates using LIMIT.** The order in which rows are retrieved is not specified.

  See Section 16.4.1.16, “Replication and LIMIT”.

- **Accesses or references log tables.** The contents of the system log table may differ between master and slave.

- **Nontransactional operations after transactional operations.** Within a transaction, allowing any nontransactional reads or writes to execute after any transactional reads or writes is considered unsafe.

  For more information, see Section 16.4.1.31, “Replication and Transactions”.

- **Accesses or references self-logging tables.** All reads and writes to self-logging tables are considered unsafe. Within a transaction, any statement following a read or write to self-logging tables is also considered unsafe.

- **LOAD DATA INFILE statements.** LOAD DATA INFILE is considered unsafe, it causes a warning in statement-based mode, and a switch to row-based format when using mixed-format logging. See Section 16.4.1.17, “Replication and LOAD DATA INFILE”.

For additional information, see Section 16.4.1, “Replication Features and Issues”.

16.1.3 Replication with Global Transaction Identifiers

In this section, we discuss transaction-based replication using *global transaction identifiers* (GTIDs). When using GTIDs, each transaction can be identified and tracked as it is committed on the originating server and applied by any slaves; this means that it is not necessary when using GTIDs to refer to log files or positions within those files when starting a new slave or failing over to a new master, which greatly simplifies these tasks. Because GTID-based replication is completely transaction-based, it is simple to determine whether masters and slaves are consistent; as long as all transactions committed on a master are also committed on a slave, consistency between the two is guaranteed. You can use either statement-based or row-based replication with GTIDs (see Section 16.1.2, “Replication Formats”); however, for best results, we recommend that you use the row-based format.

The next few sections discuss the following topics:

- How GTIDs are defined and created, and how they are represented in the MySQL Server (see Section 16.1.3.1, “GTID Concepts”).

- A general procedure for setting up and starting GTID-based replication (see Section 16.1.3.2, “Setting Up Replication Using GTIDs”).

- Suggested methods for provisioning new replication servers when using GTIDs (see Section 16.1.3.3, “Using GTIDs for Failover and Scaleout”).
- Restrictions and limitations that you should be aware of when using GTID-based replication (see Section 16.1.3.4, “Restrictions on Replication with GTIDs”).

For information about MySQL Server options and variables relating to GTID-based replication, see Section 16.1.4.5, “Global Transaction ID Options and Variables”. See also Section 12.15, “Functions Used with Global Transaction IDs”, which describes SQL functions supported by MySQL 5.7 for use with GTIDs.

16.1.3.1 GTID Concepts

A global transaction identifier (GTID) is a unique identifier created and associated with each transaction when it is committed on the server of origin (master). This identifier is unique not only to the server on which it originated, but is unique across all servers in a given replication setup. There is a 1-to-1 mapping between all transactions and all GTIDs.

A GTID is represented as a pair of coordinates, separated by a colon character (`:`), as shown here:

```
GTID = source_id:transaction_id
```

The *`source_id`* identifies the originating server. Normally, the server's `server_uuid` [2162] is used for this purpose. (It is theoretically possible for it to be determined in a different manner if the source of the transaction is not a MySQL Server instance, but this is currently not supported.) The *`transaction_id`* is a sequence number determined by the order in which the transaction was committed on this server; for example, the first transaction to be committed has `1` as its *`transaction_id`*, and the tenth transaction to be committed on the same originating server is assigned a *`transaction_id`* of `10`. (It is not possible for a transaction to have `0` as a sequence number in a GTID.) Thus, the twenty-third transaction to be committed originally on the server having the UUID `3E11FA47-71CA-11E1-9E33-C80AA9429562` has this GTID:

```
3E11FA47-71CA-11E1-9E33-C80AA9429562:23
```

This format is used to represent GTIDs in the output of statements such as `SHOW SLAVE STATUS` as well as in the binary log. They can also be seen when viewing the log file with `mysqlbinlog --base64-output=DECODE-ROWS` or in the output from `SHOW BINLOG EVENTS`.

As written in the output of statements such as `SHOW MASTER STATUS` or `SHOW SLAVE STATUS`, a sequence of GTIDs originating from the same server may be collapsed into a single expression, as shown here.

```
3E11FA47-71CA-11E1-9E33-C80AA9429562:1-5
```

The example just shown represents the first through fifth transactions originating on the MySQL Server whose `server_uuid` [2162] is `3E11FA47-71CA-11E1-9E33-C80AA9429562`.

This format is also used to supply the argument required by the `START SLAVE` options `SQL_BEFORE_GTIDS` and `SQL_AFTER_GTIDS`.

GTID sets. A GTID set is a set of global transaction identifiers which is represented as shown here:

```
gtid_set:
    uuid_set [, uuid_set] ...
    | ''

uuid_set:
    uuid:interval[:interval]...
```

```
uuid:
    hhhhhhhh-hhhh-hhhh-hhhh-hhhhhhhhhhhh

h:
    [0-9|A-F]

interval:
    n[-n]

    (n >= 1)
```

GTID sets are used in the MySQL Server in several ways. For example, the values stored by the `gtid_executed` and `gtid_purged` system variables are represented as GTID sets. In addition, the functions `GTID_SUBSET()` and `GTID_SUBTRACT()` require GTID sets as input.

GTIDs are always preserved between master and slave. This means that you can always determine the source for any transaction applied on any slave by examining its binary log. In addition, once a transaction with a given GTID is committed on a given server, any subsequent transaction having the same GTID is ignored by that server. Thus, a transaction committed on the master can be applied no more than once on the slave, which helps to guarantee consistency.

When GTIDs are in use, the slave has no need for any nonlocal data, such as the name of a file on the master and a position within that file. All necessary information for synchronizing with the master is obtained directly from the replication data stream. From the perspective of the database administrator or developer, GTIDs entirely take the place of the file-offset pairs previously required to determine points for starting, stopping, or resuming the flow of data between master and slave. This means that, when you are using GTIDs for replication, you do not need (or want) to include `MASTER_LOG_FILE` or `MASTER_LOG_POS` options in the `CHANGE MASTER TO` statement used to direct a slave to replicate from a given master; in place of these options, it is necessary only to enable the `MASTER_AUTO_POSITION` option. For the exact steps needed to configure and start masters and slaves using GTID-based replication, see Section 16.1.3.2, “Setting Up Replication Using GTIDs”.

The generation and lifecycle of a GTID consists of the following steps:

1. A transaction is executed and committed on the master.

   This transaction is assigned a GTID using the master's UUID and the smallest nonzero transaction sequence number not yet used on this server; the GTID is written to the master's binary log (immediately preceding the transaction itself in the log).

2. After the binary log data is transmitted to the slave and stored in the slave's relay log (using established mechanisms for this process—see Section 16.2, “Replication Implementation”, for details), the slave reads the GTID and sets the value of its `gtid_next` system variable as this GTID. This tells the slave that the next transaction must be logged using this GTID.

   It is important to note that the slave sets `gtid_next` in a session context.

3. The slave checks to make sure that this GTID has not already been used to log a transaction in its own binary log. If and only if this GTID has not been used, the slave then writes the GTID and applies the transaction (and writes the transaction to its binary log). By reading and checking the transaction's GTID first, before processing the transaction itself, the slave guarantees not only that no previous transaction having this GTID has been applied on the slave, but also that no other session has already read this GTID but has not yet committed the associated transaction. In other words, multiple clients are not permitted to apply the same transaction concurrently.

4. Because `gtid_next` is not empty, the slave does not attempt to generate a GTID for this transaction but instead writes the GTID stored in this variable—that is, the GTID obtained from the master—immediately preceding the transaction in its binary log.

16.1.3.2 Setting Up Replication Using GTIDs

This section describes a process for configuring and starting GTID-based replication in MySQL 5.7. This is a “cold start” procedure that assumes either that you are starting the replication master for the first time, or that it is possible to stop it; for information about provisioning replication slaves using GTIDs from a running master, see Section 16.1.3.3, “Using GTIDs for Failover and Scaleout”.

The key steps in this startup process for the simplest possible GTID replication topology—consisting of one master and one slave—are as follows:

1. If replication is already running, synchronize both servers by making them read-only.

2. Stop both servers.

3. Restart both servers with GTIDs, binary logging, and slave update logging enabled, and with statements that are unsafe for GTID-based replication disabled. In addition, the servers should be started in read-only mode, and the slave SQL and I/O threads should be prevented from starting on the slave.

   The mysqld options necessary to start the servers as described are discussed in the example that follows later in this section.

4. Instruct the slave to use the master as the replication data source and to use auto-positioning, and then start the slave.

   The SQL statements needed to accomplish this step are described in the example that follows later in this section.

5. Disable read-only mode on both servers, so that they can once again accept updates.

We now present a more detailed example. We assume two servers already running as master and slave, using MySQL's “classic” file-based replication protocol.

Most of the steps that follow require the use of the MySQL root account or another MySQL user account that has the SUPER privilege. mysqladmin shutdown requires either the SUPER privilege or the SHUTDOWN privilege.

Step 1: Synchronize the servers. Make the servers read-only. To do this, enable the read_only system variable by executing the following statement on both servers:

```
mysql> SET @@global.read_only = ON;
```

Then, allow the slave to catch up with the master. *It is extremely important that you make sure the slave has processed all updates before continuing*.

Step 2: Stop both servers. Stop each server using mysqladmin as shown here, where *username* is the user name for a MySQL user having sufficient privileges to shut down the server:

```
shell> mysqladmin -uusername -p shutdown
```

Then supply this user's password at the prompt.

Step 3: Restart both servers with GTIDs enabled. To enable binary logging with global transaction identifiers, each server must be started with GTID mode, binary logging, slave update logging enabled, and with statements that are unsafe for GTID-based replication disabled. In addition, you should prevent unwanted or accidental updates from being performed on either server by starting both in read-only mode. This means that both servers must be started with (at least) the options shown in the following invocation of mysqld_safe:

```
shell> mysqld_safe --gtid_mode=ON --log-bin --log-slave-updates --enforce-gtid-consistency &
```

In addition, you should start the slave with the `--skip-slave-start` option along with the other server options specified in the example just shown.

Although it may appear that `--gtid-mode` is a boolean, it is not (in fact, its values are enumerated). Use one of the values `ON` or `OFF` only, when setting this option. Using a numeric value such as 0 or 1 can lead to unexpected results.

For more information about the `--gtid-mode` and `--enforce-gtid-consistency` server options, see Section 16.1.4.5, "Global Transaction ID Options and Variables".

Depending on your circumstances, you may want or need to supply additional options to `mysqld_safe` (or other `mysqld` startup script).

Step 4: Direct the slave to use the master. Instruct the slave to use the master as the replication data source, and to use GTID-based auto-positioning rather than file-based positioning. You can do this by executing a `CHANGE MASTER TO` statement on the slave, using the `MASTER_AUTO_POSITION` option to tell the slave that transactions will be identified by GTIDs.

You may also need to supply appropriate values for the master's host name and port number as well as the user name and password for a replication user account which can be used by the slave to connect to the master; if these have already been set prior to Step 1 and no further changes need to be made, the corresponding options can safely be omitted from the statement shown here.

```
mysql> CHANGE MASTER TO
     >     MASTER_HOST = host,
     >     MASTER_PORT = port,
     >     MASTER_USER = user,
     >     MASTER_PASSWORD = password,
     >     MASTER_AUTO_POSITION = 1;
```

Neither the `MASTER_LOG_FILE` option nor the `MASTER_LOG_POS` option may be used with `MASTER_AUTO_POSITION` set equal to 1. Attempting to do so causes the `CHANGE MASTER TO` statement to fail with an error. (If you need to revert from GTID-based replication to replication based on files and positions, you must use one or both of these options together with `MASTER_AUTO_POSITION = 0` in the `CHANGE MASTER TO` statement.)

Assuming that the `CHANGE MASTER TO` statement has succeeded, you can then start the slave, like this:

```
mysql> START SLAVE;
```

Step 5: Disable read-only mode. Allow the master to begin accepting updates once again by running the following statement:

```
mysql> SET @@global.read_only = OFF;
```

GTID-based replication should now be running, and you can begin (or resume) activity on the master as before. Section 16.1.3.3, "Using GTIDs for Failover and Scaleout", discusses creation of new slaves when using GTIDs.

16.1.3.3 Using GTIDs for Failover and Scaleout

There are a number of techniques when using MySQL Replication with Global Transaction Identifiers (GTIDs) for provisioning a new slave which can then be used for scaleout, being promoted to master as necessary for failover. In this section, we discuss the four techniques listed here:

- Simple replication
- Copying data and transactions to the slave
- Injecting empty transactions
- Excluding transactions with gtid_purged

Global transaction identifiers were added to MySQL Replication for the purpose of simplifying in general management of the replication data flow and of failover activities in particular. Each identifier uniquely identifies a set of binary log events that together make up a transaction. GTIDs play a key role in applying changes to the database: the server automatically skips any transaction having an identifier which the server recognizes as one that it has processed before. This behavior is critical for automatic replication positioning and correct failover.

The mapping between identifiers and sets of events comprising a given transaction is captured in the binary log. This poses some challenges when a user wants to provision a new server with data from another existing server. To reproduce the identifier set on the new server, it is necessary, not only to copy the identifiers from the old server to the new one, but to preserve the relationship between the identifiers and the actual events as well, which is what is needed for restoring a slave that is immediately available as a candidate to become a new master on failover or switchover.

Simple replication. This is the easiest way to reproduce all identifiers and transactions on a new server; you simply make the new server into the slave of a master that has the entire execution history, and enable global transaction identifiers on both servers. (This requires that both master and slave are running with the options --gtid-mode=ON --log-bin --log-slave-updates --enforce-gtid-consistency; see Section 16.1.3.2, “Setting Up Replication Using GTIDs”, for more information.)

Once replication is started, the new server copies the entire binary log from the master and thus obtains all information about all GTIDs.

This method is simple and effective, but requires the slave to read the binary log from the master; it can sometimes take a comparatively long time for the new slave to catch up with the master, so this method is not suitable for fast failover or restoring from backup. We can obviate the need to fetch all of the execution history from the master by copying binary log files to the new server, as discussed in the next few paragraphs.

Copying data and transactions to the slave. Playing back the entire transaction history can be time-consuming, and represents a major bottleneck when setting up a new replication slave. To eliminate this requirement, we can take from the master a backup that includes, in addition to a dump containing a snapshot of the data set, the binary logs and the global transaction information they contain. Setting up the slave then consists of importing the snapshot, then playing back the binary log, after which replication can be started, allowing the slave to become current with any remaining transactions.

There are several variants of this method; these can be distinguished by the manner in which data (dumps) and transactions (binary logs) are shipped to the new slave, as outlined here:

Data Set	Transaction History
• Use the mysql client to import a dump file created with mysqldump. Use the --master-data option to include binary logging information and --set-gtid-purged to AUTO (the default) or ON, to include information about executed transactions. You should have --gtid-mode=ON while importing the dump on the slave.	If gtid_mode is not ON, restart the server with GTID mode enabled. • Import the binary log using mysqlbinlog, with the --read-from-remote-server and --read-from-remote-master options.

Data Set	Transaction History
• Stop the slave, copy the contents of the master's data directory to the slave's data directory, then restart the slave.	• Copy the master's binary log files to the slave. You can make copies from the slave using `mysqlbinlog --read-from-remote-server --raw`. These can be read in to the slave in either of the following ways: ◦ Update the slave's `binlog.index` file to point to the copied log files. Then execute a `CHANGE MASTER TO` statement in the `mysql` client to point to the first log file, and `START SLAVE` to read them. ◦ Use `mysqlbinlog > *file*` (without the `--raw` option) to export the binary log files to SQL files that can be processed by the `mysql` client.

See also Section 4.6.7.3, "Using `mysqlbinlog` to Back Up Binary Log Files".

This method has the advantage that a new server is available almost immediately; only those transactions that were committed while the snapshot or dump file was being replayed still need to be obtained from the existing master. This means that the slave's availability is not instantaneous—but only a relatively short amount of time should be required for the slave to catch up with these few remaining transactions.

Copying over binary logs to the target server in advance is usually faster than reading the entire transaction execution history from the master in real time. However, due to size or other considerations, it may not always be feasible to move these files to the target when required. The two remaining methods for provisioning a new slave discussed in this section use other means to convey information about transactions to the new slave.

Injecting empty transactions. The master's global `gtid_executed` variable contains the set of all transactions executed on the master. Rather than copy the binary logs when taking a snapshot to provision a new server, you can instead note the content of `gtid_executed` on the server from which the snapshot was taken. Before adding the new server to the replication chain, simply commit an empty transaction on the new server for each transaction identifier contained in the master's `gtid_executed`, like this:

```
SET GTID_NEXT='aaa-bbb-ccc-ddd:N';

BEGIN;
COMMIT;

SET GTID_NEXT='AUTOMATIC';
```

Once all transaction identifiers have been reinstated in this way using empty transactions, you must flush and purge the slave's binary logs, as shown here, where *N* is the nonzero suffix of the current binary log file name:

```
FLUSH LOGS;
PURGE BINARY LOGS TO 'master-bin.00000N';
```

You should do this to prevent this server from flooding the replication stream with false transactions in the event that it is later promoted to master. (The `FLUSH LOGS` statement forces the creation of a new binary log file; `PURGE BINARY LOGS` purges the empty transactions, but retains their identifiers.)

This method creates a server that is essentially a snapshot, but in time is able to become a master as its binary log history converges with that of the replication stream (that is, as it catches up with the master or masters). This outcome is similar in effect to that obtained using the remaining provisioning method, which we discuss in the next few paragraphs.

Excluding transactions with `gtid_purged`. The master's global `gtid_purged` variable contains the set of all transactions that have been purged from the master's binary log. As with the method discussed previously (see Injecting empty transactions), you can record the value of `gtid_executed` on the server from which the snapshot was taken (in place of copying the binary logs to the new server). Unlike the previous method, there is no need to commit empty transactions (or to issue `PURGE BINARY LOGS`); instead, you can set `gtid_purged` on the slave directly, based on the value of `gtid_executed` on the server from which the backup or snapshot was taken.

As with the method using empty transactions, this method creates a server that is functionally a snapshot, but in time is able to become a master as its binary log history converges with that of the replication master or group.

16.1.3.4 Restrictions on Replication with GTIDs

Because GTID-based replication is dependent on transactions, some features otherwise available in MySQL are not supported when using it. This section provides information about restrictions on and limitations of replication with GTIDs.

Updates involving nontransactional storage engines. When using GTIDs, updates to tables using nontransactional storage engines such as `MyISAM` cannot be made in the same statement or transaction as updates to tables using transactional storage engines such as `InnoDB`.

This restriction is due to the fact that updates to tables that use a nontransactional storage engine mixed with updates to tables that use a transactional storage engine within the same transaction can result in multiple GTIDs being assigned to the same transaction.

Such problems can also occur when the master and the slave use different storage engines for their respective versions of the same table, where one storage engine is transactional and the other is not.

In any of the cases just mentioned, the one-to-one correspondence between transactions and GTIDs is broken, with the result that GTID-based replication cannot function correctly.

`CREATE TABLE ... SELECT` statements. `CREATE TABLE ... SELECT` is not safe for statement-based replication. When using row-based replication, this statement is actually logged as two separate events—one for the creation of the table, and another for the insertion of rows from the source table into the new table just created. When this statement is executed within a transaction, it is possible in some cases for these two events to receive the same transaction identifier, which means that the transaction containing the inserts is skipped by the slave. Therefore, `CREATE TABLE ... SELECT` is not supported when using GTID-based replication.

Temporary tables. `CREATE TEMPORARY TABLE` and `DROP TEMPORARY TABLE` statements are not supported inside transactions when using GTIDs (that is, when the server was started with the `--enforce-gtid-consistency` option). It is possible to use use these statements with GTIDs enabled, but only outside of any transaction, and only with `autocommit=1`.

Preventing execution of unsupported statements. In order to prevent execution of statements that would cause GTID-based replication to fail, all servers must be started with the `--enforce-gtid-consistency` option when enabling GTIDs. This causes statements of any of the types discussed previously in this section to fail with an error.

For information about other required startup options when enabling GTIDs, see Section 16.1.3.2, “Setting Up Replication Using GTIDs”.

sql_slave_skip_counter is not supported when using GTIDs. If you need to skip transactions, use the value of the master's gtid_executed variable instead; see Injecting empty transactions, for more information.

GTID mode and mysqldump. It is possible to import a dump made using mysqldump into a MySQL Server running with GTID mode enabled, provided that there are no GTIDs in the target server's binary log.

GTID mode and mysql_upgrade. It is possible but is not recommended to to use mysql_upgrade on a MySQL Server running with --gtid-mode=ON, since mysql_upgrade can make changes to system tables that use the MyISAM storage engine, which is nontransactional.

16.1.4 Replication and Binary Logging Options and Variables

The next few sections contain information about mysqld options and server variables that are used in replication and for controlling the binary log. Options and variables for use on replication masters and replication slaves are covered separately, as are options and variables relating to binary logging. A set of quick-reference tables providing basic information about these options and variables is also included (in the next section following this one).

Of particular importance is the --server-id [2162] option.

Command-Line Format	--server-id=#	
Option-File Format	server-id	
System Variable Name	server_id	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	numeric
	Default	0
	Range	0 .. 4294967295

This option is common to both master and slave replication servers, and is used in replication to enable master and slave servers to identify themselves uniquely. For additional information, see Section 16.1.4.2, “Replication Master Options and Variables”, and Section 16.1.4.3, “Replication Slave Options and Variables”.

On the master and each slave, you *must* use the --server-id [2162] option to establish a unique replication ID in the range from 1 to 2^{32} – 1. “Unique”, means that each ID must be different from every other ID in use by any other replication master or slave. Example: server-id=3.

In MySQL 5.7.2 and earlier, if you omit --server-id [2162], the default ID is 0, in which case the master refuses connections from all slaves, and slaves refuse to connect to the master, and the server sets the server_id system variable to 1. In MySQL 5.7.3 and later, the --server-id must be used if binary logging is enabled, and a value of 0 is not changed by the server. (You can let the default be used by specifying --server-id without an argument, but the effect is the same as using 0.) In either case, if the server_id is 0, binary logging takes place, but the server cannot connect to any slaves as a master, nor can any other servers connect to it as slaves. (Bug #11763963, Bug #56718)

For more information, see Section 16.1.1.2, “Setting the Replication Slave Configuration”.

server_uuid [2162]

In MySQL 5.7, the server generates a true UUID in addition to the --server-id [2162] supplied by the user. This is available as the global, read-only variable server_uuid [2162].

System Variable Name	server_uuid [2162]	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	string

When starting, the MySQL server automatically obtains a UUID as follows:

1. Attempt to read and use the UUID written in the file *data_dir*/auto.cnf (where *data_dir* is the server's data directory); exit on success.

2. Otherwise, generate a new UUID and save it to this file, creating the file if necessary.

The auto.cnf file has a format similar to that used for my.cnf or my.ini files. In MySQL 5.7, auto.cnf has only a single [auto] section containing a single server_uuid [2162] setting and value; the file's contents appear similar to what is shown here:

```
[auto]
server_uuid=8a94f357-aab4-11df-86ab-c80aa9429562
```

Important

The auto.cnf file is automatically generated; you should not attempt to write or modify this file.

When using MySQL replication, masters and slaves know one another's UUIDs. The value of a slave's UUID can be seen in the output of SHOW SLAVE HOSTS. Once START SLAVE has been executed (but not before), the value of the master's UUID is available on the slave in the output of SHOW SLAVE STATUS.

Note

Issuing a STOP SLAVE or RESET SLAVE statement does *not* reset the master's UUID as used on the slave.

A server's server_uuid is also used in GTIDs for transactions originating on that server. For more information, see Section 16.1.3, “Replication with Global Transaction Identifiers”.

When starting, the slave I/O thread generates an error and aborts if its master's UUID is equal to its own unless the --replicate-same-server-id option has been set. In addition, the slave I/O thread generates a warning if either of the following is true:

- No master having the expected server_uuid [2162] exists.

- The master's server_uuid [2162] has changed, although no CHANGE MASTER TO statement has ever been executed.

Note

The addition of the server_uuid [2162] system variable in MySQL 5.7 does not change the requirement for setting a unique --server-id [2162] for each MySQL server as part of preparing and running MySQL replication, as described earlier in this section.

16.1.4.1 Replication and Binary Logging Option and Variable Reference

The following tables list basic information about the MySQL command-line options and system variables applicable to replication and the binary log.

Table 16.1 Replication Options and Variables: MySQL 5.7

Option or Variable Name	Command Line	System Variable	Scope
	Option File	Status Variable	Dynamic
Com_change_master	N	N	Both
	N	Y	N
Com_show_master_status	N	N	Both
	N	Y	N
Com_show_new_master	N	N	Both
	N	Y	N
Com_show_slave_hosts	N	N	Both
	N	Y	N
Com_show_slave_status	N	N	Both
	N	Y	N
Com_slave_start	N	N	Both
	N	Y	N
Com_slave_stop	N	N	Both
	N	Y	N
Rpl_semi_sync_master_clients	N	N	Global
	N	Y	N
Rpl_semi_sync_master_net_avg_wait_time	N	N	Global
	N	Y	N
Rpl_semi_sync_master_net_wait_time	N	N	Global
	N	Y	N
Rpl_semi_sync_master_net_waits	N	N	Global
	N	Y	N
Rpl_semi_sync_master_no_times	N	N	Global
	N	Y	N
Rpl_semi_sync_master_no_tx	N	N	Global
	N	Y	N
Rpl_semi_sync_master_status	N	N	Global
	N	Y	N
Rpl_semi_sync_master_timefunc_failures	N	N	Global
	N	Y	N
Rpl_semi_sync_master_tx_avg_wait_time	N	N	Global

Option or Variable Name	Command Line	System Variable	Scope
	Option File	Status Variable	Dynamic
	N	Y	N
Rpl_semi_sync_master_tx_wait_time	N	N	Global
	N	Y	N
Rpl_semi_sync_master_tx_waits	N	N	Global
	N	Y	N
Rpl_semi_sync_master_wait_pos_backtraverse	N	N	Global
	N	Y	N
Rpl_semi_sync_master_wait_sessions	N	N	Global
	N	Y	N
Rpl_semi_sync_master_yes_tx	N	N	Global
	N	Y	N
Rpl_semi_sync_slave_status	N	N	Global
	N	Y	N
slave_exec_mode	Y	Y	Global
	Y	N	Y
Slave_open_temp_tables	N	N	Global
	N	Y	N
Slave_retried_transactions	N	N	Global
	N	Y	N
Slave_running	N	N	Global
	N	Y	N
abort-slave-event-count	Y	N	Global
	Y	N	N
disconnect-slave-event-count	Y	N	Global
	Y	N	N
enforce-gtid-consistency	Y	Y	Global
	Y	N	N
enforce_gtid_consistency	Y	Y	Global
	Y	N	N
gtid_executed	N	Y	Both
	N	N	N
gtid-mode	Y	Y	Global
	Y	N	N
gtid_mode	N	Y	Global
	N	N	N

Option or Variable Name	Command Line	System Variable	Scope
	Option File	**Status Variable**	**Dynamic**
gtid_next	N	Y	Session
	N	N	Y
gtid_owned	N	Y	Both
	N	N	N
gtid_purged	N	Y	Global
	N	N	Y
init_slave	Y	Y	Global
	Y	N	Y
log-slave-updates	Y	Y	Global
	Y	N	N
log_slave_updates	Y	Y	Global
	Y	N	N
master-info-file	Y	N	Global
	Y	N	N
master-info-repository	Y	N	Global
	Y	N	N
master_info_repository	Y	Y	Global
	Y	N	Y
master-retry-count	Y	N	Global
	Y	N	N
relay-log	Y	Y	Global
	Y	N	N
relay_log_basename	N	Y	Global
	N	N	N
relay-log-index	Y	Y	Global
	Y	N	N
relay-log-info-file	Y	N	Global
	Y	N	N
relay_log_info_file	Y	Y	Global
	Y	N	N
relay-log-info-repository	Y	N	Global
	Y	N	N
relay_log_info_repository	N	Y	Global
	N	N	Y
relay_log_index	Y	Y	Global

Option or Variable Name	Command Line	System Variable	Scope
	Option File	Status Variable	Dynamic
	Y	N	N
relay_log_purge	Y	Y	Global
	Y	N	Y
relay-log-recovery	Y	N	Global
	Y	N	N
relay_log_recovery	Y	Y	Global
	Y	N	Y
relay_log_space_limit	Y	Y	Global
	Y	N	N
replicate-do-db	Y	N	Global
	Y	N	N
replicate-do-table	Y	N	Global
	Y	N	N
replicate-ignore-db	Y	N	Global
	Y	N	N
replicate-ignore-table	Y	N	Global
	Y	N	N
replicate-rewrite-db	Y	N	Global
	Y	N	N
replicate-same-server-id	Y	N	Global
	Y	N	N
replicate-wild-do-table	Y	N	Global
	Y	N	N
replicate-wild-ignore-table	Y	N	Global
	Y	N	N
report-host	Y	Y	Global
	Y	N	N
report-password	Y	Y	Global
	Y	N	N
report-port	Y	Y	Global
	Y	N	N
report-user	Y	Y	Global
	Y	N	N
rpl_semi_sync_master_enabled	N	Y	Global
	N	N	Y

Option or Variable Name	Command Line	System Variable	Scope
	Option File	Status Variable	Dynamic
rpl_semi_sync_master_timeout	N	Y	Global
	N	N	Y
rpl_semi_sync_master_trace_level	N	Y	Global
	N	N	Y
rpl_semi_sync_master_wait_for_slave_count	N	Y	Global
	N	N	Y
rpl_semi_sync_master_wait_no_slave	N	Y	Global
	N	N	Y
rpl_semi_sync_master_wait_point	N	Y	Global
	N	N	Y
rpl_semi_sync_slave_enabled	N	Y	Global
	N	N	Y
rpl_semi_sync_slave_trace_level	N	Y	Global
	N	N	Y
rpl_stop_slave_timeout	Y	Y	Global
	Y	N	Y
server_uuid [2162]	N	Y	Global
	N	N	N
show-slave-auth-info	Y	N	Global
	Y	N	N
skip-slave-start	Y	N	Global
	Y	N	N
slave_allow_batching	Y	Y	Global
	Y	N	Y
slave-load-tmpdir	Y	Y	Global
	Y	N	N
slave-skip-errors	Y	Y	Global
	Y	N	N
slave-checkpoint-group	Y	N	Global
	Y	N	N
slave_checkpoint_group	Y	Y	Global
	Y	N	Y
slave-checkpoint-period	Y	N	Global
	Y	N	N
slave_checkpoint_period	Y	Y	Global

Option or Variable Name	Command Line	System Variable	Scope
	Option File	Status Variable	Dynamic
	Y	N	Y
slave_compressed_protocol	Y	Y	Global
	Y	N	Y
slave-max-allowed-packet	Y	N	Global
	Y	N	N
slave_max_allowed_packet	N	Y	Global
	N	N	Y
slave_net_timeout	Y	Y	Global
	Y	N	Y
slave-parallel-type	Y	N	Global
	Y	N	N
slave_parallel_type	N	Y	Global
	N	N	Y
slave_parallel_workers	N	Y	Global
	N	N	Y
slave-parallel-workers	Y	N	Global
	Y	N	N
slave-pending-jobs-size-max	Y	N	Global
	N	N	N
slave_pending_jobs_size_max	N	Y	Global
	N	N	Y
slave-rows-search-algorithms	Y	N	Global
	Y	N	N
slave_rows_search_algorithms	N	Y	Global
	N	N	Y
slave_transaction_retries	Y	Y	Global
	Y	N	Y
slave_type_conversions	Y	Y	Global
	Y	N	N
sql_slave_skip_counter	N	Y	Global
	N	N	Y
sync_binlog	Y	Y	Global
	Y	N	Y
sync_master_info	Y	Y	Global
	Y	N	Y

Option or Variable Name	Command Line	System Variable	Scope
	Option File	Status Variable	Dynamic
sync_relay_log	Y	Y	Global
	Y	N	Y
sync_relay_log_info	Y	Y	Global
	Y	N	Y

Section 16.1.4.2, “Replication Master Options and Variables”, provides more detailed information about options and variables relating to replication master servers. For more information about options and variables relating to replication slaves, see Section 16.1.4.3, “Replication Slave Options and Variables”.

Table 16.2 Binary Logging Options and Variables: MySQL 5.7

Option or Variable Name	Command Line	System Variable	Scope
	Option File	Status Variable	Dynamic
Binlog_cache_disk_use	N	N	Global
	N	Y	N
binlog_row_image	Y	Y	Both
	Y	N	Y
binlog_rows_query_log_events	N	Y	Both
	N	N	Y
Binlog_stmt_cache_disk_use	N	N	Global
	N	Y	N
Binlog_cache_use	N	N	Global
	N	Y	N
Binlog_stmt_cache_use	N	N	Global
	N	Y	N
Com_show_binlog_events	N	N	Both
	N	Y	N
Com_show_binlogs	N	N	Both
	N	Y	N
binlog-checksum	Y	N	Global
	Y	N	N
binlog_checksum	N	Y	Global
	N	N	Y
binlog-do-db	Y	N	Global
	Y	N	N
binlog-ignore-db	Y	N	Global
	Y	N	N

Option or Variable Name	Command Line	System Variable	Scope
	Option File	Status Variable	Dynamic
binlog-row-event-max-size	Y	N	Global
	Y	N	N
binlog_cache_size	Y	Y	Global
	Y	N	Y
binlog_max_flush_queue_time	N	Y	Global
	N	N	Y
binlog_order_commits	N	Y	Global
	N	N	Y
binlog_stmt_cache_size	Y	Y	Global
	Y	N	Y
binlog_format	Y	Y	Both
	Y	N	Y
binlog-rows-query-log-events	Y	N	Global
	Y	N	N
binlog_direct_non_transactional_updates	Y	Y	Both
	Y	N	Y
log_bin_basename	N	Y	Global
	N	N	N
log-bin-use-v1-row-events	Y	Y	Global
	Y	N	N
log_bin_use_v1_row_events	Y	Y	Global
	Y	N	N
master-verify-checksum	Y	N	Global
	Y	N	N
master_verify_checksum	N	Y	Global
	N	N	Y
max-binlog-dump-events	Y	N	Global
	Y	N	N
max_binlog_cache_size	Y	Y	Global
	Y	N	Y
max_binlog_size	Y	Y	Global
	Y	N	Y
max_binlog_stmt_cache_size	Y	Y	Global
	Y	N	Y
slave-sql-verify-checksum	Y	N	Global

Option or Variable Name	Command Line	System Variable	Scope
	Option File	Status Variable	Dynamic
	Y	N	N
slave_sql_verify_checksum	N	Y	Global
	N	N	Y
sporadic-binlog-dump-fail	Y	N	Global
	Y	N	N

Section 16.1.4.4, “Binary Log Options and Variables”, provides more detailed information about options and variables relating to binary logging. For additional general information about the binary log, see Section 5.2.4, “The Binary Log”.

For information about the sql_log_bin and sql_log_off variables, see Section 5.1.4, “Server System Variables”.

For a table showing *all* command-line options, system and status variables used with mysqld, see Section 5.1.1, “Server Option and Variable Reference”.

16.1.4.2 Replication Master Options and Variables

This section describes the server options and system variables that you can use on replication master servers. You can specify the options either on the command line or in an option file. You can specify system variable values using SET.

On the master and each slave, you must use the server-id [2162] option to establish a unique replication ID. For each server, you should pick a unique positive integer in the range from 1 to 2^{32} – 1, and each ID must be different from every other ID in use by any other replication master or slave. Example: server-id=3.

For options used on the master for controlling binary logging, see Section 16.1.4.4, “Binary Log Options and Variables”.

System variables used on replication masters. The following system variables are used in controlling replication masters:

- auto_increment_increment

System Variable Name	auto_increment_increment	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	numeric
	Default	1
	Range	1 .. 65535

auto_increment_increment and auto_increment_offset are intended for use with master-to-master replication, and can be used to control the operation of AUTO_INCREMENT columns. Both variables have global and session values, and each can assume an integer value between 1 and 65,535 inclusive. Setting the value of either of these two variables to 0 causes its value to be set to 1 instead. Attempting to set the value of either of these two variables to an integer greater than 65,535 or less than 0 causes its value to be set to 65,535 instead. Attempting to set the value of auto_increment_increment or auto_increment_offset to a noninteger value gives rise to an error, and the actual value of the variable remains unchanged.

Note

auto_increment_increment is intended for use with MySQL Cluster, which is not currently supported in MySQL 5.7.

These two variables affect AUTO_INCREMENT column behavior as follows:

- auto_increment_increment controls the interval between successive column values. For example:

```
mysql> SHOW VARIABLES LIKE 'auto_inc%';
+--------------------------+-------+
| Variable_name            | Value |
+--------------------------+-------+
| auto_increment_increment | 1     |
| auto_increment_offset    | 1     |
+--------------------------+-------+
2 rows in set (0.00 sec)

mysql> CREATE TABLE autoinc1
    -> (col INT NOT NULL AUTO_INCREMENT PRIMARY KEY);
  Query OK, 0 rows affected (0.04 sec)

mysql> SET @@auto_increment_increment=10;
Query OK, 0 rows affected (0.00 sec)

mysql> SHOW VARIABLES LIKE 'auto_inc%';
+--------------------------+-------+
| Variable_name            | Value |
+--------------------------+-------+
| auto_increment_increment | 10    |
| auto_increment_offset    | 1     |
+--------------------------+-------+
2 rows in set (0.01 sec)

mysql> INSERT INTO autoinc1 VALUES (NULL), (NULL), (NULL), (NULL);
Query OK, 4 rows affected (0.00 sec)
Records: 4  Duplicates: 0  Warnings: 0

mysql> SELECT col FROM autoinc1;
+-----+
| col |
+-----+
|   1 |
|  11 |
|  21 |
|  31 |
+-----+
4 rows in set (0.00 sec)
```

- auto_increment_offset determines the starting point for the AUTO_INCREMENT column value. Consider the following, assuming that these statements are executed during the same session as the example given in the description for auto_increment_increment:

```
mysql> SET @@auto_increment_offset=5;
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> SHOW VARIABLES LIKE 'auto_inc%';
+--------------------------+-------+
| Variable_name            | Value |
+--------------------------+-------+
| auto_increment_increment | 10    |
| auto_increment_offset    | 5     |
+--------------------------+-------+
2 rows in set (0.00 sec)

mysql> CREATE TABLE autoinc2
    -> (col INT NOT NULL AUTO_INCREMENT PRIMARY KEY);
Query OK, 0 rows affected (0.06 sec)

mysql> INSERT INTO autoinc2 VALUES (NULL), (NULL), (NULL), (NULL);
Query OK, 4 rows affected (0.00 sec)
Records: 4  Duplicates: 0  Warnings: 0

mysql> SELECT col FROM autoinc2;
+-----+
| col |
+-----+
|   5 |
|  15 |
|  25 |
|  35 |
+-----+
4 rows in set (0.02 sec)
```

If the value of `auto_increment_offset` is greater than that of `auto_increment_increment`, the value of `auto_increment_offset` is ignored.

Should one or both of these variables be changed and then new rows inserted into a table containing an `AUTO_INCREMENT` column, the results may seem counterintuitive because the series of `AUTO_INCREMENT` values is calculated without regard to any values already present in the column, and the next value inserted is the least value in the series that is greater than the maximum existing value in the `AUTO_INCREMENT` column. In other words, the series is calculated like so:

`auto_increment_offset` + *N* × `auto_increment_increment`

where *N* is a positive integer value in the series [1, 2, 3, ...]. For example:

```
mysql> SHOW VARIABLES LIKE 'auto_inc%';
+--------------------------+-------+
| Variable_name            | Value |
+--------------------------+-------+
| auto_increment_increment | 10    |
| auto_increment_offset    | 5     |
+--------------------------+-------+
2 rows in set (0.00 sec)

mysql> SELECT col FROM autoinc1;
+-----+
| col |
+-----+
|   1 |
|  11 |
|  21 |
|  31 |
+-----+
4 rows in set (0.00 sec)

mysql> INSERT INTO autoinc1 VALUES (NULL), (NULL), (NULL), (NULL);
Query OK, 4 rows affected (0.00 sec)
Records: 4  Duplicates: 0  Warnings: 0
```

```
mysql> SELECT col FROM autoinc1;
+-----+
| col |
+-----+
|   1 |
|  11 |
|  21 |
|  31 |
|  35 |
|  45 |
|  55 |
|  65 |
+-----+
8 rows in set (0.00 sec)
```

The values shown for `auto_increment_increment` and `auto_increment_offset` generate the series 5 + *N* × 10, that is, [5, 15, 25, 35, 45, ...]. The greatest value present in the `col` column prior to the `INSERT` is 31, and the next available value in the `AUTO_INCREMENT` series is 35, so the inserted values for `col` begin at that point and the results are as shown for the `SELECT` query.

It is not possible to confine the effects of these two variables to a single table, and thus they do not take the place of the sequences offered by some other database management systems; these variables control the behavior of all `AUTO_INCREMENT` columns in *all* tables on the MySQL server. If the global value of either variable is set, its effects persist until the global value is changed or overridden by setting the session value, or until `mysqld` is restarted. If the local value is set, the new value affects `AUTO_INCREMENT` columns for all tables into which new rows are inserted by the current user for the duration of the session, unless the values are changed during that session.

The default value of `auto_increment_increment` is 1. See Section 16.4.1.1, “Replication and `AUTO_INCREMENT`”.

- `auto_increment_offset`

System Variable Name	auto_increment_offset	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	numeric
	Default	1
	Range	1 .. 65535

This variable has a default value of 1. For particulars, see the description for `auto_increment_increment`.

Note

`auto_increment_offset` is intended for use with MySQL Cluster, which is not currently supported in MySQL 5.7.

16.1.4.3 Replication Slave Options and Variables

This section describes the server options and system variables that apply to slave replication servers. You can specify the options either on the command line or in an option file. Many of the options can be set while the server is running by using the `CHANGE MASTER TO` statement. You can specify system variable values using `SET`.

Server ID. On the master and each slave, you must use the `server-id` [2162] option to establish a unique replication ID in the range from 1 to 2^{32} – 1. “Unique” means that each ID must be different from every other ID in use by any other replication master or slave. Example `my.cnf` file:

```
[mysqld]
server-id=3
```

Startup options for replication slaves. The following list describes startup options for controlling replication slave servers. Many of these options can be set while the server is running by using the `CHANGE MASTER TO` statement. Others, such as the `--replicate-*` options, can be set only when the slave server starts. Replication-related system variables are discussed later in this section.

- `--abort-slave-event-count`

Command-Line Format	`--abort-slave-event-count=#`	
Option-File Format	`abort-slave-event-count`	
	Permitted Values	
	Type	`numeric`
	Default	`0`
	Min Value	`0`

When this option is set to some positive integer *`value`* other than 0 (the default) it affects replication behavior as follows: After the slave SQL thread has started, *`value`* log events are permitted to be executed; after that, the slave SQL thread does not receive any more events, just as if the network connection from the master were cut. The slave thread continues to run, and the output from `SHOW SLAVE STATUS` displays `Yes` in both the `Slave_IO_Running` and the `Slave_SQL_Running` columns, but no further events are read from the relay log.

This option is used internally by the MySQL test suite for replication testing and debugging. It is not intended for use in a production setting.

- `--disconnect-slave-event-count`

Command-Line Format	`--disconnect-slave-event-count=#`	
Option-File Format	`disconnect-slave-event-count`	
	Permitted Values	
	Type	`numeric`
	Default	`0`

This option is used internally by the MySQL test suite for replication testing and debugging.

- `--log-slave-updates`

Command-Line Format	`--log-slave-updates`
Option-File Format	`log-slave-updates`
System Variable Name	`log_slave_updates`
Variable Scope	Global
Dynamic Variable	No

	Permitted Values	
	Type	boolean
	Default	FALSE

Normally, a slave does not log to its own binary log any updates that are received from a master server. This option tells the slave to log the updates performed by its SQL thread to its own binary log. For this option to have any effect, the slave must also be started with the --log-bin option to enable binary logging. Prior to MySQL 5.5, the server would not start when using the --log-slave-updates option without also starting the server with the --log-bin option, and would fail with an error; in MySQL 5.7, only a warning is generated. (Bug #44663) --log-slave-updates is used when you want to chain replication servers. For example, you might want to set up replication servers using this arrangement:

```
A -> B -> C
```

Here, A serves as the master for the slave B, and B serves as the master for the slave C. For this to work, B must be both a master *and* a slave. You must start both A and B with --log-bin to enable binary logging, and B with the --log-slave-updates option so that updates received from A are logged by B to its binary log.

- --log-slow-slave-statements

Removed	5.7.1	
Command-Line Format	--log-slow-slave-statements	through 5.7.0
Option-File Format	log-slow-slave-statements	
	Permitted Values	
	Type	boolean
	Default	OFF

When the slow query log is enabled, this option enables logging for queries that have taken more than long_query_time seconds to execute on the slave.

This command-line option was removed in MySQL 5.7.1 and replaced by the log_slow_slave_statements system variable. The system variable can be set on the command line or in option files the same way as the option, so there is no need for any changes at server startup, but the system variable also makes it possible to examine or set the value at runtime.

- --log-warnings[=*level*]

Deprecated	5.7.2
Command-Line Format	--log-warnings[=#]
	-W [#]
Option-File Format	log-warnings[=#]
System Variable Name	log_warnings
Variable Scope	Global
Dynamic Variable	Yes

	Permitted Values	
	Platform Bit Size	32
	Type	numeric
	Default	1
	Range	0 .. 4294967295
	Permitted Values	
	Platform Bit Size	64
	Type	numeric
	Default	1
	Range	0 .. 18446744073709547520

Note

As of MySQL 5.7.2, the `log_error_verbosity` system variable is preferred over, and should be used instead of, the `--log-warnings` option or `log_warnings` system variable. For more information, see the descriptions of `log_error_verbosity` and `log_warnings`. The `--log-warnings` command-line option and `log_warnings` system variable are deprecated and will be removed in a future MySQL release.

This option causes a server to print more messages to the error log about what it is doing. With respect to replication, the server generates warnings that it succeeded in reconnecting after a network/ connection failure, and informs you as to how each slave thread started. This option is enabled (1) by default; to disable it, use `--log-warnings=0`. If the value is greater than 1, aborted connections are written to the error log, and access-denied errors for new connection attempts are written. See Section C.5.2.11, “Communication Errors and Aborted Connections”.

Note that the effects of this option are not limited to replication. It produces warnings across a spectrum of server activities.

- `--master-info-file=`*`file_name`*

Command-Line Format	`--master-info-file=file_name`	
Option-File Format	`master-info-file`	
	Permitted Values	
	Type	file name
	Default	master.info

The name to use for the file in which the slave records information about the master. The default name is `master.info` in the data directory. For information about the format of this file, see Section 16.2.2.2, “Slave Status Logs”.

- `--master-retry-count=`*`count`*

Deprecated	5.6.1
Command-Line Format	`--master-retry-count=#`
Option-File Format	`master-retry-count`

	Permitted Values	
	Platform Bit Size	32
	Type	numeric
	Default	86400
	Range	0 .. 4294967295
	Permitted Values	
	Platform Bit Size	64
	Type	numeric
	Default	86400
	Range	0 .. 18446744073709551615

The number of times that the slave tries to connect to the master before giving up. Reconnects are attempted at intervals set by the `MASTER_CONNECT_RETRY` option of the `CHANGE MASTER TO` statement (default 60). Reconnects are triggered when data reads by the slave time out according to the `--slave-net-timeout` option. The default value is 86400. A value of 0 means “infinite”; the slave attempts to connect forever.

This option is deprecated and will be removed in a future MySQL release. Applications should be updated to use the `MASTER_RETRY_COUNT` option of the `CHANGE MASTER TO` statement instead.

- `--max-relay-log-size=size`

Command-Line Format	--max_relay_log_size=#	
Option-File Format	max_relay_log_size	
System Variable Name	max_relay_log_size	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	numeric
	Default	0
	Range	0 .. 1073741824

The size at which the server rotates relay log files automatically. For more information, see Section 16.2.2, “Replication Relay and Status Logs”. The default size is 1GB.

- `--read-only`

Command-Line Format	--read-only	
Option-File Format	read_only	
System Variable Name	read_only	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	

	Type	boolean
	Default	false

Cause the slave to permit no updates except from slave threads or from users having the SUPER privilege. On a slave server, this can be useful to ensure that the slave accepts updates only from its master server and not from clients. This variable does not apply to TEMPORARY tables.

- --relay-log=*file_name*

Command-Line Format	--relay-log=name	
Option-File Format	relay-log	
System Variable Name	relay_log	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	file name

The basename for the relay log. The default basename is *host_name*-relay-bin. The server writes the file in the data directory unless the basename is given with a leading absolute path name to specify a different directory. The server creates relay log files in sequence by adding a numeric suffix to the basename.

Due to the manner in which MySQL parses server options, if you specify this option, you must supply a value; *the default basename is used only if the option is not actually specified*. If you use the --relay-log option without specifying a value, unexpected behavior is likely to result; this behavior depends on the other options used, the order in which they are specified, and whether they are specified on the command line or in an option file. For more information about how MySQL handles server options, see Section 4.2.3, “Specifying Program Options”.

If you specify this option, the value specified is also used as the basename for the relay log index file. You can override this behavior by specifying a different relay log index file basename using the --relay-log-index option.

When the server reads an entry from the index file, it checks whether the entry contains a relative path. If it does, the relative part of the path in replaced with the absolute path set using the --relay-log option. An absolute path remains unchanged; in such a case, the index must be edited manually to enable the new path or paths to be used. (Previously, manual intervention was required whenever relocating the binary log or relay log files.) (Bug #11745230, Bug #12133)

You may find the --relay-log option useful in performing the following tasks:

- Creating relay logs whose names are independent of host names.
- If you need to put the relay logs in some area other than the data directory because your relay logs tend to be very large and you do not want to decrease max_relay_log_size.
- To increase speed by using load-balancing between disks.

You can obtain the relay log filename (and path) from the relay_log_basename system variable.

- --relay-log-index=*file_name*

Command-Line Format	`--relay-log-index=name`	
Option-File Format	`relay-log-index`	
System Variable Name	`relay_log_index`	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	`file name`

The name to use for the relay log index file. The default name is `*host_name*-relay-bin.index` in the data directory, where `*host_name*` is the name of the slave server.

Due to the manner in which MySQL parses server options, if you specify this option, you must supply a value; *the default basename is used only if the option is not actually specified*. If you use the `--relay-log-index` option without specifying a value, unexpected behavior is likely to result; this behavior depends on the other options used, the order in which they are specified, and whether they are specified on the command line or in an option file. For more information about how MySQL handles server options, see Section 4.2.3, “Specifying Program Options”.

If you specify this option, the value specified is also used as the basename for the relay logs. You can override this behavior by specifying a different relay log file basename using the `--relay-log` option.

- `--relay-log-info-file=*file_name*`

Command-Line Format	`--relay-log-info-file=file_name`	
Option-File Format	`relay-log-info-file`	
	Permitted Values	
	Type	`file name`
	Default	`relay-log.info`

The name to use for the file in which the slave records information about the relay logs. The default name is `relay-log.info` in the data directory. For information about the format of this file, see Section 16.2.2.2, “Slave Status Logs”.

- `--relay-log-purge={0|1}`

Command-Line Format	`--relay_log_purge`	
Option-File Format	`relay_log_purge`	
System Variable Name	`relay_log_purge`	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	`boolean`
	Default	`TRUE`

Disable or enable automatic purging of relay logs as soon as they are no longer needed. The default value is 1 (enabled). This is a global variable that can be changed dynamically with `SET GLOBAL relay_log_purge = *N*`.

- `--relay-log-recovery={0|1}`

Command-Line Format	`--relay-log-recovery`	
Option-File Format	`relay-log-recovery`	
	Permitted Values	
	Type	`boolean`
	Default	`FALSE`

Enables automatic relay log recovery immediately following server startup, which means that the replication slave discards all unprocessed relay logs and retrieves them from the replication master. This should be used following a crash on the replication slave to ensure that no possibly corrupted relay logs are processed. The default value is 0 (disabled).

To provide a crash-proof slave, this option must be enabled (set to 1), and `--relay-log-info-repository` must be set to `TABLE`. See Crash-safe replication, for more information.

If this option is enabled for a multi-threaded slave, and the slave fails with errors, you can use `START SLAVE UNTIL SQL_AFTER_MTS_GAPS` to ensure that any gaps in the relay log are processed; after running this statement, you can then use `CHANGE MASTER TO` to fail this slave over to a new master. (Bug #13893363)

- `--relay-log-space-limit=`*`size`*

Command-Line Format	`--relay_log_space_limit=#`	
Option-File Format	`relay_log_space_limit`	
System Variable Name	`relay_log_space_limit`	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Platform Bit Size	`32`
	Type	`numeric`
	Default	`0`
	Range	`0 .. 4294967295`
	Permitted Values	
	Platform Bit Size	`64`
	Type	`numeric`
	Default	`0`
	Range	`0 .. 18446744073709547520`

This option places an upper limit on the total size in bytes of all relay logs on the slave. A value of 0 means "no limit." This is useful for a slave server host that has limited disk space. When the limit is reached, the I/O thread stops reading binary log events from the master server until the SQL thread has caught up and deleted some unused relay logs. Note that this limit is not absolute: There are cases where the SQL thread needs more events before it can delete relay logs. In that case, the I/O thread exceeds the limit until it becomes possible for the SQL thread to delete some relay logs because not doing so would cause a deadlock. You should not set `--relay-log-space-limit` to less than twice the value of `--max-relay-log-size` (or `--max-binlog-size` if `--max-relay-log-size` is 0). In that case, there is a chance that the I/O thread waits for free space because `--relay-log-space-limit` is exceeded, but the SQL thread has no relay log to purge and is unable to satisfy the I/O thread. This forces the I/O thread to ignore `--relay-log-space-limit` temporarily.

- `--replicate-do-db=`*`db_name`*

Command-Line Format	`--replicate-do-db=name`	
Option-File Format	`replicate-do-db`	
	Permitted Values	
	Type	`string`

Creates a replication filter using the name of a database. In MySQL 5.7.3 and later, such filters can also be created using `CHANGE REPLICATION FILTER REPLICATE_DO_DB`. The precise effect of this filtering depends on whether statement-based or row-based replication is in use, and are described in the next several paragraphs.

Statement-based replication. Tell the slave SQL thread to restrict replication to statements where the default database (that is, the one selected by `USE`) is *`db_name`*. To specify more than one database, use this option multiple times, once for each database; however, doing so does *not* replicate cross-database statements such as `UPDATE` *`some_db.some_table`* `SET foo='bar'` while a different database (or no database) is selected.

> **Warning**
>
> To specify multiple databases you *must* use multiple instances of this option. Because database names can contain commas, if you supply a comma separated list then the list will be treated as the name of a single database.

An example of what does not work as you might expect when using statement-based replication: If the slave is started with `--replicate-do-db=sales` and you issue the following statements on the master, the `UPDATE` statement is *not* replicated:

```
USE prices;
UPDATE sales.january SET amount=amount+1000;
```

The main reason for this “check just the default database” behavior is that it is difficult from the statement alone to know whether it should be replicated (for example, if you are using multiple-table `DELETE` statements or multiple-table `UPDATE` statements that act across multiple databases). It is also faster to check only the default database rather than all databases if there is no need.

Row-based replication. Tells the slave SQL thread to restrict replication to database *`db_name`*. Only tables belonging to *`db_name`* are changed; the current database has no effect on this. Suppose that the slave is started with `--replicate-do-db=sales` and row-based replication is in effect, and then the following statements are run on the master:

```
USE prices;
UPDATE sales.february SET amount=amount+100;
```

The `february` table in the `sales` database on the slave is changed in accordance with the `UPDATE` statement; this occurs whether or not the `USE` statement was issued. However, issuing the following statements on the master has no effect on the slave when using row-based replication and `--replicate-do-db=sales`:

```
USE prices;
UPDATE prices.march SET amount=amount-25;
```

Even if the statement `USE prices` were changed to `USE sales`, the `UPDATE` statement's effects would still not be replicated.

Another important difference in how `--replicate-do-db` is handled in statement-based replication as opposed to row-based replication occurs with regard to statements that refer to multiple databases. Suppose that the slave is started with `--replicate-do-db=db1`, and the following statements are executed on the master:

```
USE db1;
UPDATE db1.table1 SET col1 = 10, db2.table2 SET col2 = 20;
```

If you are using statement-based replication, then both tables are updated on the slave. However, when using row-based replication, only `table1` is affected on the slave; since `table2` is in a different database, `table2` on the slave is not changed by the `UPDATE`. Now suppose that, instead of the `USE db1` statement, a `USE db4` statement had been used:

```
USE db4;
UPDATE db1.table1 SET col1 = 10, db2.table2 SET col2 = 20;
```

In this case, the `UPDATE` statement would have no effect on the slave when using statement-based replication. However, if you are using row-based replication, the `UPDATE` would change `table1` on the slave, but not `table2`—in other words, only tables in the database named by `--replicate-do-db` are changed, and the choice of default database has no effect on this behavior.

If you need cross-database updates to work, use `--replicate-wild-do-table=`*`db_name`*`.%` instead. See Section 16.2.3, “How Servers Evaluate Replication Filtering Rules”.

Note

This option affects replication in the same manner that `--binlog-do-db` affects binary logging, and the effects of the replication format on how `--replicate-do-db` affects replication behavior are the same as those of the logging format on the behavior of `--binlog-do-db`.

This option has no effect on `BEGIN`, `COMMIT`, or `ROLLBACK` statements.

- `--replicate-ignore-db=`*`db_name`*

Command-Line Format	`--replicate-ignore-db=name`	
Option-File Format	`replicate-ignore-db`	
	Permitted Values	
	Type	`string`

Creates a replication filter using the name of a database. In MySQL 5.7.3 and later, such filters can also be created using `CHANGE REPLICATION FILTER REPLICATE_IGNORE_DB`. As with `--replicate-do-db`, the precise effect of this filtering depends on whether statement-based or row-based replication is in use, and are described in the next several paragraphs.

Statement-based replication. Tells the slave SQL thread not to replicate any statement where the default database (that is, the one selected by `USE`) is *`db_name`*.

Row-based replication. Tells the slave SQL thread not to update any tables in the database *`db_name`*. The default database has no effect.

When using statement-based replication, the following example does not work as you might expect. Suppose that the slave is started with `--replicate-ignore-db=sales` and you issue the following statements on the master:

```
USE prices;
UPDATE sales.january SET amount=amount+1000;
```

The `UPDATE` statement *is* replicated in such a case because `--replicate-ignore-db` applies only to the default database (determined by the `USE` statement). Because the `sales` database was specified explicitly in the statement, the statement has not been filtered. However, when using row-based replication, the `UPDATE` statement's effects are *not* propagated to the slave, and the slave's copy of the `sales.january` table is unchanged; in this instance, `--replicate-ignore-db=sales` causes *all* changes made to tables in the master's copy of the `sales` database to be ignored by the slave.

To specify more than one database to ignore, use this option multiple times, once for each database. Because database names can contain commas, if you supply a comma separated list then the list will be treated as the name of a single database.

You should not use this option if you are using cross-database updates and you do not want these updates to be replicated. See Section 16.2.3, “How Servers Evaluate Replication Filtering Rules”.

If you need cross-database updates to work, use `--replicate-wild-ignore-table=`*`db_name`*`.%` instead. See Section 16.2.3, “How Servers Evaluate Replication Filtering Rules”.

> **Note**
>
> This option affects replication in the same manner that `--binlog-ignore-db` affects binary logging, and the effects of the replication format on how `--replicate-ignore-db` affects replication behavior are the same as those of the logging format on the behavior of `--binlog-ignore-db`.
>
> This option has no effect on `BEGIN`, `COMMIT`, or `ROLLBACK` statements.

- `--replicate-do-table=`*`db_name.tbl_name`*

Command-Line Format	`--replicate-do-table=name`	
Option-File Format	`replicate-do-table`	
	Permitted Values	
	Type	`string`

Creates a replication filter by telling the slave SQL thread to restrict replication to a given table. To specify more than one table, use this option multiple times, once for each table. This works for both cross-database updates and default database updates, in contrast to `--replicate-do-db`. See Section 16.2.3, “How Servers Evaluate Replication Filtering Rules”.

In MySQL 5.7.3 and later, you can also create such a filter by issuing a CHANGE REPLICATION FILTER REPLICATE_DO_TABLE statement.

This option affects only statements that apply to tables. It does not affect statements that apply only to other database objects, such as stored routines. To filter statements operating on stored routines, use one or more of the --replicate-*-db options.

- --replicate-ignore-table=*db_name.tbl_name*

Command-Line Format	--replicate-ignore-table=name	
Option-File Format	replicate-ignore-table	
	Permitted Values	
	Type	string

Creates a replication filter by telling the slave SQL thread not to replicate any statement that updates the specified table, even if any other tables might be updated by the same statement. To specify more than one table to ignore, use this option multiple times, once for each table. This works for cross-database updates, in contrast to --replicate-ignore-db. See Section 16.2.3, “How Servers Evaluate Replication Filtering Rules”.

In MySQL 5.7.3 and later, you can also create such a filter by issuing a CHANGE REPLICATION FILTER REPLICATE_IGNORE_TABLE statement.

This option affects only statements that apply to tables. It does not affect statements that apply only to other database objects, such as stored routines. To filter statements operating on stored routines, use one or more of the --replicate-*-db options.

- --replicate-rewrite-db=*from_name*->*to_name*

Command-Line Format	--replicate-rewrite-db=old_name->new_name	
Option-File Format	replicate-rewrite-db	
	Permitted Values	
	Type	string

Tells the slave to create a replication filter that translates the default database (that is, the one selected by USE) to *to_name* if it was *from_name* on the master. Only statements involving tables are affected (not statements such as CREATE DATABASE, DROP DATABASE, and ALTER DATABASE), and only if *from_name* is the default database on the master. To specify multiple rewrites, use this option multiple times. The server uses the first one with a *from_name* value that matches. The database name translation is done *before* the --replicate-* rules are tested.

In MySQL 5.7.3 and later, you can also create such a filter by issuing a CHANGE REPLICATION FILTER REPLICATE_REWRITE_DB statement.

Statements in which table names are qualified with database names when using this option do not work with table-level replication filtering options such as --replicate-do-table. Suppose we have a database named a on the master, one named b on the slave, each containing a table t, and have started the master with --replicate-rewrite-db='a->b'. At a later point in time, we execute DELETE FROM a.t. In this case, no relevant filtering rule works, for the reasons shown here:

1. --replicate-do-table=a.t does not work because the slave has table t in database b.

2. `--replicate-do-table=b.t` does not match the original statement and so is ignored.
3. `--replicate-do-table=*.t` is handled identically to `--replicate-do-table=a.t`, and thus does not work, either.

Similarly, the `--replication-rewrite-db` option does not work with cross-database updates.

If you use this option on the command line and the “`>`” character is special to your command interpreter, quote the option value. For example:

```
shell> mysqld --replicate-rewrite-db="olddb->newdb"
```

- `--replicate-same-server-id`

Command-Line Format	`--replicate-same-server-id`	
Option-File Format	`replicate-same-server-id`	
	Permitted Values	
	Type	`boolean`
	Default	`FALSE`

To be used on slave servers. Usually you should use the default setting of 0, to prevent infinite loops caused by circular replication. If set to 1, the slave does not skip events having its own server ID. Normally, this is useful only in rare configurations. Cannot be set to 1 if `--log-slave-updates` is used. By default, the slave I/O thread does not write binary log events to the relay log if they have the slave's server ID (this optimization helps save disk usage). If you want to use `--replicate-same-server-id`, be sure to start the slave with this option before you make the slave read its own events that you want the slave SQL thread to execute.

- `--replicate-wild-do-table=db_name.tbl_name`

Command-Line Format	`--replicate-wild-do-table=name`	
Option-File Format	`replicate-wild-do-table`	
	Permitted Values	
	Type	`string`

Creates a replication filter by telling the slave thread to restrict replication to statements where any of the updated tables match the specified database and table name patterns. Patterns can contain the “`%`” and “`_`” wildcard characters, which have the same meaning as for the `LIKE` pattern-matching operator. To specify more than one table, use this option multiple times, once for each table. This works for cross-database updates. See Section 16.2.3, “How Servers Evaluate Replication Filtering Rules”.

In MySQL 5.7.3 and later, you can also create such a filter by issuing a `CHANGE REPLICATION FILTER REPLICATE_WILD_DO_TABLE` statement.

This option applies to tables, views, and triggers. It does not apply to stored procedures and functions, or events. To filter statements operating on the latter objects, use one or more of the `--replicate-*-db` options.

Example: `--replicate-wild-do-table=foo%.bar%` replicates only updates that use a table where the database name starts with `foo` and the table name starts with `bar`.

If the table name pattern is `%`, it matches any table name and the option also applies to database-level statements (`CREATE DATABASE`, `DROP DATABASE`, and `ALTER DATABASE`). For example, if you use `--replicate-wild-do-table=foo%.%`, database-level statements are replicated if the database name matches the pattern `foo%`.

To include literal wildcard characters in the database or table name patterns, escape them with a backslash. For example, to replicate all tables of a database that is named `my_own%db`, but not replicate tables from the `my1ownAABCdb` database, you should escape the "`_`" and "`%`" characters like this: `--replicate-wild-do-table=my\_own\%db`. If you use the option on the command line, you might need to double the backslashes or quote the option value, depending on your command interpreter. For example, with the `bash` shell, you would need to type `--replicate-wild-do-table=my\\_own\\%db`.

- `--replicate-wild-ignore-table=`*`db_name.tbl_name`*

Command-Line Format	`--replicate-wild-ignore-table=name`	
Option-File Format	`replicate-wild-ignore-table`	
	Permitted Values	
	Type	`string`

Creates a replication filter which keeps the slave thread from replicating a statement in which any table matches the given wildcard pattern. To specify more than one table to ignore, use this option multiple times, once for each table. This works for cross-database updates. See Section 16.2.3, "How Servers Evaluate Replication Filtering Rules".

In MySQL 5.7.3 and later, you can also create such a filter by issuing a `CHANGE REPLICATION FILTER REPLICATE_WILD_IGNORE_TABLE` statement.

Example: `--replicate-wild-ignore-table=foo%.bar%` does not replicate updates that use a table where the database name starts with `foo` and the table name starts with `bar`.

For information about how matching works, see the description of the `--replicate-wild-do-table` option. The rules for including literal wildcard characters in the option value are the same as for `--replicate-wild-ignore-table` as well.

- `--report-host=`*`host_name`*

Command-Line Format	`--report-host=host_name`	
Option-File Format	`report-host`	
System Variable Name	`report_host`	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	`string`

The host name or IP address of the slave to be reported to the master during slave registration. This value appears in the output of `SHOW SLAVE HOSTS` on the master server. Leave the value unset if you do not want the slave to register itself with the master. Note that it is not sufficient for the master to simply read the IP address of the slave from the TCP/IP socket after the slave connects. Due to NAT and other routing issues, that IP may not be valid for connecting to the slave from the master or other hosts.

- `--report-password=password`

Command-Line Format	`--report-password=name`	
Option-File Format	`report-password`	
System Variable Name	`report_password`	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	`string`

The account password of the slave to be reported to the master during slave registration. This value appears in the output of `SHOW SLAVE HOSTS` on the master server if the `--show-slave-auth-info` option is given.

Although the name of this option might imply otherwise, `--report-password` is not connected to the MySQL user privilege system and so is not necessarily (or even likely to be) the same as the password for the MySQL replication user account.

- `--report-port=slave_port_num`

Command-Line Format	`--report-port=#`	
Option-File Format	`report-port`	
System Variable Name	`report_port`	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	`numeric`
	Default	`[slave_port]`
	Range	`0 .. 65535`

The TCP/IP port number for connecting to the slave, to be reported to the master during slave registration. Set this only if the slave is listening on a nondefault port or if you have a special tunnel from the master or other clients to the slave. If you are not sure, do not use this option.

The default value for this option is the port number actually used by the slave (Bug #13333431). This is also the default value displayed by `SHOW SLAVE HOSTS`.

- `--report-user=user_name`

Command-Line Format	`--report-user=name`	
Option-File Format	`report-user`	
System Variable Name	`report_user`	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	`string`

The account user name of the slave to be reported to the master during slave registration. This value appears in the output of `SHOW SLAVE HOSTS` on the master server if the `--show-slave-auth-info` option is given.

Although the name of this option might imply otherwise, `--report-user` is not connected to the MySQL user privilege system and so is not necessarily (or even likely to be) the same as the name of the MySQL replication user account.

- `--show-slave-auth-info`

Command-Line Format	`--show-slave-auth-info`	
Option-File Format	`show-slave-auth-info`	
	Permitted Values	
	Type	`boolean`
	Default	`FALSE`

Display slave user names and passwords in the output of `SHOW SLAVE HOSTS` on the master server for slaves started with the `--report-user` and `--report-password` options.

- `--slave-checkpoint-group=#`

Command-Line Format	`--slave-checkpoint-group=#`	
Option-File Format	`slave-checkpoint-group`	
	Permitted Values	
	Type	`numeric`
	Default	`512`
	Range	`32 .. 524280`
	Block Size	`8`

Sets the maximum number of transactions that can be processed by a multi-threaded slave before a checkpoint operation is called to update its status as shown by `SHOW SLAVE STATUS`. Setting this option has no effect on slaves for which multi-threading is not enabled.

This option works in combination with the `--slave-checkpoint-period` option in such a way that, when either limit is exceeded, the checkpoint is executed and the counters tracking both the number of transactions and the time elapsed since the last checkpoint are reset.

The minimum allowed value for this option is 32, unless the server was built using `-DWITH_DEBUG`, in which case the minimum value is 1. The effective value is always a multiple of 8; you can set it to a value that is not such a multiple, but the server rounds it down to the next lower multiple of 8 before storing the value. (*Exception*: No such rounding is performed by the debug server.) Regardless of how the server was built, the default value is 512, and the maximum allowed value is 524280.

- `--slave-checkpoint-period=#`

Command-Line Format	`--slave-checkpoint-period=#`	
Option-File Format	`slave-checkpoint-period`	
	Permitted Values	

	Type	numeric
	Default	300
	Range	1 .. 4G

Sets the maximum time (in milliseconds) that is allowed to pass before a checkpoint operation is called to update the status of a multi-threaded slave as shown by SHOW SLAVE STATUS. Setting this option has no effect on slaves for which multi-threading is not enabled.

This option works in combination with the --slave-checkpoint-group option in such a way that, when either limit is exceeded, the checkpoint is executed and the counters tracking both the number of transactions and the time elapsed since the last checkpoint are reset.

The minimum allowed value for this option is 1, unless the server was built using -DWITH_DEBUG, in which case the minimum value is 0. Regardless of how the server was built, the default value is 300, and the maximum possible value is 4294967296 (4GB).

- --slave-parallel-workers

Command-Line Format	--slave-parallel-workers=#	
Option-File Format	slave-parallel-workers	
	Permitted Values	
	Type	numeric
	Default	0
	Range	0 .. 1024

Sets the number of slave worker threads for executing replication events (transactions) in parallel. Setting this variable to 0 (the default) disables parallel execution. The maximum is 1024.

When parallel execution is enabled, the slave SQL thread acts as the coordinator for the slave worker threads, among which transactions are distributed on a per-database basis. This means that a worker thread on the slave slave can process successive transactions on a given database without waiting for updates to other databases to complete. The current implementation of multi-threading on the slave assumes that the data is partitioned per database, and that updates within a given database occur in the same relative order as they do on the master, in order to work correctly. However, transactions do not need to be coordinated between any two databases.

Due to the fact that transactions on different databases can occur in a different order on the slave than on the master, checking for the most recently executed transaction does not guarantee that all previous transactions from the master have been executed on the slave. This has implications for logging and recovery when using a multi-threaded slave. For information about how to interpret binary logging information when using multi-threading on the slave, see Section 13.7.5.33, “SHOW SLAVE STATUS Syntax”. In addition, this means that START SLAVE UNTIL is not supported with a multi-threaded slave.

When multi-threading is enabled, slave_transaction_retries is treated as equal to 0, and cannot be changed. (Currently, retrying of transactions is not supported with multi-threaded slaves.)

You should also note that enforcing foreign key relationships between tables in different databases causes multi-threaded slaves to use sequential rather than parallel mode, which can have a negative impact on performance. (Bug #14092635)

- --slave-pending-jobs-size-max=#

Command-Line Format	`--slave-pending-jobs-size-max=#`	
	Permitted Values	
	Type	`numeric`
	Default	`16M`
	Range	`1024 .. 18EB`
	Block Size	`1024`

For multi-threaded slaves, this option sets the maximum amount of memory (in bytes) available to slave worker queues holding events not yet applied. Setting this option has no effect on slaves for which multi-threading is not enabled.

The minimum possible value for this option is 1024; the default is 16MB. The maximum possible value is 18446744073709551615 (16 exabytes). Values that are not exact multiples of 1024 are rounded down to the next-highest multiple of 1024 prior to being stored.

> **Important**
>
> The value for this option must not be less than the master's value for `max_allowed_packet`; otherwise a slave worker queue may become full while there remain events coming from the master to be processed.

- `--skip-slave-start`

Command-Line Format	`--skip-slave-start`	
Option-File Format	`skip-slave-start`	
	Permitted Values	
	Type	`boolean`
	Default	`FALSE`

Tells the slave server not to start the slave threads when the server starts. To start the threads later, use a `START SLAVE` statement.

- `--slave_compressed_protocol={0|1}`

Command-Line Format	`--slave_compressed_protocol`	
Option-File Format	`slave_compressed_protocol`	
System Variable Name	`slave_compressed_protocol`	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	`boolean`
	Default	`OFF`

If this option is set to 1, use compression for the slave/master protocol if both the slave and the master support it. The default is 0 (no compression).

- `--slave-load-tmpdir=`*`file_name`*

Command-Line Format	`--slave-load-tmpdir=path`	
Option-File Format	`slave-load-tmpdir`	
System Variable Name	`slave_load_tmpdir`	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	`directory name`
	Default	`/tmp`

The name of the directory where the slave creates temporary files. This option is by default equal to the value of the `tmpdir` system variable. When the slave SQL thread replicates a `LOAD DATA INFILE` statement, it extracts the file to be loaded from the relay log into temporary files, and then loads these into the table. If the file loaded on the master is huge, the temporary files on the slave are huge, too. Therefore, it might be advisable to use this option to tell the slave to put temporary files in a directory located in some file system that has a lot of available space. In that case, the relay logs are huge as well, so you might also want to use the `--relay-log` option to place the relay logs in that file system.

The directory specified by this option should be located in a disk-based file system (not a memory-based file system) because the temporary files used to replicate `LOAD DATA INFILE` must survive machine restarts. The directory also should not be one that is cleared by the operating system during the system startup process.

- `slave-max-allowed-packet=`*`bytes`*

Command-Line Format	`--slave-max-allowed-packet=#`	
Option-File Format	`slave-max-allowed-packet`	
	Permitted Values	
	Type	`numeric`
	Default	`1073741824`
	Range	`1024 .. 1073741824`

This option sets the maximum packet size in bytes for the slave SQL and I/O threads, so that large updates using row-based replication do not cause replication to fail because an update exceeded `max_allowed_packet`. (Bug #12400221, Bug #60926)

The corresponding server variable `slave_max_allowed_packet` always has a value that is a positive integer multiple of 1024; if you set it to some value that is not such a multiple, the value is automatically rounded down to the next highest multiple of 1024. (For example, if you start the server with `--slave-max-allowed-packet=10000`, the value used is 9216; setting 0 as the value causes 1024 to be used.) A truncation warning is issued in such cases.

The maximum (and default) value is 1073741824 (1 GB); the minimum is 1024.

- `--slave-net-timeout=`*`seconds`*

Command-Line Format	`--slave-net-timeout=#`
Option-File Format	`slave-net-timeout`
System Variable Name	`slave_net_timeout`

Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	numeric
	Default	3600
	Min Value	1

The number of seconds to wait for more data from the master before the slave considers the connection broken, aborts the read, and tries to reconnect. The first retry occurs immediately after the timeout. The interval between retries is controlled by the `MASTER_CONNECT_RETRY` option for the `CHANGE MASTER TO` statement, and the number of reconnection attempts is limited by the `--master-retry-count` option. The default is 3600 seconds (one hour).

- `--slave-parallel-type=`*`type`*

Introduced	5.7.2	
Command-Line Format	`--slave-parallel-type=type`	
Option-File Format	`slave-parallel-type`	
	Permitted Values	
	Type	enumeration
	Default	DATABASE
	Valid Values	DATABASE
		LOGICAL_CLOCK

Normally, transactions are applied in parallel only if they do not make changes in the same database. Beginning with MySQL 5.7.2, is it possible to enable parallel execution on the slave of all uncommitted threads already in the prepare phase, without violating consistency, by starting the slave with `--slave-parallel-type=LOGICAL_CLOCK`, or by setting the `slave_parallel_type` system variable.

When this feature is enabled, each transaction is marked with a logical timestamp generated by the master. The timestamp identifies the last transaction committed at the time that the current transaction entered the prepare stage, and all transactions having the same timestamp can execute in parallel.

This option was added in MySQL 5.7.2. The default value is `DATABASE`.

- `slave-rows-search-algorithms=`*`list`*

Command-Line Format	`--slave-rows-search-algorithms=list`	
Option-File Format	`slave-rows-search-algorithms`	
	Permitted Values	
	Type	set
	Default	TABLE_SCAN,INDEX_SCAN
	Valid Values	TABLE_SCAN,INDEX_SCAN
		INDEX_SCAN,HASH_SCAN
		TABLE_SCAN,HASH_SCAN

		TABLE_SCAN,INDEX_SCAN,HASH_SCAN (equivalent to INDEX_SCAN,HASH_SCAN)

When preparing batches of rows for row-based logging and replication using slave_allow_batching, this option controls how the rows are searched for matches—that is, whether or not hashing is used for searches using a primary or unique key, some other key, or no key at all. This option takes a comma-separated list of any 2 (or possibly 3) values from the list INDEX_SCAN, TABLE_SCAN, HASH_SCAN. The list need not be quoted, but must contain no spaces, whether or not quotes are used. Possible combinations (lists) and their effects are shown in the following table:

Index used / option value	**INDEX_SCAN,HASH_SCAN or INDEX_SCAN,TABLE_SCAN,HASH_SCAN**	**INDEX_SCAN,TABLE_SCAN**	**TABLE_SCAN,HASH_SCAN**
Primary key or unique key	Index scan	Index scan	Hash scan over index
(Other) Key	Hash scan over index	Index scan	Hash scan over index
No index	Hash scan	Table scan	Hash scan

The order in which the algorithms are specified in the list does not make any difference in the order in which they are displayed by a SELECT or SHOW VARIABLES statement (which is the same as that used in the table just shown previously).The default value is TABLE_SCAN,INDEX_SCAN, which means that all searches that can use indexes do use them, and searches without any indexes use table scans.

Specifying INDEX_SCAN,TABLE_SCAN,HASH_SCAN has the same effect as specifying INDEX_SCAN,HASH_SCAN. To use hashing for any searches that does not use a primary or unique key, set this option to INDEX_SCAN,HASH_SCAN. To force hashing for *all* searches, set it to TABLE_SCAN,HASH_SCAN.

- --slave-skip-errors=[*err_code1*,*err_code2*,...|all|ddl_exist_errors]

Command-Line Format	--slave-skip-errors=name	
Option-File Format	slave-skip-errors	
System Variable Name	slave_skip_errors	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	string
	Default	OFF
	Valid Values	OFF
		[list of error codes]
		all
		ddl_exist_errors

Normally, replication stops when an error occurs on the slave. This gives you the opportunity to resolve the inconsistency in the data manually. This option tells the slave SQL thread to continue replication when a statement returns any of the errors listed in the option value.

Do not use this option unless you fully understand why you are getting errors. If there are no bugs in your replication setup and client programs, and no bugs in MySQL itself, an error that stops replication should never occur. Indiscriminate use of this option results in slaves becoming hopelessly out of synchrony with the master, with you having no idea why this has occurred.

For error codes, you should use the numbers provided by the error message in your slave error log and in the output of `SHOW SLAVE STATUS`. Appendix C, *Errors, Error Codes, and Common Problems*, lists server error codes.

You can also (but should not) use the very nonrecommended value of `all` to cause the slave to ignore all error messages and keeps going regardless of what happens. Needless to say, if you use `all`, there are no guarantees regarding the integrity of your data. Please do not complain (or file bug reports) in this case if the slave's data is not anywhere close to what it is on the master. *You have been warned*.

MySQL 5.7 supports an additional shorthand value `ddl_exist_errors`, which is equivalent to the error code list `1007,1008,1050,1051,1054,1060,1061,1068,1094,1146`.

Examples:

```
--slave-skip-errors=1062,1053
--slave-skip-errors=all
--slave-skip-errors=ddl_exist_errors
```

- `--slave-sql-verify-checksum={0|1}`

Command-Line Format	`--slave-sql-verify-checksum=value`	
Option-File Format	`slave-sql-verify-checksum`	
	Permitted Values	
	Type	`boolean`
	Default	`0`

	Valid Values	`0`
		`1`

When this option is enabled, the slave examines checksums read from the relay log, in the event of a mismatch, the slave stops with an error. Disabled by default.

Obsolete options. *The following options were removed in MySQL 5.6 and are no longer supported. If you attempt to start `mysqld` with any of these options in MySQL 5.7, the server aborts with an `unknown variable` error*. To set the replication parameters formerly associated with these options, you must use the `CHANGE MASTER TO ...` statement (see Section 13.4.2.1, "`CHANGE MASTER TO` Syntax").

The options affected are shown in this list:

- `--master-host`
- `--master-user`
- `--master-password`
- `--master-port`
- `--master-connect-retry`
- `--master-ssl`
- `--master-ssl-ca`
- `--master-ssl-capath`

- `--master-ssl-cert`
- `--master-ssl-cipher`
- `--master-ssl-key`

System variables used on replication slaves. The following list describes system variables for controlling replication slave servers. They can be set at server startup and some of them can be changed at runtime using `SET`. Server options used with replication slaves are listed earlier in this section.

- `slave_allow_batching`

Command-Line Format	`--slave-allow-batching`	
Option-File Format	`slave_allow_batching`	
System Variable Name	`slave_allow_batching`	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	`boolean`
	Default	`off`

Whether or not batched updates are enabled on replication slaves.

- `init_slave`

Command-Line Format	`--init-slave=name`	
Option-File Format	`init_slave`	
System Variable Name	`init_slave`	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	`string`

This variable is similar to `init_connect`, but is a string to be executed by a slave server each time the SQL thread starts. The format of the string is the same as for the `init_connect` variable.

> **Note**
>
> The SQL thread sends an acknowledgment to the client before it executes `init_slave`. Therefore, it is not guaranteed that `init_slave` has been executed when `START SLAVE` returns. See Section 13.4.2.6, "`START SLAVE` Syntax", for more information.

- `log_slow_slave_statements`

Introduced	5.7.1
System Variable Name	`log_slow_slave_statements`
Variable Scope	Global
Dynamic Variable	Yes

	Permitted Values	
	Type	boolean
	Default	OFF

When the slow query log is enabled, this variable enables logging for queries that have taken more than long_query_time seconds to execute on the slave. This variable was added in MySQL 5.7.1.

- master_info_repository

Command-Line Format	--master-info-repository=FILE|TABLE	
Option-File Format	master_info_repository	
System Variable Name	master_info_repository	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	string
	Default	FILE
	Valid Values	FILE
		TABLE

The setting of this variable determines whether the slave logs master status and connection information to a FILE (master.info), or to a TABLE (mysql.slave_master_info).

The setting of this variable also has a direct bearing on the effect had by the setting of the sync_master_info system variable; see that variable's description for further information.

- relay_log

Command-Line Format	--relay-log=name	
Option-File Format	relay-log	
System Variable Name	relay_log	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	file name

The name of the relay log file.

- relay_log_basename

System Variable Name	relay_log_basename	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	file name
	Default	datadir + '/' + hostname + '-relay-bin'

Holds the name and complete path to the relay log file.

- `relay_log_index`

Command-Line Format	`--relay-log-index`	
Option-File Format	`relay_log_index`	
System Variable Name	`relay_log_index`	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	`file name`
	Default	`*host_name*-relay-bin.index`

The name of the relay log index file. The default name is `host_name-relay-bin.index` in the data directory, where `host_name` is the name of the slave server.

- `relay_log_info_file`

Command-Line Format	`--relay-log-info-file=file_name`	
Option-File Format	`relay_log_info_file`	
System Variable Name	`relay_log_info_file`	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	`file name`
	Default	`relay-log.info`

The name of the file in which the slave records information about the relay logs. The default name is `relay-log.info` in the data directory.

- `relay_log_info_repository`

System Variable Name	`relay_log_info_repository`	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	`string`
	Default	`FILE`
	Valid Values	`FILE`
		`TABLE`

This variable determines whether the slave's position in the relay logs is written to a `FILE` (`relay-log.info`) or to a `TABLE` (`mysql.slave_relay_log_info`).

The setting of this variable also has a direct bearing on the effect had by the setting of the `sync_relay_log_info` system variable; see that variable's description for further information.

- `relay_log_recovery`

Command-Line Format	`--relay-log-recovery`	
Option-File Format	`relay_log_recovery`	
System Variable Name	`relay_log_recovery`	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	`boolean`
	Default	`FALSE`

Enables automatic relay log recovery immediately following server startup, which means that the replication slave discards all unprocessed relay logs and retrieves them from the replication master. In MySQL 5.7, this global variable is read-only; its value can be changed by starting the slave with the `--relay-log-recovery` option, which should be used following a crash on the replication slave to ensure that no possibly corrupted relay logs are processed, and must be used in order to guarantee a crash-proof slave. The default value is 0 (disabled).

When `relay_log_recovery` is enabled and the slave has stopped due to errors encountered while running in multi-threaded mode, you can use `START SLAVE UNTIL SQL_AFTER_MTS_GAPS` to ensure that all gaps are processed before switching back to single-threaded mode or executing a `CHANGE MASTER TO` statement.

- `rpl_stop_slave_timeout`

Introduced	5.7.2	
Command-Line Format	`--rpl-stop-slave-timeout=seconds`	
Option-File Format	`rpl_stop_slave_timeout`	
System Variable Name	`rpl_stop_slave_timeout`	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	`integer`
	Default	`31536000`
	Range	`2 .. 31536000`

In MySQL 5.7.2 and later, you can control the length of time (in seconds) that `STOP SLAVE` waits before timing out by setting this variable. This can be used to avoid deadlocks between `STOP SLAVE` and other slave SQL statements using different client connections to the slave. The maximum and default value of `rpl_stop_slave_timeout` is 31536000 seconds (1 year). The minimum is 2 seconds.

- `slave_checkpoint_group`

Command-Line Format	`--slave-checkpoint-group=#`
Option-File Format	`slave_checkpoint_group`
System Variable Name	`slave_checkpoint_group=#`
Variable Scope	Global

Dynamic Variable	Yes	
	Permitted Values	
	Type	`numeric`
	Default	`512`
	Range	`32 .. 524280`
	Block Size	`8`

Sets the maximum number of transactions that can be processed by a multi-threaded slave before a checkpoint operation is called to update its status as shown by `SHOW SLAVE STATUS`. Setting this variable has no effect on slaves for which multi-threading is not enabled.

This variable works in combination with the `slave_checkpoint_period` system variable in such a way that, when either limit is exceeded, the checkpoint is executed and the counters tracking both the number of transactions and the time elapsed since the last checkpoint are reset.

The minimum allowed value for this variable is 32, unless the server was built using `-DWITH_DEBUG`, in which case the minimum value is 1. The effective value is always a multiple of 8; you can set it to a value that is not such a multiple, but the server rounds it down to the next lower multiple of 8 before storing the value. (*Exception*: No such rounding is performed by the debug server.) Regardless of how the server was built, the default value is 512, and the maximum allowed value is 524280.

- `slave_checkpoint_period`

Command-Line Format	`--slave-checkpoint-period=#`	
Option-File Format	`slave_checkpoint_period`	
System Variable Name	`slave_checkpoint_period=#`	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	`numeric`
	Default	`300`
	Range	`1 .. 4G`

Sets the maximum time (in milliseconds) that is allowed to pass before a checkpoint operation is called to update the status of a multi-threaded slave as shown by `SHOW SLAVE STATUS`. Setting this variable has no effect on slaves for which multi-threading is not enabled.

This variable works in combination with the `slave_checkpoint_group` system variable in such a way that, when either limit is exceeded, the checkpoint is executed and the counters tracking both the number of transactions and the time elapsed since the last checkpoint are reset.

The minimum allowed value for this variable is 1, unless the server was built using `-DWITH_DEBUG`, in which case the minimum value is 0. Regardless of how the server was built, the default value is 300, and the maximum possible value is 4294967296 (4GB).

- `slave_compressed_protocol`

Command-Line Format	`--slave_compressed_protocol`

Option-File Format	`slave_compressed_protocol`	
System Variable Name	`slave_compressed_protocol`	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	`boolean`
	Default	`OFF`

Whether to use compression of the slave/master protocol if both the slave and the master support it.

- `slave_exec_mode`

Command-Line Format	`--slave-exec-mode=mode`	
Option-File Format	`slave_exec_mode`	
System Variable Name	`slave_exec_mode`	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	`enumeration`
	Default	`STRICT` (ALL)
	Default	`IDEMPOTENT` (NDB)
	Valid Values	`IDEMPOTENT`
		`STRICT`

Controls whether `IDEMPOTENT` or `STRICT` mode is used in replication conflict resolution and error checking. `IDEMPOTENT` mode causes suppression of duplicate-key and no-key-found errors. This mode should be employed in multi-master replication, circular replication, and some other special replication scenarios. `STRICT` mode is the default, and is suitable for most other cases.

- `slave_load_tmpdir`

Command-Line Format	`--slave-load-tmpdir=path`	
Option-File Format	`slave-load-tmpdir`	
System Variable Name	`slave_load_tmpdir`	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	`directory name`
	Default	`/tmp`

The name of the directory where the slave creates temporary files for replicating `LOAD DATA INFILE` statements.

- `slave_max_allowed_packet`

System Variable Name	slave_max_allowed_packet	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	numeric
	Default	1073741824
	Range	1024 .. 1073741824

In MySQL 5.6.6 and later, this variable sets the maximum packet size for the slave SQL and I/O threads, so that large updates using row-based replication do not cause replication to fail because an update exceeded max_allowed_packet.

This global variable always has a value that is a positive integer multiple of 1024; if you set it to some value that is not, the value is rounded down to the next highest multiple of 1024 for it is stored or used; setting slave_max_allowed_packet to 0 causes 1024 to be used. (A truncation warning is issued in all such cases.) The default and maximum value is 1073741824 (1 GB); the minimum is 1024.

slave_max_allowed_packet can also be set at startup, using the --slave-max-allowed-packet option.

- slave_net_timeout

Command-Line Format	--slave-net-timeout=#	
Option-File Format	slave-net-timeout	
System Variable Name	slave_net_timeout	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	numeric
	Default	3600
	Min Value	1

The number of seconds to wait for more data from a master/slave connection before aborting the read.

- slave_parallel_type=*type*

Introduced	5.7.2	
Command-Line Format	--slave-parallel-type=type	
Option-File Format	slave-parallel-type	
	Permitted Values	
	Type	enumeration
	Default	DATABASE
	Valid Values	DATABASE
		LOGICAL_CLOCK

Normally, transactions are applied in parallel only if they do not make any changes in the same database. Beginning with MySQL 5.7.2, it is possible to enable parallel execution on the slave of all uncommitted threads already in the prepare phase, without violating consistency, using the SQL statement shown here:

```
SET @@global.slave_parallel_type='LOGICAL_CLOCK';
```

You can set `slave_parallel_type` in a running MySQL server only when the slave is stopped; that is, before issuing `START SLAVE` or after issuing `STOP SLAVE`.

When parallel execution of prepared transactions is enabled, each transaction is marked with a logical timestamp by the master. This timestamp identifies the last transaction committed at the time that the current transaction entered the prepare stage, and all transactions having the same timestamp can execute in parallel. You can also enable this feature using the `--slave-parallel-type` option when starting the MySQL Server.

To disable parallel execution of prepared transactions in a running server, set `slave_parallel_type` to `'DATABASE'` (the default value).

This variable is global. In addition, when setting `slave_parallel_type`, the value (one of `'DATABASE'` or `'LOGICAL_CLOCK'`) must be quoted.

- `slave_parallel_workers`

System Variable Name	`slave_parallel_workers`	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	`numeric`
	Default	`0`
	Range	`0 .. 1024`

Sets the number of slave worker threads for executing replication events (transactions) in parallel. Setting this variable to 0 (the default) disables parallel execution. The maximum is 1024.

When parallel execution is enabled, the slave SQL thread acts as the coordinator for the slave worker threads, among which transactions are distributed on a per-database basis. This means that a worker thread on the slave slave can process successive transactions on a given database without waiting for updates to other databases to complete. The current implementation of multi-threading on the slave assumes that the data is partitioned per database, and that updates within a given database occur in the same relative order as they do on the master, in order to work correctly. However, transactions do not need to be coordinated between any two databases.

Due to the fact that transactions on different databases can occur in a different order on the slave than on the master, checking for the most recently executed transaction does not guarantee that all previous transactions from the master have been executed on the slave. This has implications for logging and recovery when using a multi-threaded slave. For information about how to interpret binary logging information when using multi-threading on the slave, see Section 13.7.5.33, “`SHOW SLAVE STATUS` Syntax”. In addition, this means that `START SLAVE UNTIL` is not supported with a multi-threaded slave.

When multi-threading is enabled, slave_transaction_retries is treated as equal to 0, and cannot be changed. (Currently, retrying of transactions is not supported with multi-threaded slaves.)

- slave_pending_jobs_size_max

System Variable Name	slave_pending_jobs_size_max	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	numeric
	Default	0
	Range	1024 .. 18EB
	Block Size	1024

For multi-threaded slaves, this variable sets the maximum amount of memory (in bytes) available to slave worker queues holding events not yet applied. Setting this variable has no effect on slaves for which multi-threading is not enabled.

The minimum possible value for this variable is 1024; the default is 16MB. The maximum possible value is 18446744073709551615 (16 exabytes). Values that are not exact multiples of 1024 are rounded down to the next-highest multiple of 1024 prior to being stored.

> **Important**
>
> The value of this variable must not be less than the master's value for max_allowed_packet; otherwise a slave worker queue may become full while there remain events coming from the master to be processed.

- slave_rows_search_algorithms

System Variable Name	slave_rows_search_algorithms=list	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	set
	Default	TABLE_SCAN,INDEX_SCAN
	Valid Values	TABLE_SCAN,INDEX_SCAN
		INDEX_SCAN,HASH_SCAN
		TABLE_SCAN,HASH_SCAN
		TABLE_SCAN,INDEX_SCAN,HASH_SCAN (equivalent to INDEX_SCAN,HASH_SCAN)

When preparing batches of rows for row-based logging and replication using slave_allow_batching, the slave_rows_search_algorithms variable controls how the rows are searched for matches—that is, whether or not hashing is used for searches using a primary or unique key, using some other key, or using no key at all. This option takes a comma-separated list of at least 2 values from the list INDEX_SCAN, TABLE_SCAN, HASH_SCAN. The value expected as a string, so the value must be quoted.

In addition, the value must not contain any spaces. Possible combinations (lists) and their effects are shown in the following table:

Index used / option value	**INDEX_SCAN,HASH_SCAN or INDEX_SCAN,TABLE_SCAN,HASH_SCAN**	**INDEX_SCAN,TABLE_SCAN**	**TABLE_SCAN,HASH_SCAN**
Primary key or unique key	Index scan	index scan	Index hash
(Other) Key	Index hash	Index scan	Index hash
No index	Table hash	Table scan	Table hash

The order in which the algorithms are specified in the list does not make any difference in the order in which they are displayed by a SELECT or SHOW VARIABLES statement, as shown here:

```
mysql> SET GLOBAL slave_rows_search_algorithms = "INDEX_SCAN,TABLE_SCAN";
Query OK, 0 rows affected (0.00 sec)

mysql> SHOW VARIABLES LIKE '%algorithms%';
+------------------------------+-----------------------+
| Variable_name                | Value                 |
+------------------------------+-----------------------+
| slave_rows_search_algorithms | TABLE_SCAN,INDEX_SCAN |
+------------------------------+-----------------------+
1 row in set (0.00 sec)

mysql> SET GLOBAL slave_rows_search_algorithms = "TABLE_SCAN,INDEX_SCAN";
Query OK, 0 rows affected (0.00 sec)

mysql> SHOW VARIABLES LIKE '%algorithms%';
+------------------------------+-----------------------+
| Variable_name                | Value                 |
+------------------------------+-----------------------+
| slave_rows_search_algorithms | TABLE_SCAN,INDEX_SCAN |
+------------------------------+-----------------------+
1 row in set (0.00 sec)
```

The default value is TABLE_SCAN,INDEX_SCAN, which means that all searches that can use indexes do use them, and searches without any indexes use table scans.

Specifying INDEX_SCAN,TABLE_SCAN,HASH_SCAN has the same effect as specifying INDEX_SCAN,HASH_SCAN. To use hashing for any searches that does not use a primary or unique key, set this variable to INDEX_SCAN,HASH_SCAN. To force hashing for *all* searches, set it to TABLE_SCAN,HASH_SCAN.

- slave_skip_errors

Command-Line Format	--slave-skip-errors=name	
Option-File Format	slave-skip-errors	
System Variable Name	slave_skip_errors	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	string
	Default	OFF

	Valid Values	OFF
		[list of error codes]
		all
		ddl_exist_errors

Normally, replication stops when an error occurs on the slave. This gives you the opportunity to resolve the inconsistency in the data manually. This variable tells the slave SQL thread to continue replication when a statement returns any of the errors listed in the variable value.

- `slave_sql_verify_checksum`

System Variable Name	slave_sql_verify_checksum	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	boolean
	Default	1
	Valid Values	0
		1

Cause the slave SQL thread to verify data using the checksums read from the relay log. In the event of a mismatch, the slave stops with an error.

> **Note**
>
> The slave I/O thread always reads checksums if possible when accepting events from over the network.

- `slave_transaction_retries`

Command-Line Format	--slave_transaction_retries=#	
Option-File Format	slave_transaction_retries	
System Variable Name	slave_transaction_retries	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Platform Bit Size	32
	Type	numeric
	Default	10
	Range	0 .. 4294967295
	Permitted Values	
	Platform Bit Size	64
	Type	numeric

	Default	10
	Range	0 .. 18446744073709547520

If a replication slave SQL thread fails to execute a transaction because of an InnoDB deadlock or because the transaction's execution time exceeded InnoDB's innodb_lock_wait_timeout, it automatically retries slave_transaction_retries times before stopping with an error. The default value is 10.

Transactions cannot be retried when using a multi-threaded slave. In other words, whenever slave_parallel_workers is greater than 0, slave_transaction_retries is treated as equal to 0, and cannot be changed.

- slave_type_conversions

Command-Line Format	--slave_type_conversions=set	
Option-File Format	slave_type_conversions	
System Variable Name	slave_type_conversions	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values (<= 5.7.1)	
	Type	set
	Default	
	Valid Values	ALL_LOSSY
		ALL_NON_LOSSY
	Permitted Values (>= 5.7.2)	
	Type	set
	Default	
	Valid Values	ALL_LOSSY
		ALL_NON_LOSSY
		ALL_SIGNED
		ALL_UNSIGNED

Controls the type conversion mode in effect on the slave when using row-based replication. In MySQL 5.7.2 and later, its value is a comma-delimited set of zero or more elements from the list: ALL_LOSSY, ALL_NON_LOSSY, ALL_SIGNED, ALL_UNSIGNED. Set this variable to an empty string to disallow type conversions between the master and the slave. Changes require a restart of the slave to take effect.

ALL_SIGNED and ALL_UNSIGNED were added in MySQL 5.7.2 (Bug#15831300). For additional information on type conversion modes applicable to attribute promotion and demotion in row-based replication, see Row-based replication: attribute promotion and demotion.

- sql_slave_skip_counter

System Variable Name	sql_slave_skip_counter
Variable Scope	Global
Dynamic Variable	Yes
	Permitted Values

	Type	numeric

The number of events from the master that a slave server should skip.

This option is incompatible with GTID-based replication, and must not be set to a nonzero value when `--gtid-mode=ON`. In MySQL 5.7.1 and later, trying to do so is specifically disallowed. (Bug #15833516) If you need to skip transactions when employing GTIDs, use `gtid_executed` from the master instead. See Injecting empty transactions, for information about how to do this.

> **Important**
>
> If skipping the number of events specified by setting this variable would cause the slave to begin in the middle of an event group, the slave continues to skip until it finds the beginning of the next event group and begins from that point. For more information, see Section 13.4.2.5, "`SET GLOBAL sql_slave_skip_counter` Syntax".

- `sync_master_info`

Command-Line Format	`--sync-master-info=#`	
Option-File Format	`sync_master_info`	
System Variable Name	`sync_master_info`	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Platform Bit Size	32
	Type	numeric
	Default	10000
	Range	0 .. 4294967295
	Permitted Values	
	Platform Bit Size	64
	Type	numeric
	Default	10000
	Range	0 .. 18446744073709547520

The effects of this variable on a replication slave depend on whether the slave's `master_info_repository` is set to `FILE` or `TABLE`, as explained in the following paragraphs.

`master_info_repository = FILE`. If the value of `sync_master_info` is greater than 0, the slave synchronizes its `master.info` file to disk (using `fdatasync()`) after every `sync_master_info` events. If it is 0, the MySQL server performs no synchronization of the `master.info` file to disk; instead, the server relies on the operating system to flush its contents periodically as with any other file.

`master_info_repository = TABLE`. If the value of `sync_master_info` is greater than 0, the slave updates its master info repository table after every `sync_master_info` events. If it is 0, the table is never updated.

The default value for sync_master_info is 10000.

- sync_relay_log

Command-Line Format	--sync-relay-log=#	
Option-File Format	sync_relay_log	
System Variable Name	sync_relay_log	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Platform Bit Size	32
	Type	numeric
	Default	10000
	Range	0 .. 4294967295
	Permitted Values	
	Platform Bit Size	64
	Type	numeric
	Default	10000
	Range	0 .. 18446744073709547520

If the value of this variable is greater than 0, the MySQL server synchronizes its relay log to disk (using fdatasync()) after every sync_relay_log events are written to the relay log.

Setting sync_relay_log to 0 causes no synchronization to be done to disk; in this case, the server relies on the operating system to flush the relay log's contents from time to time as for any other file.

A value of 1 is the safest choice because in the event of a crash you lose at most one event from the relay log. However, it is also the slowest choice (unless the disk has a battery-backed cache, which makes synchronization very fast).

- sync_relay_log_info

Command-Line Format	--sync-relay-log-info=#	
Option-File Format	sync_relay_log_info	
System Variable Name	sync_relay_log_info	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Platform Bit Size	32
	Type	numeric
	Default	10000
	Range	0 .. 4294967295

	Permitted Values	
	Platform Bit Size	`64`
	Type	`numeric`
	Default	`10000`
	Range	`0 .. 18446744073709547520`

The effects of this variable on the slave depend on the server's `relay_log_info_repository` setting (`FILE` or `TABLE`), and if this is `TABLE`, additionally on whether the storage engine used by the relay log info table is transactional (such as `InnoDB`) or not (`MyISAM`). The effects of these factors on the behavior of the server for `sync_relay_log_info` values of zero and greater than zero are shown in the following table:

`sync_relay_log_info`	**`relay_log_info_repository`**		
	`FILE`	**`TABLE`**	
		Transactional	**Nontransactional**
`N > 0`	The slave synchronizes its `relay-log.info` file to disk (using `fdatasync()`) after every *`N`* transactions.	The table is updated after each transaction. (*`N`* is effectively ignored.)	The table is updated after every *`N`* events.
`0`	The MySQL server performs no synchronization of the `relay-log.info` file to disk; instead, the server relies on the operating system to flush its contents periodically as with any other file.		The table is never updated.

The default value for `sync_relay_log_info` is 10000.

Options for logging slave status to tables. MySQL logging of replication slave status information to tables rather than files. Writing of the master info log and the relay log info log can be configured separately using the two server options listed here:

- `--master-info-repository={FILE|TABLE}`

Command-Line Format	`--master-info-repository=FILE|TABLE`	
Option-File Format	`master-info-repository`	
	Permitted Values	
	Type	`string`
	Default	`FILE`
	Valid Values	`FILE`
		`TABLE`

This option causes the server to write its master info log to a file or a table. The name of the file defaults to `master.info`; you can change the name of the file using the `--master-info-file` server option.

The default value for this option is `FILE`. If you use `TABLE`, the log is written to the `slave_master_info` table in the `mysql` database.

- `--relay-log-info-repository={FILE|TABLE}`

Command-Line Format	--relay-log-info-repository=FILE|TABLE	
Option-File Format	relay-log-info-repository	
	Permitted Values	
	Type	string
	Default	FILE
	Valid Values	FILE
		TABLE

This option causes the server to log its relay log info to a file or a table. The name of the file defaults to `relay-log.info`; you can change the name of the file using the `--relay-log-info-file` server option.

The default value for this option is `FILE`. If you use `TABLE`, the log is written to the `slave_relay_log_info` table in the `mysql` database.

For replication to be crash-safe, this option must be set to `TABLE`; in additon, the `--relay-log-recovery` option must be enabled. See Crash-safe replication, for more information.

The info log tables and their contents are considered local to a given MySQL Server. They are not replicated, and changes to them are not written to the binary log.

For more information, see Section 16.2.2, “Replication Relay and Status Logs”.

16.1.4.4 Binary Log Options and Variables

You can use the `mysqld` options and system variables that are described in this section to affect the operation of the binary log as well as to control which statements are written to the binary log. For additional information about the binary log, see Section 5.2.4, “The Binary Log”. For additional information about using MySQL server options and system variables, see Section 5.1.3, “Server Command Options”, and Section 5.1.4, “Server System Variables”.

Startup options used with binary logging. The following list describes startup options for enabling and configuring the binary log. System variables used with binary logging are discussed later in this section.

- `--binlog-row-event-max-size=N`

Command-Line Format	--binlog-row-event-max-size=#	
Option-File Format	binlog-row-event-max-size	
	Permitted Values	
	Platform Bit Size	32
	Type	numeric
	Default	8192
	Range	256 .. 4294967295
	Permitted Values	
	Platform Bit Size	64

	Type	numeric
	Default	8192
	Range	256 .. 18446744073709547520

Specify the maximum size of a row-based binary log event, in bytes. Rows are grouped into events smaller than this size if possible. The value should be a multiple of 256. The default is 8192. See Section 16.1.2, “Replication Formats”.

- `--log-bin[=*base_name*]`

Command-Line Format	`--log-bin`	
Option-File Format	`log-bin`	
System Variable Name	`log_bin`	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	`file name`

Enable binary logging. The server logs all statements that change data to the binary log, which is used for backup and replication. See Section 5.2.4, “The Binary Log”.

The option value, if given, is the basename for the log sequence. The server creates binary log files in sequence by adding a numeric suffix to the basename. It is recommended that you specify a basename (see Section C.5.8, “Known Issues in MySQL”, for the reason). Otherwise, MySQL uses `*host_name*-bin` as the basename.

When the server reads an entry from the index file, it checks whether the entry contains a relative path, and if it does, the relative part of the path in replaced with the absolute path set using the `--log-bin` option. An absolute path remains unchanged; in such a case, the index must be edited manually to enable the new path or paths to be used. (In older versions of MySQL, manual intervention was required whenever relocating the binary log or relay log files.) (Bug #11745230, Bug #12133)

Setting this option causes the `log_bin` system variable to be set to `ON` (or `1`), and not to the basename. The binary log filename (with path) is available as the `log_bin_basename` system variable.

In MySQL 5.7.3 and later, if you specify this option without also specifying a `--server-id` [2162], the server is not allowed to start. (Bug #11763963, Bug #56739)

- `--log-bin-index[=*file_name*]`

Command-Line Format	`--log-bin-index=name`	
Option-File Format	`log-bin-index`	
	Permitted Values	
	Type	`file name`

The index file for binary log file names. See Section 5.2.4, “The Binary Log”. If you omit the file name, and if you did not specify one with `--log-bin`, MySQL uses `*host_name*-bin.index` as the file name.

- `--log-bin-trust-function-creators[={0|1}]`

Command-Line Format	`--log-bin-trust-function-creators`

Option-File Format	log-bin-trust-function-creators	
System Variable Name	log_bin_trust_function_creators	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	boolean
	Default	FALSE

This option sets the corresponding log_bin_trust_function_creators system variable. If no argument is given, the option sets the variable to 1. log_bin_trust_function_creators affects how MySQL enforces restrictions on stored function and trigger creation. See Section 18.7, “Binary Logging of Stored Programs”.

- --log-bin-use-v1-row-events[={0|1}]

Command-Line Format	--log-bin-use-v1-row-events[={0|1}]	
Option-File Format	log-bin-use-v1-row-events	
System Variable Name	log_bin_use_v1_row_events	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	boolean
	Default	0

MySQL 5.7 uses Version 2 binary log row events, which cannot be read by MySQL Server releases prior to MySQL 5.6.6. Setting this option to 1 causes mysqld to write the binary log using Version 1 logging events, which is the only version of binary log events used in previous releases, and thus produce binary logs that can be read by older slaves. Setting --log-bin-use-v1-row-events to 0 (the default) causes mysqld to use Version 2 binary log events.

The value used for this option can be obtained from the read-only log_bin_use_v1_row_events system variable.

- --log-short-format

Command-Line Format	--log-short-format	
Option-File Format	log-short-format	
	Permitted Values	
	Type	boolean
	Default	FALSE

Log less information to the binary log and slow query log, if they have been activated.

Statement selection options. The options in the following list affect which statements are written to the binary log, and thus sent by a replication master server to its slaves. There are also options for slave servers that control which statements received from the master should be executed or ignored. For details, see Section 16.1.4.3, “Replication Slave Options and Variables”.

- `--binlog-do-db=db_name`

Command-Line Format	`--binlog-do-db=name`	
Option-File Format	`binlog-do-db`	
	Permitted Values	
	Type	`string`

This option affects binary logging in a manner similar to the way that `--replicate-do-db` affects replication.

The effects of this option depend on whether the statement-based or row-based logging format is in use, in the same way that the effects of `--replicate-do-db` depend on whether statement-based or row-based replication is in use. You should keep in mind that the format used to log a given statement may not necessarily be the same as that indicated by the value of `binlog_format`. For example, DDL statements such as `CREATE TABLE` and `ALTER TABLE` are always logged as statements, without regard to the logging format in effect, so the following statement-based rules for `--binlog-do-db` always apply in determining whether or not the statement is logged.

Statement-based logging. Only those statements are written to the binary log where the default database (that is, the one selected by `USE`) is *`db_name`*. To specify more than one database, use this option multiple times, once for each database; however, doing so does *not* cause cross-database statements such as `UPDATE` *`some_db.some_table`* `SET foo='bar'` to be logged while a different database (or no database) is selected.

> **Warning**
>
> To specify multiple databases you *must* use multiple instances of this option. Because database names can contain commas, the list will be treated as the name of a single database if you supply a comma-separated list.

An example of what does not work as you might expect when using statement-based logging: If the server is started with `--binlog-do-db=sales` and you issue the following statements, the `UPDATE` statement is *not* logged:

```
USE prices;
UPDATE sales.january SET amount=amount+1000;
```

The main reason for this “just check the default database” behavior is that it is difficult from the statement alone to know whether it should be replicated (for example, if you are using multiple-table `DELETE` statements or multiple-table `UPDATE` statements that act across multiple databases). It is also faster to check only the default database rather than all databases if there is no need.

Another case which may not be self-evident occurs when a given database is replicated even though it was not specified when setting the option. If the server is started with `--binlog-do-db=sales`, the following `UPDATE` statement is logged even though `prices` was not included when setting `--binlog-do-db`:

```
USE sales;
UPDATE prices.discounts SET percentage = percentage + 10;
```

Because `sales` is the default database when the `UPDATE` statement is issued, the `UPDATE` is logged.

Row-based logging. Logging is restricted to database *db_name*. Only changes to tables belonging to *db_name* are logged; the default database has no effect on this. Suppose that the server is started with --binlog-do-db=sales and row-based logging is in effect, and then the following statements are executed:

```
USE prices;
UPDATE sales.february SET amount=amount+100;
```

The changes to the february table in the sales database are logged in accordance with the UPDATE statement; this occurs whether or not the USE statement was issued. However, when using the row-based logging format and --binlog-do-db=sales, changes made by the following UPDATE are not logged:

```
USE prices;
UPDATE prices.march SET amount=amount-25;
```

Even if the USE prices statement were changed to USE sales, the UPDATE statement's effects would still not be written to the binary log.

Another important difference in --binlog-do-db handling for statement-based logging as opposed to the row-based logging occurs with regard to statements that refer to multiple databases. Suppose that the server is started with --binlog-do-db=db1, and the following statements are executed:

```
USE db1;
UPDATE db1.table1 SET col1 = 10, db2.table2 SET col2 = 20;
```

If you are using statement-based logging, the updates to both tables are written to the binary log. However, when using the row-based format, only the changes to table1 are logged; table2 is in a different database, so it is not changed by the UPDATE. Now suppose that, instead of the USE db1 statement, a USE db4 statement had been used:

```
USE db4;
UPDATE db1.table1 SET col1 = 10, db2.table2 SET col2 = 20;
```

In this case, the UPDATE statement is not written to the binary log when using statement-based logging. However, when using row-based logging, the change to table1 is logged, but not that to table2—in other words, only changes to tables in the database named by --binlog-do-db are logged, and the choice of default database has no effect on this behavior.

- --binlog-ignore-db=*db_name*

Command-Line Format	--binlog-ignore-db=name	
Option-File Format	binlog-ignore-db	
	Permitted Values	
	Type	string

This option affects binary logging in a manner similar to the way that --replicate-ignore-db affects replication.

The effects of this option depend on whether the statement-based or row-based logging format is in use, in the same way that the effects of --replicate-ignore-db depend on whether statement-based or row-based replication is in use. You should keep in mind that the format used to log a given statement may not necessarily be the same as that indicated by the value of binlog_format. For example, DDL statements such as CREATE TABLE and ALTER TABLE are always logged as statements, without regard to the logging format in effect, so the following statement-based rules for --binlog-ignore-db always apply in determining whether or not the statement is logged.

Statement-based logging. Tells the server to not log any statement where the default database (that is, the one selected by USE) is *db_name*.

Prior to MySQL 5.7.2, this option caused any statements containing fully qualified table names not to be logged if there was no default database specified (that is, when SELECT DATABASE() returned NULL). In MySQL 5.7.2 and later, when there is no default database, no --binlog-ignore-db options are applied, and such statements are always logged. (Bug #11829838, Bug #60188)

Row-based format. Tells the server not to log updates to any tables in the database *db_name*. The current database has no effect.

When using statement-based logging, the following example does not work as you might expect. Suppose that the server is started with --binlog-ignore-db=sales and you issue the following statements:

```
USE prices;
UPDATE sales.january SET amount=amount+1000;
```

The UPDATE statement *is* logged in such a case because --binlog-ignore-db applies only to the default database (determined by the USE statement). Because the sales database was specified explicitly in the statement, the statement has not been filtered. However, when using row-based logging, the UPDATE statement's effects are *not* written to the binary log, which means that no changes to the sales.january table are logged; in this instance, --binlog-ignore-db=sales causes *all* changes made to tables in the master's copy of the sales database to be ignored for purposes of binary logging.

To specify more than one database to ignore, use this option multiple times, once for each database. Because database names can contain commas, the list will be treated as the name of a single database if you supply a comma-separated list.

You should not use this option if you are using cross-database updates and you do not want these updates to be logged.

Checksum options. MySQL 5.7 supports reading and writing of binary log checksums. These are enabled using the two options listed here:

- --binlog-checksum={NONE|CRC32}

Command-Line Format	--binlog-checksum=type	
Option-File Format	binlog-checksum	
	Permitted Values	
	Type	string
	Default	CRC32
	Valid Values	NONE
		CRC32

Enabling this option causes the master to write checksums for events written to the binary log. Set to NONE to disable, or the name of the algorithm to be used for generating checksums; currently, only CRC32 checksums are supported, and CRC32 is the default.

- `--master-verify-checksum={0|1}`

Command-Line Format	`--master-verify-checksum=name`	
Option-File Format	`master-verify-checksum`	
	Permitted Values	
	Type	`boolean`
	Default	`OFF`

  Enabling this option causes the master to verify events from the binary log using checksums, and to stop with an error in the event of a mismatch. Disabled by default.

To control reading of checksums by the slave (from the relay) log, use the `--slave-sql-verify-checksum` option.

Testing and debugging options. The following binary log options are used in replication testing and debugging. They are not intended for use in normal operations.

- `--max-binlog-dump-events=N`

Command-Line Format	`--max-binlog-dump-events=#`	
Option-File Format	`max-binlog-dump-events`	
	Permitted Values	
	Type	`numeric`
	Default	`0`

  This option is used internally by the MySQL test suite for replication testing and debugging.

- `--sporadic-binlog-dump-fail`

Command-Line Format	`--sporadic-binlog-dump-fail`	
Option-File Format	`sporadic-binlog-dump-fail`	
	Permitted Values	
	Type	`boolean`
	Default	`FALSE`

  This option is used internally by the MySQL test suite for replication testing and debugging.

- `--binlog-rows-query-log-events`

Command-Line Format	`--binlog-rows-query-log-events`	
Option-File Format	`binlog-rows-query-log-events`	
	Permitted Values	
	Type	`boolean`
	Default	`FALSE`

  This option enables `binlog_rows_query_log_events`.

System variables used with the binary log. The following list describes system variables for controlling binary logging. They can be set at server startup and some of them can be changed at runtime using SET. Server options used to control binary logging are listed earlier in this section. For information about the sql_log_bin and sql_log_off variables, see Section 5.1.4, “Server System Variables”.

- binlog_cache_size

Command-Line Format	--binlog_cache_size=#	
Option-File Format	binlog_cache_size	
System Variable Name	binlog_cache_size	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Platform Bit Size	32
	Type	numeric
	Default	32768
	Range	4096 .. 4294967295
	Permitted Values	
	Platform Bit Size	64
	Type	numeric
	Default	32768
	Range	4096 .. 18446744073709547520

The size of the cache to hold changes to the binary log during a transaction. A binary log cache is allocated for each client if the server supports any transactional storage engines and if the server has the binary log enabled (--log-bin option). If you often use large transactions, you can increase this cache size to get better performance. The Binlog_cache_use and Binlog_cache_disk_use status variables can be useful for tuning the size of this variable. See Section 5.2.4, “The Binary Log”.

binlog_cache_size sets the size for the transaction cache only; the size of the statement cache is governed by the binlog_stmt_cache_size system variable.

- binlog_checksum

System Variable Name	binlog_checksum	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	string
	Default	CRC32
	Valid Values	NONE
		CRC32

When enabled, this variable causes the master to write a checksum for each event in the binary log. binlog_checksum supports the values NONE (disabled) and CRC32. The default is CRC32.

When `binlog_checksum` is disabled (value `NONE`), the server verifies that it is writing only complete events to the binary log by writing and checking the event length (rather than a checksum) for each event.

Changing the value of this variable causes the binary log to be rotated; checksums are always written to an entire binary log file, and never to only part of one.

Setting this variable on the master to a value unrecognized by the slave causes the slave to set its own `binlog_checksum` value to `NONE`, and to stop replication with an error. (Bug #13553750, Bug #61096) If backward compatibility with older slaves is a concern, you may want to set the value explicitly to `NONE`.

- `binlog_direct_non_transactional_updates`

Command-Line Format	`--binlog_direct_non_transactional_updates[=value]`	
Option-File Format	`binlog_direct_non_transactional_updates`	
System Variable Name	`binlog_direct_non_transactional_updates`	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	`boolean`
	Default	`OFF`

Due to concurrency issues, a slave can become inconsistent when a transaction contains updates to both transactional and nontransactional tables. MySQL tries to preserve causality among these statements by writing nontransactional statements to the transaction cache, which is flushed upon commit. However, problems arise when modifications done to nontransactional tables on behalf of a transaction become immediately visible to other connections because these changes may not be written immediately into the binary log.

The `binlog_direct_non_transactional_updates` variable offers one possible workaround to this issue. By default, this variable is disabled. Enabling `binlog_direct_non_transactional_updates` causes updates to nontransactional tables to be written directly to the binary log, rather than to the transaction cache.

`binlog_direct_non_transactional_updates` works only for statements that are replicated using the statement-based binary logging format; that is, it works only when the value of `binlog_format` is `STATEMENT`, or when `binlog_format` is `MIXED` and a given statement is being replicated using the statement-based format. This variable has no effect when the binary log format is `ROW`, or when `binlog_format` is set to `MIXED` and a given statement is replicated using the row-based format.

> **Important**
>
> Before enabling this variable, you must make certain that there are no dependencies between transactional and nontransactional tables; an example of such a dependency would be the statement `INSERT INTO myisam_table SELECT * FROM innodb_table`. Otherwise, such statements are likely to cause the slave to diverge from the master.

In MySQL 5.7, this variable has no effect when the binary log format is `ROW` or `MIXED`. (Bug #51291)

- `binlog_format`

Command-Line Format	`--binlog-format=format`	
Option-File Format	`binlog-format`	
System Variable Name	`binlog_format`	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	`enumeration`
	Default	`STATEMENT`
	Valid Values	`ROW`
		`STATEMENT`
		`MIXED`

This variable sets the binary logging format, and can be any one of `STATEMENT`, `ROW`, or `MIXED`. See Section 16.1.2, “Replication Formats”. `binlog_format` is set by the `--binlog-format` option at startup, or by the `binlog_format` variable at runtime.

> **Note**
>
> While you can change the logging format at runtime, it is *not* recommended that you change it while replication is ongoing. This is due in part to the fact that slaves do not honor the master's `binlog_format` setting; a given MySQL Server can change only its own logging format.

In MySQL 5.7, the default format is `STATEMENT`.

You must have the `SUPER` privilege to set either the global or session `binlog_format` value.

The rules governing when changes to this variable take effect and how long the effect lasts are the same as for other MySQL server system variables. See Section 13.7.4, “`SET` Syntax”, for more information.

When `MIXED` is specified, statement-based replication is used, except for cases where only row-based replication is guaranteed to lead to proper results. For example, this happens when statements contain user-defined functions (UDF) or the `UUID()` function. An exception to this rule is that `MIXED` always uses statement-based replication for stored functions and triggers.

There are exceptions when you cannot switch the replication format at runtime:

- From within a stored function or a trigger.
- If the session is currently in row-based replication mode and has open temporary tables.
- From within a transaction.

Trying to switch the format in those cases results in an error.

The binary log format affects the behavior of the following server options:

- `--replicate-do-db`
- `--replicate-ignore-db`

- --binlog-do-db
- --binlog-ignore-db

These effects are discussed in detail in the descriptions of the individual options.

- binlog_max_flush_queue_time

System Variable Name	binlog_max_flush_queue_time	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	numeric
	Default	0
	Range	0 .. 100000

How long in microseconds to keep reading transactions from the flush queue before proceeding with the group commit (and syncing the log to disk, if sync_binlog is greater than 0). If the value is 0 (the default), there is no timeout and the server keeps reading new transactions until the queue is empty.

Normally, binlog_max_flush_queue_time can remain set to 0. If the server processes a large number of connections (for example, 100 or more) and many short transactions with low-latency requirements, it may be useful to set the value larger than 0 to force more frequent flushes to disk.

- binlog_order_commits

System Variable Name	binlog_order_commits	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	boolean
	Default	ON

If this variable is enabled (the default), transactions are committed in the same order they are written to the binary log. If disabled, transactions may be committed in parallel. In some cases, disabling this variable might produce a performance increment.

- binlog_row_image

Command-Line Format	--binlog-row-image=image_type	
Option-File Format	binlog_row_image	
System Variable Name	binlog_row_image=image_type	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	enumeration
	Default	full

	Valid Values	full (Log all columns)
		minimal (Log only changed columns, and columns needed to identify rows)
		noblob (Log all columns, except for unneeded BLOB and TEXT columns)

In MySQL row-based replication, each row change event contains two images, a “before” image whose columns are matched against when searching for the row to be updated, and an “after” image containing the changes. Normally, MySQL logs full rows (that is, all columns) for both the before and after images. However, it is not strictly necessary to include every column in both images, and we can often save disk, memory, and network usage by logging only those columns which are actually required.

Note

When deleting a row, only the before image is logged, since there are no changed values to propagate following the deletion. When inserting a row, only the after image is logged, since there is no existing row to be matched. Only when updating a row are both the before and after images required, and both written to the binary log.

For the before image, it is necessary only that the minimum set of columns required to uniquely identify rows is logged. If the table containing the row has a primary key, then only the primary key column or columns are written to the binary log. Otherwise, if the table has a unique key all of whose columns are `NOT NULL`, then only the columns in the unique key need be logged. (If the table has neither a primary key nor a unique key without any `NULL` columns, then all columns must be used in the before image, and logged.) In the after image, it is necessary to log only the columns which have actually changed.

You can cause the server to log full or minimal rows using the `binlog_row_image` system variable. This variable actually takes one of three possible values, as shown in the following list:

- `full`: Log all columns in both the before image and the after image.
- `minimal`: Log only those columns in the before image that are required to identify the row to be changed; log only those columns in the after image that are actually changed.
- `noblob`: Log all columns (same as `full`), except for `BLOB` and `TEXT` columns that are not required to identify rows, or that have not changed.

The default value is `full`.

In MySQL 5.5 and earlier, full row images are always used for both before images and after images. If you need to replicate from a newer master to a slave running MySQL 5.5 or earlier, the master should always use this value.

When using `minimal` or `noblob`, deletes and updates are guaranteed to work correctly for a given table if and only if the following conditions are true for both the source and destination tables:

- All columns must be present and in the same order; each column must use the same data type as its counterpart in the other table.
- The tables must have identical primary key definitions.

(In other words, the tables must be identical with the possible exception of indexes that are not part of the tables' primary keys.)

If these conditions are not met, it is possible that the primary key column values in the destination table may prove insufficient to provide a unique match for a delete or update. In this event, no warning or error is issued; the master and slave silently diverge, thus breaking consistency.

Setting this variable has no effect when the binary logging format is `STATEMENT`. When `binlog_format` is `MIXED`, the setting for `binlog_row_image` is applied to changes that are logged using row-based format, but this setting no effect on changes logged as statements.

Setting `binlog_row_image` on either the global or session level does not cause an implicit commit; this means that this variable can be changed while a transaction is in progress without affecting the transaction.

- `binlog_rows_query_log_events`

System Variable Name	`binlog_rows_query_log_events`	
Variable Scope	Global, Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	`boolean`
	Default	`FALSE`

The `binlog_rows_query_log_events` system variable affects row-based logging only. When enabled, it causes the MySQL Server to write informational log events such as row query log events into its binary log. This information can be used for debugging and related purposes; such as obtaining the original query issued on the master when it cannot be reconstructed from the row updates.

These events are normally ignored by MySQL programs reading the binary log and so cause no issues when replicating or restoring from backup.

- `binlog_stmt_cache_size`

Command-Line Format	`--binlog_stmt_cache_size=#`	
Option-File Format	`binlog_stmt_cache_size`	
System Variable Name	`binlog_stmt_cache_size`	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Platform Bit Size	`32`
	Type	`numeric`
	Default	`32768`
	Range	`4096 .. 4294967295`
	Permitted Values	
	Platform Bit Size	`64`
	Type	`numeric`
	Default	`32768`

	Range	4096 .. 18446744073709547520

This variable determines the size of the cache for the binary log to hold nontransactional statements issued during a transaction. Separate binary log transaction and statement caches are allocated for each client if the server supports any transactional storage engines and if the server has the binary log enabled (--log-bin option). If you often use large nontransactional statements during transactions, you can increase this cache size to get better performance. The Binlog_stmt_cache_use and Binlog_stmt_cache_disk_use status variables can be useful for tuning the size of this variable. See Section 5.2.4, “The Binary Log”.

The binlog_cache_size system variable sets the size for the transaction cache.

- log_bin

System Variable Name	log_bin
Variable Scope	Global
Dynamic Variable	No

Whether the binary log is enabled. If the --log-bin option is used, then the value of this variable is ON; otherwise it is OFF. This variable reports only on the status of binary logging (enabled or disabled); it does not actually report the value to which --log-bin is set.

See Section 5.2.4, “The Binary Log”.

- log_bin_basename

System Variable Name	log_bin_basename	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	file name
	Default	datadir + '/' + hostname + '-bin'

Holds the name and complete path to the binary log file. Unlike the log_bin system variable, log_bin_basename reflects the name set with the --log-bin server option.

- log_bin_index

System Variable Name	log_bin_index	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	file name

The index file for binary log file names.

- log_bin_use_v1_row_events

Command-Line Format	--log-bin-use-v1-row-events[={0|1}]
Option-File Format	log_bin_use_v1_row_events

System Variable Name	log_bin_use_v1_row_events	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	boolean
	Default	0

Shows whether Version 2 binary logging is in use. A value of 1 shows that the server is writing the binary log using Version 1 logging events (the only version of binary log events used in previous releases), and thus producing a binary log that can be read by older slaves. 0 indicates that Version 2 binary log events are in use.

This variable is read-only. To switch between Version 1 and Version 2 binary event binary logging, it is necessary to restart mysqld with the --log-bin-use-v1-row-events option.

- log_slave_updates

Command-Line Format	--log-slave-updates	
Option-File Format	log_slave_updates	
System Variable Name	log_slave_updates	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	boolean
	Default	FALSE

Whether updates received by a slave server from a master server should be logged to the slave's own binary log. Binary logging must be enabled on the slave for this variable to have any effect. See Section 16.1.4, “Replication and Binary Logging Options and Variables”.

- master_verify_checksum

System Variable Name	master_verify_checksum	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	boolean
	Default	OFF

Enabling this variable causes the master to examine checksums when reading from the binary log. master_verify_checksum is disabled by default; in this case, the master uses the event length from the binary log to verify events, so that only complete events are read from the binary log.

- max_binlog_cache_size

Command-Line Format	--max_binlog_cache_size=#
Option-File Format	max_binlog_cache_size

System Variable Name	max_binlog_cache_size	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	numeric
	Default	18446744073709547520
	Range	4096 .. 18446744073709547520

If a transaction requires more than this many bytes of memory, the server generates a `Multi-statement transaction required more than 'max_binlog_cache_size' bytes of storage` error. The minimum value is 4096. The maximum possible value is 16EB (exabytes). The maximum recommended value is 4GB; this is due to the fact that MySQL currently cannot work with binary log positions greater than 4GB.

`max_binlog_cache_size` sets the size for the transaction cache only; the upper limit for the statement cache is governed by the `max_binlog_stmt_cache_size` system variable.

In MySQL 5.7, the visibility to sessions of `max_binlog_cache_size` matches that of the `binlog_cache_size` system variable; in other words, changing its value effects only new sessions that are started after the value is changed.

- `max_binlog_size`

Command-Line Format	--max_binlog_size=#	
Option-File Format	max_binlog_size	
System Variable Name	max_binlog_size	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	numeric
	Default	1073741824
	Range	4096 .. 1073741824

If a write to the binary log causes the current log file size to exceed the value of this variable, the server rotates the binary logs (closes the current file and opens the next one). The minimum value is 4096 bytes. The maximum and default value is 1GB.

A transaction is written in one chunk to the binary log, so it is never split between several binary logs. Therefore, if you have big transactions, you might see binary log files larger than `max_binlog_size`.

If `max_relay_log_size` is 0, the value of `max_binlog_size` applies to relay logs as well.

- `max_binlog_stmt_cache_size`

Command-Line Format	--max_binlog_stmt_cache_size=#
Option-File Format	max_binlog_stmt_cache_size
System Variable Name	max_binlog_stmt_cache_size
Variable Scope	Global

Dynamic Variable	Yes	
	Permitted Values	
	Type	numeric
	Default	18446744073709547520
	Range	4096 .. 18446744073709547520

If nontransactional statements within a transaction require more than this many bytes of memory, the server generates an error. The minimum value is 4096. The maximum and default values are 4GB on 32-bit platforms and 16EB (exabytes) on 64-bit platforms.

max_binlog_stmt_cache_size sets the size for the statement cache only; the upper limit for the transaction cache is governed exclusively by the max_binlog_cache_size system variable.

- sync_binlog

Command-Line Format	--sync-binlog=#	
Option-File Format	sync_binlog	
System Variable Name	sync_binlog	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Platform Bit Size	32
	Type	numeric
	Default	0
	Range	0 .. 4294967295
	Permitted Values	
	Platform Bit Size	64
	Type	numeric
	Default	0
	Range	0 .. 18446744073709547520

If the value of this variable is greater than 0, the MySQL server synchronizes its binary log to disk (using fdatasync()) after sync_binlog commit groups are written to the binary log. The default value of sync_binlog is 0, which does no synchronizing to disk—in this case, the server relies on the operating system to flush the binary log's contents from time to time as for any other file. A value of 1 is the safest choice because in the event of a crash you lose at most one commit group from the binary log. However, it is also the slowest choice (unless the disk has a battery-backed cache, which makes synchronization very fast).

16.1.4.5 Global Transaction ID Options and Variables

The MySQL Server options and system variables described in this section are used to monitor and control Global Transaction Identifiers (GTIDs).

For additional information, see Section 16.1.3, “Replication with Global Transaction Identifiers”.

Startup options used in GTID replication. The followup server startup options are used with GTID-based replication:

- `--enforce-gtid-consistency`

Command-Line Format	`--enforce-gtid-consistency[=value]`	
Option-File Format	`enforce-gtid-consistency`	
System Variable Name	`enforce_gtid_consistency`	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	`boolean`
	Default	`false`

When set, this option allows execution of only those statements that can be logged in a transactionally safe manner. This means that the following operations *cannot* be used when this option is enabled:

- `CREATE TABLE ... SELECT` statements
- `CREATE TEMPORARY TABLE` statements inside transactions
- Transactions or statements that update both transactional and nontransactional tables.

Nontransactional DML statements are allowed on temporary tables with `--enforce-gtid-consistency` as long as all affected tables are temporary tables.

This option is intended chiefly for use with programs such as `mysql_install_db` and `mysql_upgrade`.

- `--gtid-mode`

Command-Line Format	`--gtid-mode=MODE`	
Option-File Format	`gtid-mode`	
System Variable Name	`gtid_mode`	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	`enumeration`
	Default	`OFF`
	Valid Values	`OFF`
		`UPGRADE_STEP_1`
		`UPGRADE_STEP_2`
		`ON`

This option specifies whether GTIDs are enabled. Starting the server with `--gtid-mode=ON` requires that the server also be started with the `--log-bin` and `--log-slave-updates` options as well. (In addition, you should also use `--enforce-gtid-consistency`.)

Setting this option to `OFF` when there are GTIDs in the binary log or in the relay log, or to `ON` when there remain anonymous transactions to be executed, causes an error.

> **Important**
>
> This option does not employ boolean values; its values are in fact enumerated. You should not attempt to use numeric values when setting this option, as these may lead to unexpected results. The values `UPGRADE_STEP_1` and `UPGRADE_STEP_2` are reserved for future use, but currently are not supported in production; if you use one of these two values with `--gtid-mode`, the server refuses to start.

It is possible but not recommended to run `mysql_upgrade` on a server where `--gtid-mode=ON`, since it may make changes to MySQL system tables that use the `MyISAM` storage engine, which is nontransactional.

Prior to MySQL 5.7.1, setting the global value for the `sql_slave_skip_counter` variable to 1 had no effect `--gtid-mode` was set to `ON`.. (Bug #15833516) A workaround in in previous versions is to reset the slave's position using `CHANGE MASTER TO ... MASTER_LOG_FILE = ... MASTER_LOG_POS = ...`, including the `MASTER_AUTO_POSITION = 0` option with this statement if needed.

System variables used on replication masters. The following system variables are used with GTID-based replication:

- `enforce_gtid_consistency`

Command-Line Format	`--enforce-gtid-consistency[=value]`	
Option-File Format	`enforce_gtid_consistency`	
System Variable Name	`enforce_gtid_consistency`	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	`boolean`
	Default	`false`

When this variable is true, execution is allowed of only those statements that can be logged in a transactionally safe manner, which means that the following operations cannot be used:

- `CREATE TABLE ... SELECT` statements
- `CREATE TEMPORARY TABLE` statements inside transactions
- Transactions or statements that update both transactional and nontransactional tables.

This variable is read-only. To set it, use the `--enforce-gtid-consistency` option on the command line or in an option file when starting the MySQL Server.

- `gtid_executed`

System Variable Name	`gtid_executed`
Variable Scope	Global, Session

Dynamic Variable	No	
	Permitted Values	
	Type	`string`

When used with global scope, this variable contains a representation of the set of all transactions that are logged in the binary log. This is the same as the value of the `Executed_Gtid_Set` column in the output of `SHOW MASTER STATUS` and `SHOW SLAVE STATUS`.

When used with session scope, this variable contains a representation of the set of transactions that are written to the cache in the current session.

The set of transactions that can be found in the binary logs at any given time is equal to `GTID_SUBTRACT(@@global.gtid_executed, @@global.gtid_purged)`; that is, to all transactions in the binary log that have not yet been purged.

When the server starts, `@@global.gtid_executed` is initialized to the union of the following two sets:

- The GTIDs listed in the `Previous_gtids_log_event` of the newest binary log
- The GTIDs found in every `Gtid_log_event` in the newest binary log.

Thereafter, GTIDs are added to the set as transactions are executed.

Issuing `RESET MASTER` causes the global value (but not the session value) of this variable to be reset to an empty string. GTIDs are not otherwise removed from this set other than when the set is cleared due to `RESET MASTER`. The set is also cleared if the server is shut down and all binary logs are removed.

- `gtid_mode`

System Variable Name	`gtid_mode`	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	`enumeration`
	Default	`OFF`
	Valid Values	`OFF`
		`UPGRADE_STEP_1`
		`UPGRADE_STEP_2`
		`ON`

Shows whether GTIDs are enabled. Read-only; set using `--gtid-mode`.

- `gtid_next`

System Variable Name	`gtid_next`	
Variable Scope	Session	
Dynamic Variable	Yes	
	Permitted Values	
	Type	`enumeration`

	Default	AUTOMATIC
	Valid Values	AUTOMATIC
		ANONYMOUS
		UUID:NUMBER

This variable is used to specify whether and how the next GTID is obtained. gtid_next can take any of the following values:

- AUTOMATIC: Use the next automatically-generated global transaction ID.
- ANONYMOUS: Transactions do not have global identifiers, and are identified by file and position only.
- A global transaction ID in *UUID*:*NUMBER* format.

You must have the SUPER privilege to set this variable. Setting this variable has no effect if gtid_mode is OFF.

In MySQL 5.7.1, you cannot execute any of the statements CHANGE MASTER TO, START SLAVE, STOP SLAVE, REPAIR TABLE, OPTIMIZE TABLE, ANALYZE TABLE, CHECK TABLE, CREATE SERVER, ALTER SERVER, DROP SERVER, CACHE INDEX, LOAD INDEX INTO CACHE, FLUSH, or RESET when gtid_next is set to any value other than AUTOMATIC; in such cases, the statement fails with an error. Such statements are *not* disallowed in MySQL 5.7.2 and later. (Bug #16062608, Bug #16715809, Bug #69045) (Bug #16062608)

- gtid_owned

System Variable Name	gtid_owned	
Variable Scope	Global, Session	
Dynamic Variable	No	
	Permitted Values	
	Type	string

This read-only variable holds a list whose contents depend on its scope. When used with session scope, the list holds all GTIDs that are owned by this client; when used with global scope, it holds a list of all GTIDs along with their owners.

- gtid_purged

System Variable Name	gtid_purged	
Variable Scope	Global	
Dynamic Variable	Yes	
	Permitted Values	
	Type	string

The set of all transactions that have been purged from the binary log.

When the server starts, the global value of gtid_purged is initialized to the set of GTIDs contained by the Previous_gtid_log_event of the oldest binary log. When a binary log is purged, @@global.gtid_purged is re-read from the binary log that has now become the oldest one.

It is possible to update the value of this variable, but only by adding GTIDs to those already listed, and only when gtid_executed is unset—that is, on a new server.

Issuing RESET MASTER causes the value of this variable to be reset to an empty string.

16.1.5 Common Replication Administration Tasks

Once replication has been started it should execute without requiring much regular administration. Depending on your replication environment, you will want to check the replication status of each slave periodically, daily, or even more frequently.

16.1.5.1 Checking Replication Status

The most common task when managing a replication process is to ensure that replication is taking place and that there have been no errors between the slave and the master. The primary statement for this is SHOW SLAVE STATUS, which you must execute on each slave:

```
mysql> SHOW SLAVE STATUS\G
*************************** 1. row ***************************
               Slave_IO_State: Waiting for master to send event
                  Master_Host: master1
                  Master_User: root
                  Master_Port: 3306
                Connect_Retry: 60
              Master_Log_File: mysql-bin.000004
          Read_Master_Log_Pos: 931
               Relay_Log_File: slave1-relay-bin.000056
                Relay_Log_Pos: 950
        Relay_Master_Log_File: mysql-bin.000004
             Slave_IO_Running: Yes
            Slave_SQL_Running: Yes
              Replicate_Do_DB:
          Replicate_Ignore_DB:
           Replicate_Do_Table:
       Replicate_Ignore_Table:
      Replicate_Wild_Do_Table:
  Replicate_Wild_Ignore_Table:
                   Last_Errno: 0
                   Last_Error:
                 Skip_Counter: 0
          Exec_Master_Log_Pos: 931
              Relay_Log_Space: 1365
              Until_Condition: None
               Until_Log_File:
                Until_Log_Pos: 0
           Master_SSL_Allowed: No
           Master_SSL_CA_File:
           Master_SSL_CA_Path:
              Master_SSL_Cert:
            Master_SSL_Cipher:
               Master_SSL_Key:
        Seconds_Behind_Master: 0
Master_SSL_Verify_Server_Cert: No
                Last_IO_Errno: 0
                Last_IO_Error:
               Last_SQL_Errno: 0
               Last_SQL_Error:
  Replicate_Ignore_Server_Ids: 0
```

The key fields from the status report to examine are:

- Slave_IO_State: The current status of the slave. See Section 8.12.5.5, “Replication Slave I/O Thread States”, and Section 8.12.5.6, “Replication Slave SQL Thread States”, for more information.

- Slave_IO_Running: Whether the I/O thread for reading the master's binary log is running. Normally, you want this to be Yes unless you have not yet started replication or have explicitly stopped it with STOP SLAVE.

- Slave_SQL_Running: Whether the SQL thread for executing events in the relay log is running. As with the I/O thread, this should normally be Yes.

- Last_IO_Error, Last_SQL_Error: The last errors registered by the I/O and SQL threads when processing the relay log. Ideally these should be blank, indicating no errors.

- Seconds_Behind_Master: The number of seconds that the slave SQL thread is behind processing the master binary log. A high number (or an increasing one) can indicate that the slave is unable to handle events from the master in a timely fashion.

  A value of 0 for Seconds_Behind_Master can usually be interpreted as meaning that the slave has caught up with the master, but there are some cases where this is not strictly true. For example, this can occur if the network connection between master and slave is broken but the slave I/O thread has not yet noticed this—that is, slave_net_timeout has not yet elapsed.

  It is also possible that transient values for Seconds_Behind_Master may not reflect the situation accurately. When the slave SQL thread has caught up on I/O, Seconds_Behind_Master displays 0; but when the slave I/O thread is still queuing up a new event, Seconds_Behind_Master may show a large value until the SQL thread finishes executing the new event. This is especially likely when the events have old timestamps; in such cases, if you execute SHOW SLAVE STATUS several times in a relatively short period, you may see this value change back and forth repeatedly between 0 and a relatively large value.

Several pairs of fields provide information about the progress of the slave in reading events from the master binary log and processing them in the relay log:

- (Master_Log_file, Read_Master_Log_Pos): Coordinates in the master binary log indicating how far the slave I/O thread has read events from that log.

- (Relay_Master_Log_File, Exec_Master_Log_Pos): Coordinates in the master binary log indicating how far the slave SQL thread has executed events received from that log.

- (Relay_Log_File, Relay_Log_Pos): Coordinates in the slave relay log indicating how far the slave SQL thread has executed the relay log. These correspond to the preceding coordinates, but are expressed in slave relay log coordinates rather than master binary log coordinates.

On the master, you can check the status of connected slaves using SHOW PROCESSLIST to examine the list of running processes. Slave connections have Binlog Dump in the Command field:

```
mysql> SHOW PROCESSLIST \G;
*************************** 4. row ***************************
     Id: 10
   User: root
   Host: slave1:58371
     db: NULL
Command: Binlog Dump
   Time: 777
  State: Has sent all binlog to slave; waiting for binlog to be updated
   Info: NULL
```

Because it is the slave that drives the replication process, very little information is available in this report.

For slaves that were started with the --report-host option and are connected to the master, the SHOW SLAVE HOSTS statement on the master shows basic information about the slaves. The output includes the ID of the slave server, the value of the --report-host option, the connecting port, and master ID:

```
mysql> SHOW SLAVE HOSTS;
+-----------+--------+------+-------------------+-----------+
| Server_id | Host   | Port | Rpl_recovery_rank | Master_id |
+-----------+--------+------+-------------------+-----------+
|        10 | slave1 | 3306 |                 0 |         1 |
+-----------+--------+------+-------------------+-----------+
1 row in set (0.00 sec)
```

16.1.5.2 Pausing Replication on the Slave

You can stop and start the replication of statements on the slave using the STOP SLAVE and START SLAVE statements.

To stop processing of the binary log from the master, use STOP SLAVE:

```
mysql> STOP SLAVE;
```

When replication is stopped, the slave I/O thread stops reading events from the master binary log and writing them to the relay log, and the SQL thread stops reading events from the relay log and executing them. You can pause the I/O or SQL thread individually by specifying the thread type:

```
mysql> STOP SLAVE IO_THREAD;
mysql> STOP SLAVE SQL_THREAD;
```

To start execution again, use the START SLAVE statement:

```
mysql> START SLAVE;
```

To start a particular thread, specify the thread type:

```
mysql> START SLAVE IO_THREAD;
mysql> START SLAVE SQL_THREAD;
```

For a slave that performs updates only by processing events from the master, stopping only the SQL thread can be useful if you want to perform a backup or other task. The I/O thread will continue to read events from the master but they are not executed. This makes it easier for the slave to catch up when you restart the SQL thread.

Stopping only the I/O thread enables the events in the relay log to be executed by the SQL thread up to the point where the relay log ends. This can be useful when you want to pause execution to catch up with events already received from the master, when you want to perform administration on the slave but also ensure that it has processed all updates to a specific point. This method can also be used to pause event receipt on the slave while you conduct administration on the master. Stopping the I/O thread but permitting the SQL thread to run helps ensure that there is not a massive backlog of events to be executed when replication is started again.

16.2 Replication Implementation

Replication is based on the master server keeping track of all changes to its databases (updates, deletes, and so on) in its binary log. The binary log serves as a written record of all events that modify database structure or content (data) from the moment the server was started. Typically, SELECT statements are not recorded because they modify neither database structure nor content.

Each slave that connects to the master requests a copy of the binary log. That is, it pulls the data from the master, rather than the master pushing the data to the slave. The slave also executes the events from the binary log that it receives. This has the effect of repeating the original changes just as they were made on the master. Tables are created or their structure modified, and data is inserted, deleted, and updated according to the changes that were originally made on the master.

Because each slave is independent, the replaying of the changes from the master's binary log occurs independently on each slave that is connected to the master. In addition, because each slave receives a copy of the binary log only by requesting it from the master, the slave is able to read and update the copy of the database at its own pace and can start and stop the replication process at will without affecting the ability to update to the latest database status on either the master or slave side.

For more information on the specifics of the replication implementation, see Section 16.2.1, "Replication Implementation Details".

Masters and slaves report their status in respect of the replication process regularly so that you can monitor them. See Section 8.12.5, "Examining Thread Information", for descriptions of all replicated-related states.

The master binary log is written to a local relay log on the slave before it is processed. The slave also records information about the current position with the master's binary log and the local relay log. See Section 16.2.2, "Replication Relay and Status Logs".

Database changes are filtered on the slave according to a set of rules that are applied according to the various configuration options and variables that control event evaluation. For details on how these rules are applied, see Section 16.2.3, "How Servers Evaluate Replication Filtering Rules".

16.2.1 Replication Implementation Details

MySQL replication capabilities are implemented using three threads, one on the master server and two on the slave:

- **Binlog dump thread.** The master creates a thread to send the binary log contents to a slave when the slave connects. This thread can be identified in the output of `SHOW PROCESSLIST` on the master as the `Binlog Dump` thread.

  The binlog dump thread acquires a lock on the master's binary log for reading each event that is to be sent to the slave. As soon as the event has been read, the lock is released, even before the event is sent to the slave.

- **Slave I/O thread.** When a `START SLAVE` statement is issued on a slave server, the slave creates an I/O thread, which connects to the master and asks it to send the updates recorded in its binary logs.

  The slave I/O thread reads the updates that the master's `Binlog Dump` thread sends (see previous item) and copies them to local files that comprise the slave's relay log.

  The state of this thread is shown as `Slave_IO_running` in the output of `SHOW SLAVE STATUS` or as `Slave_running` in the output of `SHOW STATUS`.

- **Slave SQL thread.** The slave creates an SQL thread to read the relay log that is written by the slave I/O thread and execute the events contained therein.

In the preceding description, there are three threads per master/slave connection. A master that has multiple slaves creates one binlog dump thread for each currently connected slave, and each slave has its own I/O and SQL threads.

A slave uses two threads to separate reading updates from the master and executing them into independent tasks. Thus, the task of reading statements is not slowed down if statement execution is slow. For example, if the slave server has not been running for a while, its I/O thread can quickly fetch all the binary log contents from the master when the slave starts, even if the SQL thread lags far behind. If the slave stops before the SQL thread has executed all the fetched statements, the I/O thread has at least fetched everything so that a safe copy of the statements is stored locally in the slave's relay logs, ready for execution the next time that the slave starts. This enables the master server to purge its binary logs sooner because it no longer needs to wait for the slave to fetch their contents.

The `SHOW PROCESSLIST` statement provides information that tells you what is happening on the master and on the slave regarding replication. For information on master states, see Section 8.12.5.4, “Replication Master Thread States”. For slave states, see Section 8.12.5.5, “Replication Slave I/O Thread States”, and Section 8.12.5.6, “Replication Slave SQL Thread States”.

The following example illustrates how the three threads show up in the output from `SHOW PROCESSLIST`.

On the master server, the output from `SHOW PROCESSLIST` looks like this:

```
mysql> SHOW PROCESSLIST\G
*************************** 1. row ***************************
     Id: 2
   User: root
   Host: localhost:32931
     db: NULL
Command: Binlog Dump
   Time: 94
  State: Has sent all binlog to slave; waiting for binlog to
         be updated
   Info: NULL
```

Here, thread 2 is a `Binlog Dump` replication thread that services a connected slave. The `State` information indicates that all outstanding updates have been sent to the slave and that the master is waiting for more updates to occur. If you see no `Binlog Dump` threads on a master server, this means that replication is not running; that is, no slaves are currently connected.

On a slave server, the output from `SHOW PROCESSLIST` looks like this:

```
mysql> SHOW PROCESSLIST\G
*************************** 1. row ***************************
     Id: 10
   User: system user
   Host:
     db: NULL
Command: Connect
   Time: 11
  State: Waiting for master to send event
   Info: NULL
*************************** 2. row ***************************
     Id: 11
   User: system user
   Host:
     db: NULL
Command: Connect
   Time: 11
  State: Has read all relay log; waiting for the slave I/O
         thread to update it
   Info: NULL
```

The `State` information indicates that thread 10 is the I/O thread that is communicating with the master server, and thread 11 is the SQL thread that is processing the updates stored in the relay logs. At the time that `SHOW PROCESSLIST` was run, both threads were idle, waiting for further updates.

The value in the `Time` column can show how late the slave is compared to the master. See Section B.13, “MySQL 5.7 FAQ: Replication”. If sufficient time elapses on the master side without activity on the `Binlog Dump` thread, the master determines that the slave is no longer connected. As for any other client connection, the timeouts for this depend on the values of `net_write_timeout` and `net_retry_count`; for more information about these, see Section 5.1.4, “Server System Variables”.

The `SHOW SLAVE STATUS` statement provides additional information about replication processing on a slave server. See Section 16.1.5.1, “Checking Replication Status”.

16.2.2 Replication Relay and Status Logs

During replication, a slave server creates several logs that hold the binary log events relayed from the master to the slave, and to record information about the current status and location within the relay log. There are three types of logs used in the process, listed here:

- The *relay log* consists of the events read from the binary log of the master and written by the slave I/O thread. Events in the relay log are executed on the slave as part of the SQL thread.

- The *master info log* contains status and current configuration information for the slave's connection to the master. This log holds information on the master host name, login credentials, and coordinates indicating how far the slave has read from the master's binary log.

  This log can be written to the `mysql.slave_master_info` table instead of a file, by starting the slave with `--master-info-repository=TABLE`.

- The *relay log info log* holds status information about the execution point within the slave's relay log.

  This log can be written to the `mysql.slave_relay_log_info` table instead of a file by starting the slave with `--relay-log-info-repository=TABLE`.

Crash-safe replication. In order for replication to be crash-safe when using tables for logging status and relay information, these tables must use a transactional storage engine, such as `InnoDB`. Beginning with MySQL 5.6.6, these tables are created using `InnoDB`. (Bug #13538891)

Therefore, in order to guarantee crash safety on the slave, you must run the slave with `--relay-log-recovery` enabled, in addition to setting `--relay-log-info-repository` to `TABLE`.

In MySQL 5.7, a warning is given when `mysqld` is unable to initialize the replication logging tables, but the slave is allowed to continue starting. (Bug #13971348) This situation is most likely to occur when upgrading from a version of MySQL that does not support slave logging tables to one in which they are supported.

In MySQL 5.7, execution of any statement requiring a write lock on either or both of the `slave_master_info` and `slave_relay_log_info` tables is disallowed while replication is ongoing, while statements that perform only reads are permitted at any time.

> ⚠ **Important**
>
> Do not attempt to update or insert rows in the `slave_master_info` or `slave_relay_log_info` table manually. Doing so can cause undefined behavior, and is not supported.

16.2.2.1 The Slave Relay Log

The relay log, like the binary log, consists of a set of numbered files containing events that describe database changes, and an index file that contains the names of all used relay log files.

The term “relay log file” generally denotes an individual numbered file containing database events. The term “relay log” collectively denotes the set of numbered relay log files plus the index file.

Relay log files have the same format as binary log files and can be read using mysqlbinlog (see Section 4.6.7, “mysqlbinlog — Utility for Processing Binary Log Files”).

By default, relay log file names have the form *host_name*-relay-bin.*nnnnnn* in the data directory, where *host_name* is the name of the slave server host and *nnnnnn* is a sequence number. Successive relay log files are created using successive sequence numbers, beginning with 000001. The slave uses an index file to track the relay log files currently in use. The default relay log index file name is *host_name*-relay-bin.index in the data directory.

The default relay log file and relay log index file names can be overridden with, respectively, the --relay-log and --relay-log-index server options (see Section 16.1.4, “Replication and Binary Logging Options and Variables”).

If a slave uses the default host-based relay log file names, changing a slave's host name after replication has been set up can cause replication to fail with the errors Failed to open the relay log and Could not find target log during relay log initialization. This is a known issue (see Bug #2122). If you anticipate that a slave's host name might change in the future (for example, if networking is set up on the slave such that its host name can be modified using DHCP), you can avoid this issue entirely by using the --relay-log and --relay-log-index options to specify relay log file names explicitly when you initially set up the slave. This will make the names independent of server host name changes.

If you encounter the issue after replication has already begun, one way to work around it is to stop the slave server, prepend the contents of the old relay log index file to the new one, and then restart the slave. On a Unix system, this can be done as shown here:

```
shell> cat new_relay_log_name.index >> old_relay_log_name.index
shell> mv old_relay_log_name.index new_relay_log_name.index
```

A slave server creates a new relay log file under the following conditions:

- Each time the I/O thread starts.
- When the logs are flushed; for example, with FLUSH LOGS or mysqladmin flush-logs.
- When the size of the current relay log file becomes “too large,” determined as follows:
  - If the value of max_relay_log_size is greater than 0, that is the maximum relay log file size.
  - If the value of max_relay_log_size is 0, max_binlog_size determines the maximum relay log file size.

The SQL thread automatically deletes each relay log file as soon as it has executed all events in the file and no longer needs it. There is no explicit mechanism for deleting relay logs because the SQL thread takes care of doing so. However, FLUSH LOGS rotates relay logs, which influences when the SQL thread deletes them.

16.2.2.2 Slave Status Logs

A replication slave server creates two logs. By default, these logs are files named master.info and relay-log.info and created in the data directory. The names and locations of these files can be changed by using the --master-info-file and --relay-log-info-file options, respectively. In MySQL 5.7, either or both of these logs can also be written to tables in the mysql database by starting the server with the appropriate option: use --master-info-repository to have the master info log written to the mysql.slave_master_info table, and use --relay-log-info-repository to have the relay log info log written to the mysql.slave_relay_log_info table. See Section 16.1.4, “Replication and Binary Logging Options and Variables”.

The two status logs contain information like that shown in the output of the SHOW SLAVE STATUS statement, which is discussed in Section 13.4.2, “SQL Statements for Controlling Slave Servers”. Because the status logs are stored on disk, they survive a slave server's shutdown. The next time the slave starts up, it reads the two logs to determine how far it has proceeded in reading binary logs from the master and in processing its own relay logs.

The master info log file or table should be protected because it contains the password for connecting to the master. See Section 6.1.2.3, “Passwords and Logging”.

The slave I/O thread updates the master info log. The following table shows the correspondence between the lines in the master.info file, the columns in the mysql.slave_master_info table, and the columns displayed by SHOW SLAVE STATUS.

Line in master.info File	slave_master_info Table Column	SHOW SLAVE STATUS Column	Description
1	Number_of_lines	[None]	Number of lines in the file
2	Master_log_name	Master_Log_File	The name of the master binary log currently being read from the master
3	Master_log_pos	Read_Master_Log_Pos	The current position within the master binary log that have been read from the master
4	Host	Master_Host	The host name of the master
5	User	Master_User	The user name used to connect to the master
6	User_password	Password (not shown by SHOW SLAVE STATUS)	The password used to connect to the master
7	Port	Master_Port	The network port used to connect to the master
8	Connect_retry	Connect_Retry	The period (in seconds) that the slave will wait before trying to reconnect to the master
9	Enabled_ssl	Master_SSL_Allowed	Indicates whether the

Line in master.info File	slave_master_info Table Column	SHOW SLAVE STATUS Column	Description
			server supports SSL connections
10	Ssl_ca	Master_SSL_CA_File	The file used for the Certificate Authority (CA) certificate
11	Ssl_capath	Master_SSL_CA_Path	The path to the Certificate Authority (CA) certificates
12	Ssl_cert	Master_SSL_Cert	The name of the SSL certificate file
13	Ssl_cipher	Master_SSL_Cipher	The list of possible ciphers used in the handshake for the SSL connection
14	Ssl_key	Master_SSL_Key	The name of the SSL key file
15	Ssl_verify_server_cert	Master_SSL_Verify_Server_Cert	Whether to verify the server certificate
16	Heartbeat	[None]	Interval between replication heartbeats, in seconds
17	Bind	Master_Bind	Which of the slave's network interfaces should be used for connecting to the master
18	Ignored_server_ids	Replicate_Ignore_Server_Ids	The number of server IDs to be ignored, followed by the actual server IDs
19	Uuid	Master_UUID	The master's unique ID
20	Retry_count	Master_Retry_Count	Maximum number of reconnection attempts permitted

The slave SQL thread updates the relay log info log. In MySQL 5.7, the relay-log.info file includes a line count and a replication delay value. The following table shows the correspondence between the lines in the relay-log.info file, the columns in the mysql.slave_relay_log_info table, and the columns displayed by SHOW SLAVE STATUS.

Line in relay-log.info	slave_relay_log_info Table Column	SHOW SLAVE STATUS Column	Description
1	Number_of_lines	[None]	Number of lines in the file or rows in the table
2	Relay_log_name	Relay_Log_File	The name of the current relay log file
3	Relay_log_pos	Relay_Log_Pos	The current position within the relay log file; events up to this position have been executed on the slave database
4	Master_log_name	Relay_Master_Log_File	The name of the master binary log file from which the events in the relay log file were read
5	Master_log_pos	Exec_Master_Log_Pos	The equivalent position within the master's binary log file of events that have already been executed
5	Sql_delay	SQL_Delay	The number of seconds that the slave must lag the master

In older versions of MySQL (prior to MySQL 5.6), the relay-log.info file does not include a line count or a delay value (and the slave_relay_log_info table is not available).

Line	Status Column	Description
1	Relay_Log_File	The name of the current relay log file
2	Relay_Log_Pos	The current position within the relay log file; events up to this position have been executed on the slave database
3	Relay_Master_Log_File	The name of the master binary log file from which the events in the relay log file were read
4	Exec_Master_Log_Pos	The equivalent position within the master's binary log file of events that have already been executed

Note

If you downgrade a slave server to a version older than MySQL 5.6, the older server does not read the relay-log.info file correctly. To address this, modify the file in a text editor by deleting the initial line containing the number of lines.

The contents of the relay-log.info file and the states shown by the SHOW SLAVE STATUS statement might not match if the relay-log.info file has not been flushed to disk. Ideally, you should only view relay-log.info on a slave that is offline (that is, mysqld is not running). For a running system, you can use SHOW SLAVE STATUS, or query the slave_master_info and slave_relay_log_info tables if you are writing the status logs to tables.

When you back up the slave's data, you should back up these two status logs, along with the relay log files. The status logs are needed to resume replication after you restore the data from the slave. If you lose the relay logs but still have the relay log info log, you can check it to determine how far the SQL thread has executed in the master binary logs. Then you can use CHANGE MASTER TO with the MASTER_LOG_FILE and MASTER_LOG_POS options to tell the slave to re-read the binary logs from that point. Of course, this requires that the binary logs still exist on the master.

16.2.3 How Servers Evaluate Replication Filtering Rules

If a master server does not write a statement to its binary log, the statement is not replicated. If the server does log the statement, the statement is sent to all slaves and each slave determines whether to execute it or ignore it.

On the master, you can control which databases to log changes for by using the --binlog-do-db and --binlog-ignore-db options to control binary logging. For a description of the rules that servers use in evaluating these options, see Section 16.2.3.1, “Evaluation of Database-Level Replication and Binary Logging Options”. You should not use these options to control which databases and tables are replicated. Instead, use filtering on the slave to control the events that are executed on the slave.

On the slave side, decisions about whether to execute or ignore statements received from the master are made according to the --replicate-* options that the slave was started with. (See Section 16.1.4, “Replication and Binary Logging Options and Variables”.) In MySQL 5.7.3 and later, the filters governed by these options can also be set dynamically using the CHANGE REPLICATION FILTER statement. The rules governing such filters are the same whether they are created on startup using --replicate-* options or while the slave server is running by CHANGE REPLICATION FILTER.

In the simplest case, when there are no --replicate-* options, the slave executes all statements that it receives from the master. Otherwise, the result depends on the particular options given.

Database-level options (--replicate-do-db, --replicate-ignore-db) are checked first; see Section 16.2.3.1, “Evaluation of Database-Level Replication and Binary Logging Options”, for a description of this process. If no database-level options are used, option checking proceeds to any table-level options that may be in use (see Section 16.2.3.2, “Evaluation of Table-Level Replication Options”, for a discussion of these). If one or more database-level options are used but none are matched, the statement is not replicated.

For statements affecting databases only (that is, CREATE DATABASE, DROP DATABASE, and ALTER DATABASE), database-level options always take precedence over any --replicate-wild-do-table options. In other words, for such statements, --replicate-wild-do-table options are checked if and only if there are no database-level options that apply. This is a change in behavior from previous versions of MySQL, where the statement CREATE DATABASE dbx was not replicated if the slave had been started with --replicate-do-db=dbx --replicate-wild-do-table=db%.t1. (Bug #46110)

To make it easier to determine what effect an option set will have, it is recommended that you avoid mixing “do” and “ignore” options, or wildcard and nonwildcard options.

If any `--replicate-rewrite-db` options were specified, they are applied before the `--replicate-*` filtering rules are tested.

Note

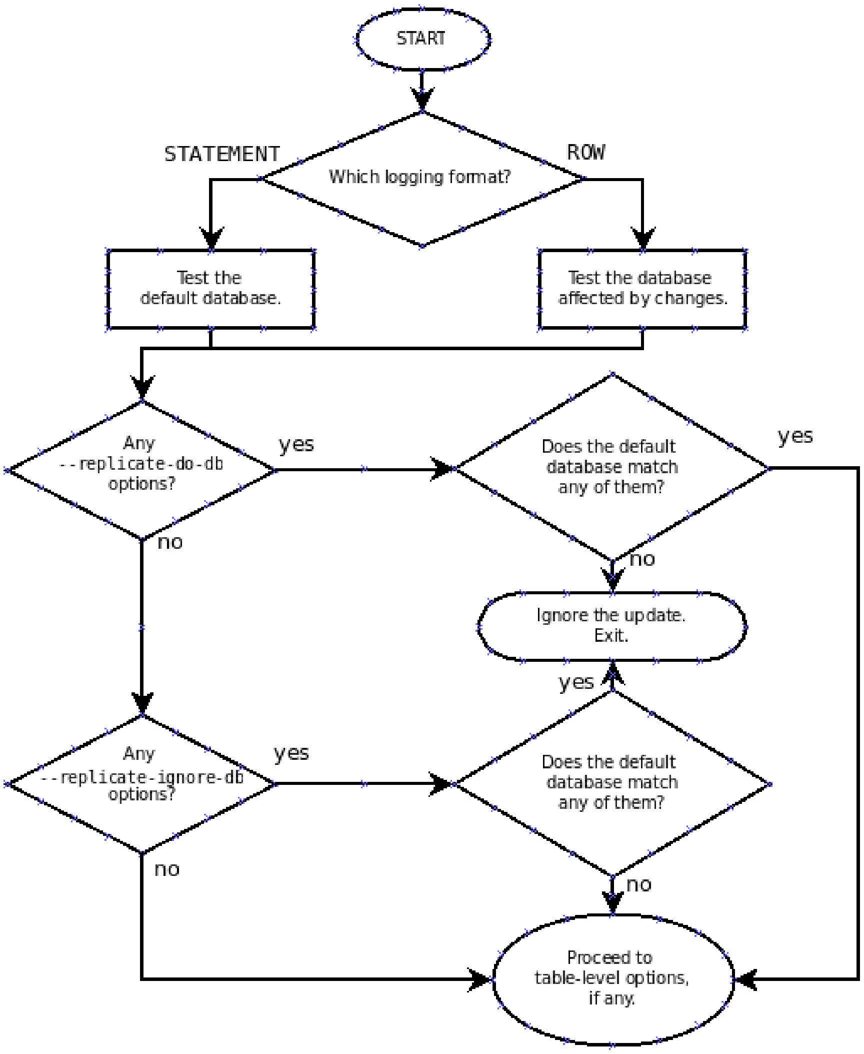
In MySQL 5.7, all replication filtering options follow the same rules for case sensitivity that apply to names of databases and tables elsewhere in the MySQL server, including the effects of the `lower_case_table_names` system variable.

This is a change from previous versions of MySQL. (Bug #51639)

16.2.3.1 Evaluation of Database-Level Replication and Binary Logging Options

When evaluating replication options, the slave begins by checking to see whether there are any `--replicate-do-db` or `--replicate-ignore-db` options that apply. When using `--binlog-do-db` or `--binlog-ignore-db`, the process is similar, but the options are checked on the master.

With statement-based replication, the default database is checked for a match. With row-based replication, the database where data is to be changed is the database that is checked. Regardless of the binary logging format, checking of database-level options proceeds as shown in the following diagram.


START
Which logging format?
STATEMENT
ROW
Test the default database.
Test the database affected by changes.
Any --replicate-do-db options?
yes
no
Does the default database match any of them?
yes
no
Ignore the update. Exit.
yes
Any --replicate-ignore-db options?
yes
no
Does the default database match any of them?
no
Proceed to table-level options, if any.

The steps involved are listed here:

1. Are there any `--replicate-do-db` options?
    - **Yes.** Do any of them match the database?
        - **Yes.** Execute the statement and exit.
        - **No.** Ignore the statement and exit.
    - **No.** Continue to step 2.
2. Are there any `--replicate-ignore-db` options?
    - **Yes.** Do any of them match the database?
        - **Yes.** Ignore the statement and exit.
        - **No.** Continue to step 3.
    - **No.** Continue to step 3.
3. Proceed to checking the table-level replication options, if there are any. For a description of how these options are checked, see Section 16.2.3.2, “Evaluation of Table-Level Replication Options”.

> **Important**
>
> A statement that is still permitted at this stage is not yet actually executed. The statement is not executed until all table-level options (if any) have also been checked, and the outcome of that process permits execution of the statement.

For binary logging, the steps involved are listed here:

1. Are there any `--binlog-do-db` or `--binlog-ignore-db` options?
    - **Yes.** Continue to step 2.
    - **No.** Log the statement and exit.
2. Is there a default database (has any database been selected by `USE`)?
    - **Yes.** Continue to step 3.
    - **No.** Ignore the statement and exit.
3. There is a default database. Are there any `--binlog-do-db` options?
    - **Yes.** Do any of them match the database?
        - **Yes.** Log the statement and exit.
        - **No.** Ignore the statement and exit.
    - **No.** Continue to step 4.
4. Do any of the `--binlog-ignore-db` options match the database?
    - **Yes.** Ignore the statement and exit.
    - **No.** Log the statement and exit.

Important

For statement-based logging, an exception is made in the rules just given for the CREATE DATABASE, ALTER DATABASE, and DROP DATABASE statements. In those cases, the database being *created, altered, or dropped* replaces the default database when determining whether to log or ignore updates.

--binlog-do-db can sometimes mean “ignore other databases”. For example, when using statement-based logging, a server running with only --binlog-do-db=sales does not write to the binary log statements for which the default database differs from sales. When using row-based logging with the same option, the server logs only those updates that change data in sales.

16.2.3.2 Evaluation of Table-Level Replication Options

The slave checks for and evaluates table options only if either of the following two conditions is true:

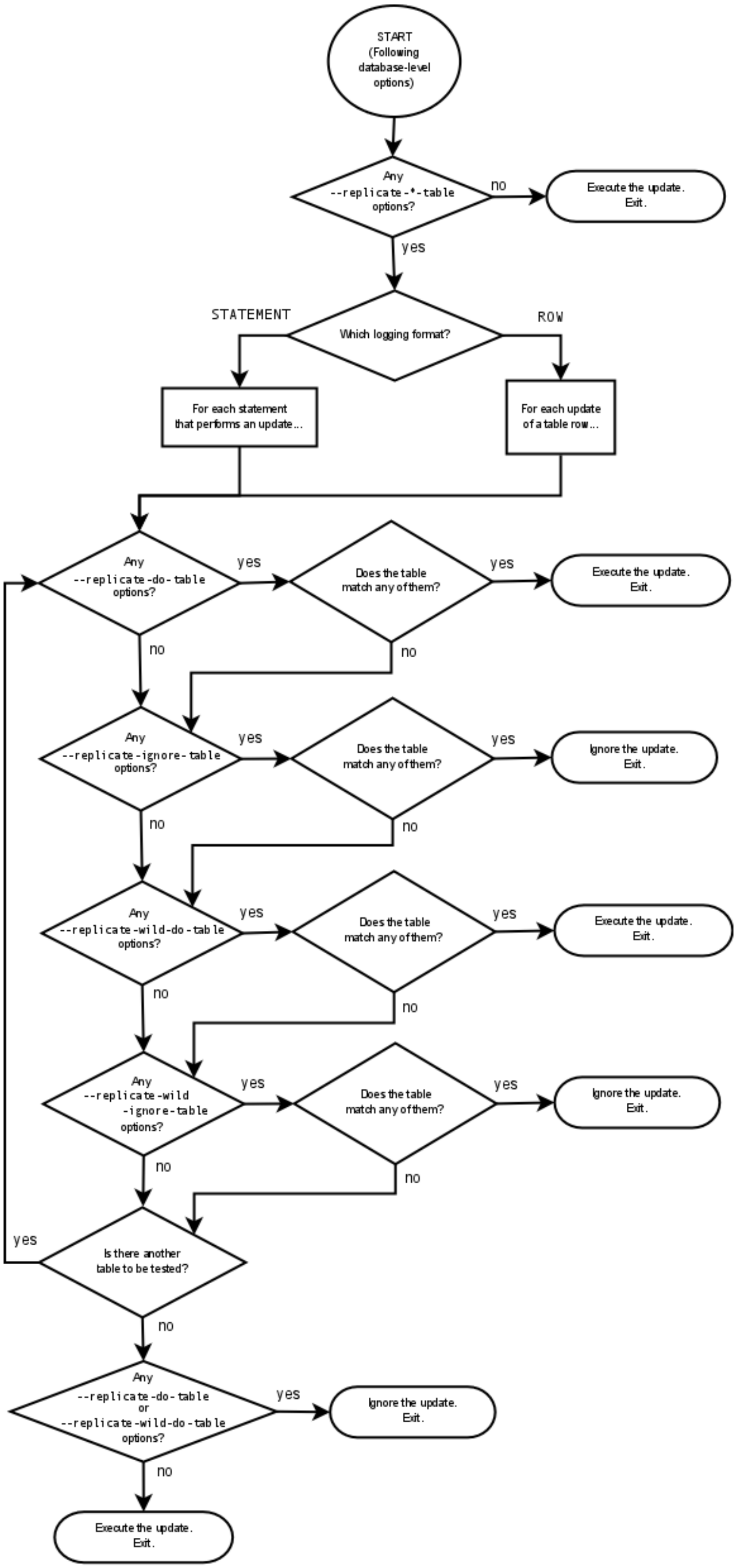
- No matching database options were found.
- One or more database options were found, and were evaluated to arrive at an “execute” condition according to the rules described in the previous section (see Section 16.2.3.1, “Evaluation of Database-Level Replication and Binary Logging Options”).

First, as a preliminary condition, the slave checks whether statement-based replication is enabled. If so, and the statement occurs within a stored function, the slave executes the statement and exits. If row-based replication is enabled, the slave does not know whether a statement occurred within a stored function on the master, so this condition does not apply.

Note

For statement-based replication, replication events represent statements (all changes making up a given event are associated with a single SQL statement); for row-based replication, each event represents a change in a single table row (thus a single statement such as UPDATE mytable SET mycol = 1 may yield many row-based events). When viewed in terms of events, the process of checking table options is the same for both row-based and statement-based replication.

Having reached this point, if there are no table options, the slave simply executes all events. If there are any --replicate-do-table or --replicate-wild-do-table options, the event must match one of these if it is to be executed; otherwise, it is ignored. If there are any --replicate-ignore-table or --replicate-wild-ignore-table options, all events are executed except those that match any of these options. This process is illustrated in the following diagram.


START (Following database-level options)
Any --replicate-*-table options?
no
Execute the update. Exit.
yes
STATEMENT
Which logging format?
ROW
For each statement that performs an update...
For each update of a table row...
Any --replicate-do-table options?
yes
Does the table match any of them?
yes
Execute the update. Exit.
no
no
Any --replicate-ignore-table options?
yes
Does the table match any of them?
yes
Ignore the update. Exit.
no
no
Any --replicate-wild-do-table options?
yes
Does the table match any of them?
yes
Execute the update. Exit.
no
no
Any --replicate-wild-ignore-table options?
yes
Does the table match any of them?
yes
Ignore the update. Exit.
no
no
yes
Is there another table to be tested?
no
Any --replicate-do-table or --replicate-wild-do-table options?
yes
Ignore the update. Exit.
no
Execute the update. Exit.

The following steps describe this evaluation in more detail:

1. Are there any table options?
    - **Yes.** Continue to step 2.
    - **No.** Execute the event and exit.
2. Are there any `--replicate-do-table` options?
    - **Yes.** Does the table match any of them?
        - **Yes.** Execute the event and exit.
        - **No.** Continue to step 3.
    - **No.** Continue to step 3.
3. Are there any `--replicate-ignore-table` options?
    - **Yes.** Does the table match any of them?
        - **Yes.** Ignore the event and exit.
        - **No.** Continue to step 4.
    - **No.** Continue to step 4.
4. Are there any `--replicate-wild-do-table` options?
    - **Yes.** Does the table match any of them?
        - **Yes.** Execute the event and exit.
        - **No.** Continue to step 5.
    - **No.** Continue to step 5.
5. Are there any `--replicate-wild-ignore-table` options?
    - **Yes.** Does the table match any of them?
        - **Yes.** Ignore the event and exit.
        - **No.** Continue to step 6.
    - **No.** Continue to step 6.
6. Are there any `--replicate-do-table` or `--replicate-wild-do-table` options?
    - **Yes.** Ignore the event and exit.
    - **No.** Execute the event and exit.

16.2.3.3 Replication Rule Application

This section provides additional explanation and examples of usage for different combinations of replication filtering options.

Some typical combinations of replication filter rule types are given in the following table:

Condition (Types of Options)	Outcome
No `--replicate-*` options at all:	The slave executes all events that it receives from the master.
`--replicate-*-db` options, but no table options:	The slave accepts or ignores events using the database options. It executes all events permitted by those options because there are no table restrictions.
`--replicate-*-table` options, but no database options:	All events are accepted at the database-checking stage because there are no database conditions. The slave executes or ignores events based solely on the table options.
A combination of database and table options:	The slave accepts or ignores events using the database options. Then it evaluates all events permitted by those options according to the table options. This can sometimes lead to results that seem counterintuitive, and that may be different depending on whether you are using statement-based or row-based replication; see the text for an example.

A more complex example follows, in which we examine the outcomes for both statement-based and row-based settings.

Suppose that we have two tables `mytbl1` in database `db1` and `mytbl2` in database `db2` on the master, and the slave is running with the following options (and no other replication filtering options):

```
replicate-ignore-db = db1
replicate-do-table  = db2.tbl2
```

Now we execute the following statements on the master:

```
USE db1;
INSERT INTO db2.tbl2 VALUES (1);
```

The results on the slave vary considerably depending on the binary log format, and may not match initial expectations in either case.

Statement-based replication. The `USE` statement causes `db1` to be the default database. Thus the `--replicate-ignore-db` option matches, *and the `INSERT` statement is ignored*. The table options are not checked.

Row-based replication. The default database has no effect on how the slave reads database options when using row-based replication. Thus, the `USE` statement makes no difference in how the `--replicate-ignore-db` option is handled: the database specified by this option does not match the database where the `INSERT` statement changes data, so the slave proceeds to check the table options. The table specified by `--replicate-do-table` matches the table to be updated, *and the row is inserted*.

16.3 Replication Solutions

Replication can be used in many different environments for a range of purposes. This section provides general notes and advice on using replication for specific solution types.

For information on using replication in a backup environment, including notes on the setup, backup procedure, and files to back up, see Section 16.3.1, “Using Replication for Backups”.

For advice and tips on using different storage engines on the master and slaves, see Section 16.3.2, “Using Replication with Different Master and Slave Storage Engines”.

Using replication as a scale-out solution requires some changes in the logic and operation of applications that use the solution. See Section 16.3.3, “Using Replication for Scale-Out”.

For performance or data distribution reasons, you may want to replicate different databases to different replication slaves. See Section 16.3.4, “Replicating Different Databases to Different Slaves”

As the number of replication slaves increases, the load on the master can increase and lead to reduced performance (because of the need to replicate the binary log to each slave). For tips on improving your replication performance, including using a single secondary server as an replication master, see Section 16.3.5, “Improving Replication Performance”.

For guidance on switching masters, or converting slaves into masters as part of an emergency failover solution, see Section 16.3.6, “Switching Masters During Failover”.

To secure your replication communication, you can use SSL to encrypt the communication channel. For step-by-step instructions, see Section 16.3.7, “Setting Up Replication Using SSL”.

16.3.1 Using Replication for Backups

To use replication as a backup solution, replicate data from the master to a slave, and then back up the data slave. The slave can be paused and shut down without affecting the running operation of the master, so you can produce an effective snapshot of “live” data that would otherwise require the master to be shut down.

How you back up a database depends on its size and whether you are backing up only the data, or the data and the replication slave state so that you can rebuild the slave in the event of failure. There are therefore two choices:

- If you are using replication as a solution to enable you to back up the data on the master, and the size of your database is not too large, the `mysqldump` tool may be suitable. See Section 16.3.1.1, “Backing Up a Slave Using `mysqldump`”.
- For larger databases, where `mysqldump` would be impractical or inefficient, you can back up the raw data files instead. Using the raw data files option also means that you can back up the binary and relay logs that will enable you to recreate the slave in the event of a slave failure. For more information, see Section 16.3.1.2, “Backing Up Raw Data from a Slave”.

Another backup strategy, which can be used for either master or slave servers, is to put the server in a read-only state. The backup is performed against the read-only server, which then is changed back to its usual read/write operational status. See Section 16.3.1.3, “Backing Up a Master or Slave by Making It Read Only”.

16.3.1.1 Backing Up a Slave Using `mysqldump`

Using `mysqldump` to create a copy of a database enables you to capture all of the data in the database in a format that enables the information to be imported into another instance of MySQL Server (see Section 4.5.4, “`mysqldump` — A Database Backup Program”). Because the format of the information is SQL statements, the file can easily be distributed and applied to running servers in the event that you need access to the data in an emergency. However, if the size of your data set is very large, `mysqldump` may be impractical.

When using `mysqldump`, you should stop replication on the slave before starting the dump process to ensure that the dump contains a consistent set of data:

1. Stop the slave from processing requests. You can stop replication completely on the slave using `mysqladmin`:

   ```
   shell> mysqladmin stop-slave
   ```

Alternatively, you can stop only the slave SQL thread to pause event execution:

```
shell> mysql -e 'STOP SLAVE SQL_THREAD;'
```

This enables the slave to continue to receive data change events from the master's binary log and store them in the relay logs using the I/O thread, but prevents the slave from executing these events and changing its data. Within busy replication environments, permitting the I/O thread to run during backup may speed up the catch-up process when you restart the slave SQL thread.

2. Run `mysqldump` to dump your databases. You may either dump all databases or select databases to be dumped. For example, to dump all databases:

   ```
   shell> mysqldump --all-databases > fulldb.dump
   ```

3. Once the dump has completed, start slave operations again:

   ```
   shell> mysqladmin start-slave
   ```

In the preceding example, you may want to add login credentials (user name, password) to the commands, and bundle the process up into a script that you can run automatically each day.

If you use this approach, make sure you monitor the slave replication process to ensure that the time taken to run the backup does not affect the slave's ability to keep up with events from the master. See Section 16.1.5.1, “Checking Replication Status”. If the slave is unable to keep up, you may want to add another slave and distribute the backup process. For an example of how to configure this scenario, see Section 16.3.4, “Replicating Different Databases to Different Slaves”.

16.3.1.2 Backing Up Raw Data from a Slave

To guarantee the integrity of the files that are copied, backing up the raw data files on your MySQL replication slave should take place while your slave server is shut down. If the MySQL server is still running, background tasks may still be updating the database files, particularly those involving storage engines with background processes such as `InnoDB`. With `InnoDB`, these problems should be resolved during crash recovery, but since the slave server can be shut down during the backup process without affecting the execution of the master it makes sense to take advantage of this capability.

To shut down the server and back up the files:

1. Shut down the slave MySQL server:

   ```
   shell> mysqladmin shutdown
   ```

2. Copy the data files. You can use any suitable copying or archive utility, including `cp`, `tar` or `WinZip`. For example, assuming that the data directory is located under the current directory, you can archive the entire directory as follows:

   ```
   shell> tar cf /tmp/dbbackup.tar ./data
   ```

3. Start the MySQL server again. Under Unix:

   ```
   shell> mysqld_safe &
   ```

   Under Windows:

   ```
   C:\> "C:\Program Files\MySQL\MySQL Server 5.7\bin\mysqld"
   ```

Normally you should back up the entire data directory for the slave MySQL server. If you want to be able to restore the data and operate as a slave (for example, in the event of failure of the slave), then in addition to the slave's data, you should also back up the slave status files, the master info and relay log info repositories, and the relay log files. These files are needed to resume replication after you restore the slave's data.

If you lose the relay logs but still have the `relay-log.info` file, you can check it to determine how far the SQL thread has executed in the master binary logs. Then you can use `CHANGE MASTER TO` with the `MASTER_LOG_FILE` and `MASTER_LOG_POS` options to tell the slave to re-read the binary logs from that point. This requires that the binary logs still exist on the master server.

If your slave is replicating `LOAD DATA INFILE` statements, you should also back up any `SQL_LOAD-*` files that exist in the directory that the slave uses for this purpose. The slave needs these files to resume replication of any interrupted `LOAD DATA INFILE` operations. The location of this directory is the value of the `--slave-load-tmpdir` option. If the server was not started with that option, the directory location is the value of the `tmpdir` system variable.

16.3.1.3 Backing Up a Master or Slave by Making It Read Only

It is possible to back up either master or slave servers in a replication setup by acquiring a global read lock and manipulating the `read_only` system variable to change the read-only state of the server to be backed up:

1. Make the server read-only, so that it processes only retrievals and blocks updates.

2. Perform the backup.

3. Change the server back to its normal read/write state.

Note

The instructions in this section place the server to be backed up in a state that is safe for backup methods that get the data from the server, such as `mysqldump` (see Section 4.5.4, “`mysqldump` — A Database Backup Program”). You should not attempt to use these instructions to make a binary backup by copying files directly because the server may still have modified data cached in memory and not flushed to disk.

The following instructions describe how to do this for a master server and for a slave server. For both scenarios discussed here, suppose that you have the following replication setup:

- A master server M1

- A slave server S1 that has M1 as its master

- A client C1 connected to M1

- A client C2 connected to S1

In either scenario, the statements to acquire the global read lock and manipulate the `read_only` variable are performed on the server to be backed up and do not propagate to any slaves of that server.

Scenario 1: Backup with a Read-Only Master

Put the master M1 in a read-only state by executing these statements on it:

```
mysql> FLUSH TABLES WITH READ LOCK;
mysql> SET GLOBAL read_only = ON;
```

While M1 is in a read-only state, the following properties are true:

- Requests for updates sent by C1 to M1 will block because the server is in read-only mode.
- Requests for query results sent by C1 to M1 will succeed.
- Making a backup on M1 is safe.
- Making a backup on S1 is not safe. This server is still running, and might be processing the binary log or update requests coming from client C2

While M1 is read only, perform the backup. For example, you can use `mysqldump`.

After the backup operation on M1 completes, restore M1 to its normal operational state by executing these statements:

```
mysql> SET GLOBAL read_only = OFF;
mysql> UNLOCK TABLES;
```

Although performing the backup on M1 is safe (as far as the backup is concerned), it is not optimal for performance because clients of M1 are blocked from executing updates.

This strategy applies to backing up a master server in a replication setup, but can also be used for a single server in a nonreplication setting.

Scenario 2: Backup with a Read-Only Slave

Put the slave S1 in a read-only state by executing these statements on it:

```
mysql> FLUSH TABLES WITH READ LOCK;
mysql> SET GLOBAL read_only = ON;
```

While S1 is in a read-only state, the following properties are true:

- The master M1 will continue to operate, so making a backup on the master is not safe.
- The slave S1 is stopped, so making a backup on the slave S1 is safe.

These properties provide the basis for a popular backup scenario: Having one slave busy performing a backup for a while is not a problem because it does not affect the entire network, and the system is still running during the backup. In particular, clients can still perform updates on the master server, which remains unaffected by backup activity on the slave.

While S1 is read only, perform the backup. For example, you can use `mysqldump`.

After the backup operation on S1 completes, restore S1 to its normal operational state by executing these statements:

```
mysql> SET GLOBAL read_only = OFF;
mysql> UNLOCK TABLES;
```

After the slave is restored to normal operation, it again synchronizes to the master by catching up with any outstanding updates from the binary log of the master.

16.3.2 Using Replication with Different Master and Slave Storage Engines

It does not matter for the replication process whether the source table on the master and the replicated table on the slave use different engine types. In fact, the `default_storage_engine` and `storage_engine` system variables are not replicated.

This provides a number of benefits in the replication process in that you can take advantage of different engine types for different replication scenarios. For example, in a typical scale-out scenario (see Section 16.3.3, “Using Replication for Scale-Out”), you want to use `InnoDB` tables on the master to take advantage of the transactional functionality, but use `MyISAM` on the slaves where transaction support is not required because the data is only read. When using replication in a data-logging environment you may want to use the `Archive` storage engine on the slave.

Configuring different engines on the master and slave depends on how you set up the initial replication process:

- If you used `mysqldump` to create the database snapshot on your master, you could edit the dump file text to change the engine type used on each table.

  Another alternative for `mysqldump` is to disable engine types that you do not want to use on the slave before using the dump to build the data on the slave. For example, you can add the `--skip-innodb` option on your slave to disable the `InnoDB` engine. If a specific engine does not exist for a table to be created, MySQL will use the default engine type, usually `MyISAM`. (This requires that the `NO_ENGINE_SUBSTITUTION` SQL mode is not enabled.) If you want to disable additional engines in this way, you may want to consider building a special binary to be used on the slave that only supports the engines you want.

- If you are using raw data files (a binary backup) to set up the slave, you will be unable to change the initial table format. Instead, use `ALTER TABLE` to change the table types after the slave has been started.

- For new master/slave replication setups where there are currently no tables on the master, avoid specifying the engine type when creating new tables.

If you are already running a replication solution and want to convert your existing tables to another engine type, follow these steps:

1. Stop the slave from running replication updates:

   ```
   mysql> STOP SLAVE;
   ```

   This will enable you to change engine types without interruptions.

2. Execute an `ALTER TABLE ... ENGINE='engine_type'` for each table to be changed.

3. Start the slave replication process again:

   ```
   mysql> START SLAVE;
   ```

Although the `default_storage_engine` variable is not replicated, be aware that `CREATE TABLE` and `ALTER TABLE` statements that include the engine specification will be correctly replicated to the slave. For example, if you have a CSV table and you execute:

```
mysql> ALTER TABLE csvtable Engine='MyISAM';
```

The above statement will be replicated to the slave and the engine type on the slave will be converted to `MyISAM`, even if you have previously changed the table type on the slave to an engine other than CSV. If you want to retain engine differences on the master and slave, you should be careful to use the `default_storage_engine` variable on the master when creating a new table. For example, instead of:

```
mysql> CREATE TABLE tablea (columna int) Engine=MyISAM;
```

Use this format:

```
mysql> SET default_storage_engine=MyISAM;
mysql> CREATE TABLE tablea (columna int);
```

When replicated, the `default_storage_engine` variable will be ignored, and the `CREATE TABLE` statement will execute on the slave using the slave's default engine.

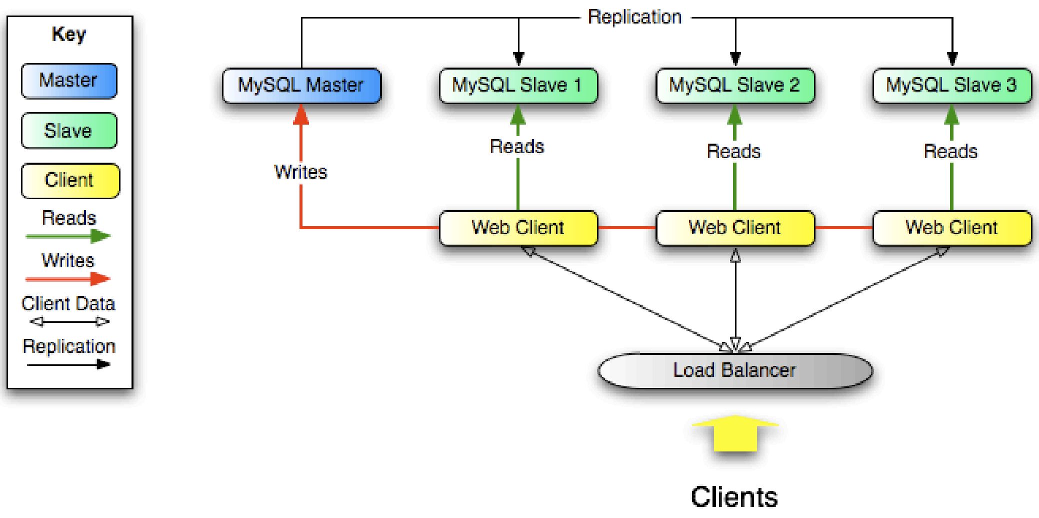
16.3.3 Using Replication for Scale-Out

You can use replication as a scale-out solution; that is, where you want to split up the load of database queries across multiple database servers, within some reasonable limitations.

Because replication works from the distribution of one master to one or more slaves, using replication for scale-out works best in an environment where you have a high number of reads and low number of writes/updates. Most Web sites fit into this category, where users are browsing the Web site, reading articles, posts, or viewing products. Updates only occur during session management, or when making a purchase or adding a comment/message to a forum.

Replication in this situation enables you to distribute the reads over the replication slaves, while still enabling your web servers to communicate with the replication master when a write is required. You can see a sample replication layout for this scenario in Figure 16.1, "Using Replication to Improve Performance During Scale-Out".

Figure 16.1 Using Replication to Improve Performance During Scale-Out



If the part of your code that is responsible for database access has been properly abstracted/modularized, converting it to run with a replicated setup should be very smooth and easy. Change the implementation of your database access to send all writes to the master, and to send reads to either the master or a slave. If your code does not have this level of abstraction, setting up a replicated system gives you the opportunity and motivation to clean it up. Start by creating a wrapper library or module that implements the following functions:

- `safe_writer_connect()`

- safe_reader_connect()
- safe_reader_statement()
- safe_writer_statement()

safe_ in each function name means that the function takes care of handling all error conditions. You can use different names for the functions. The important thing is to have a unified interface for connecting for reads, connecting for writes, doing a read, and doing a write.

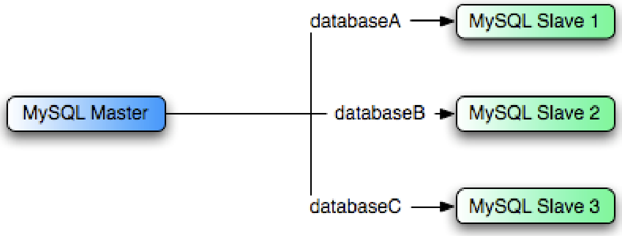
Then convert your client code to use the wrapper library. This may be a painful and scary process at first, but it pays off in the long run. All applications that use the approach just described are able to take advantage of a master/slave configuration, even one involving multiple slaves. The code is much easier to maintain, and adding troubleshooting options is trivial. You need modify only one or two functions; for example, to log how long each statement took, or which statement among those issued gave you an error.

If you have written a lot of code, you may want to automate the conversion task by using the replace utility that comes with standard MySQL distributions, or write your own conversion script. Ideally, your code uses consistent programming style conventions. If not, then you are probably better off rewriting it anyway, or at least going through and manually regularizing it to use a consistent style.

16.3.4 Replicating Different Databases to Different Slaves

There may be situations where you have a single master and want to replicate different databases to different slaves. For example, you may want to distribute different sales data to different departments to help spread the load during data analysis. A sample of this layout is shown in Figure 16.2, “Using Replication to Replicate Databases to Separate Replication Slaves”.

Figure 16.2 Using Replication to Replicate Databases to Separate Replication Slaves



You can achieve this separation by configuring the master and slaves as normal, and then limiting the binary log statements that each slave processes by using the --replicate-wild-do-table configuration option on each slave.

⚠ **Important**

You should *not* use --replicate-do-db for this purpose when using statement-based replication, since statement-based replication causes this option's affects to vary according to the database that is currently selected. This applies to mixed-format replication as well, since this enables some updates to be replicated using the statement-based format.

However, it should be safe to use --replicate-do-db for this purpose if you are using row-based replication only, since in this case the currently selected database has no effect on the option's operation.

For example, to support the separation as shown in Figure 16.2, “Using Replication to Replicate Databases to Separate Replication Slaves”, you should configure each replication slave as follows, before executing `START SLAVE`:

- Replication slave 1 should use `--replicate-wild-do-table=databaseA.%`.
- Replication slave 2 should use `--replicate-wild-do-table=databaseB.%`.
- Replication slave 3 should use `--replicate-wild-do-table=databaseC.%`.

Each slave in this configuration receives the entire binary log from the master, but executes only those events from the binary log that apply to the databases and tables included by the `--replicate-wild-do-table` option in effect on that slave.

If you have data that must be synchronized to the slaves before replication starts, you have a number of choices:

- Synchronize all the data to each slave, and delete the databases, tables, or both that you do not want to keep.
- Use `mysqldump` to create a separate dump file for each database and load the appropriate dump file on each slave.
- Use a raw data file dump and include only the specific files and databases that you need for each slave.

Note

This does not work with `InnoDB` databases unless you use `innodb_file_per_table`.

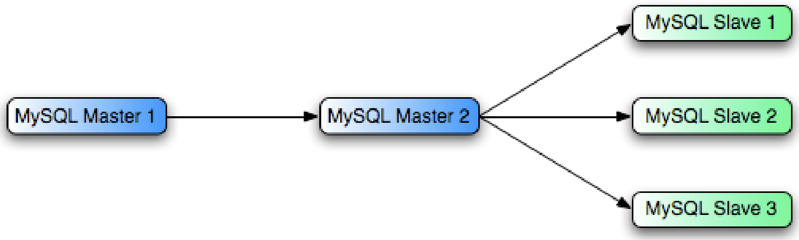
16.3.5 Improving Replication Performance

As the number of slaves connecting to a master increases, the load, although minimal, also increases, as each slave uses a client connection to the master. Also, as each slave must receive a full copy of the master binary log, the network load on the master may also increase and create a bottleneck.

If you are using a large number of slaves connected to one master, and that master is also busy processing requests (for example, as part of a scale-out solution), then you may want to improve the performance of the replication process.

One way to improve the performance of the replication process is to create a deeper replication structure that enables the master to replicate to only one slave, and for the remaining slaves to connect to this primary slave for their individual replication requirements. A sample of this structure is shown in Figure 16.3, “Using an Additional Replication Host to Improve Performance”.

Figure 16.3 Using an Additional Replication Host to Improve Performance



For this to work, you must configure the MySQL instances as follows:

- Master 1 is the primary master where all changes and updates are written to the database. Binary logging should be enabled on this machine.
- Master 2 is the slave to the Master 1 that provides the replication functionality to the remainder of the slaves in the replication structure. Master 2 is the only machine permitted to connect to Master 1. Master 2 also has binary logging enabled, and the `--log-slave-updates` option so that replication instructions from Master 1 are also written to Master 2's binary log so that they can then be replicated to the true slaves.
- Slave 1, Slave 2, and Slave 3 act as slaves to Master 2, and replicate the information from Master 2, which actually consists of the upgrades logged on Master 1.

The above solution reduces the client load and the network interface load on the primary master, which should improve the overall performance of the primary master when used as a direct database solution.

If your slaves are having trouble keeping up with the replication process on the master, there are a number of options available:

- If possible, put the relay logs and the data files on different physical drives. To do this, use the `--relay-log` option to specify the location of the relay log.
- If the slaves are significantly slower than the master, you may want to divide up the responsibility for replicating different databases to different slaves. See Section 16.3.4, “Replicating Different Databases to Different Slaves”.
- If your master makes use of transactions and you are not concerned about transaction support on your slaves, use `MyISAM` or another nontransactional engine on the slaves. See Section 16.3.2, “Using Replication with Different Master and Slave Storage Engines”.
- If your slaves are not acting as masters, and you have a potential solution in place to ensure that you can bring up a master in the event of failure, then you can switch off `--log-slave-updates`. This prevents “dumb” slaves from also logging events they have executed into their own binary log.

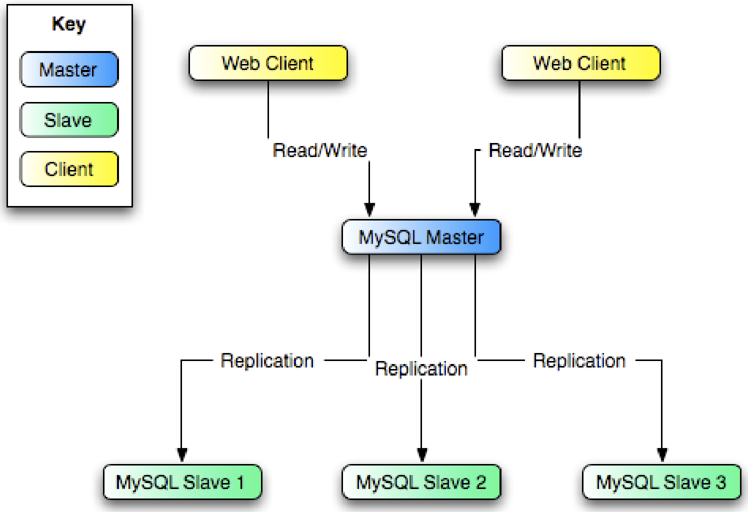
16.3.6 Switching Masters During Failover

When using replication with GTIDs (see Section 16.1.3, “Replication with Global Transaction Identifiers”), you can provide failover between master and slaves in the event of a failure using `mysqlfailover`, which is provided by the MySQL Utilities; see `mysqlfailover` — Automatic replication health monitoring and failover, for more information. Otherwise, you must set up a master and one or more slaves; then, you need to write an application or script that monitors the master to check whether it is up, and instructs the slaves and applications to change to another master in case of failure. This section discusses some of the issues encountered when setting up failover in this fashion.

You can tell a slave to change to a new master using the `CHANGE MASTER TO` statement. The slave does not check whether the databases on the master are compatible with those on the slave; it simply begins reading and executing events from the specified coordinates in the new master's binary log. In a failover situation, all the servers in the group are typically executing the same events from the same binary log file, so changing the source of the events should not affect the structure or integrity of the database, provided that you exercise care in making the change.

Slaves should be run with the `--log-bin` option and without `--log-slave-updates`. In this way, the slave is ready to become a master without restarting the slave `mysqld`. Assume that you have the structure shown in Figure 16.4, “Redundancy Using Replication, Initial Structure”.

Figure 16.4 Redundancy Using Replication, Initial Structure



In this diagram, the MySQL Master holds the master database, the MySQL Slave hosts are replication slaves, and the Web Client machines are issuing database reads and writes. Web clients that issue only reads (and would normally be connected to the slaves) are not shown, as they do not need to switch to a new server in the event of failure. For a more detailed example of a read/write scale-out replication structure, see Section 16.3.3, "Using Replication for Scale-Out".

Each MySQL Slave (Slave 1, Slave 2, and Slave 3) is a slave running with --log-bin and without --log-slave-updates. Because updates received by a slave from the master are not logged in the binary log unless --log-slave-updates is specified, the binary log on each slave is empty initially. If for some reason MySQL Master becomes unavailable, you can pick one of the slaves to become the new master. For example, if you pick Slave 1, all Web Clients should be redirected to Slave 1, which writes the updates to its binary log. Slave 2 and Slave 3 should then replicate from Slave 1.

The reason for running the slave without --log-slave-updates is to prevent slaves from receiving updates twice in case you cause one of the slaves to become the new master. If Slave 1 has --log-slave-updates enabled, it writes any updates that it receives from Master in its own binary log. This means that, when Slave 2 changes from Master to Slave 1 as its master, it may receive updates from Slave 1 that it has already received from Master.

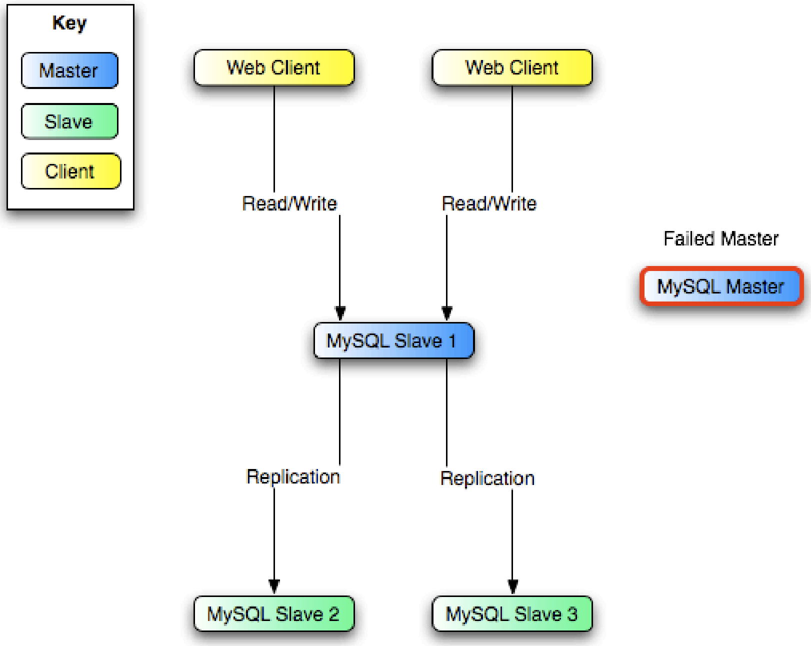
Make sure that all slaves have processed any statements in their relay log. On each slave, issue STOP SLAVE IO_THREAD, then check the output of SHOW PROCESSLIST until you see Has read all relay log. When this is true for all slaves, they can be reconfigured to the new setup. On the slave Slave 1 being promoted to become the master, issue STOP SLAVE and RESET MASTER.

On the other slaves Slave 2 and Slave 3, use STOP SLAVE and CHANGE MASTER TO MASTER_HOST='Slave1' (where 'Slave1' represents the real host name of Slave 1). To use CHANGE MASTER TO, add all information about how to connect to Slave 1 from Slave 2 or Slave 3 (*user*, *password*, *port*). When issuing the CHANGE MASTER TO statement in this, there is no need to specify the name of the Slave 1 binary log file or log position to read from, since the first binary log file and position 4, are the defaults. Finally, execute START SLAVE on Slave 2 and Slave 3.

Once the new replication setup is in place, you need to tell each Web Client to direct its statements to Slave 1. From that point on, all updates statements sent by Web Client to Slave 1 are written to the binary log of Slave 1, which then contains every update statement sent to Slave 1 since Master died.

The resulting server structure is shown in Figure 16.5, “Redundancy Using Replication, After Master Failure”.

Figure 16.5 Redundancy Using Replication, After Master Failure



When Master becomes available again, you should make it a slave of Slave 1. To do this, issue on Master the same CHANGE MASTER TO statement as that issued on Slave 2 and Slave 3 previously. Master then becomes a slave of S1ave 1 and picks up the Web Client writes that it missed while it was offline.

To make Master a master again (for example, because it is the most powerful machine), use the preceding procedure as if Slave 1 was unavailable and Master was to be the new master. During this procedure, do not forget to run RESET MASTER on Master before making Slave 1, Slave 2, and Slave 3 slaves of Master. If you fail to do this, the slaves may pick up stale writes from the Web Client applications dating from before the point at which Master became unavailable.

You should be aware that that there is no synchronization between slaves, even when they share the same master, and thus some slaves might be considerably ahead of others. This means that in some cases the procedure outlined in the previous example might not work as expected. In practice, however, relay logs on all slaves should be relatively close together.

One way to keep applications informed about the location of the master is to have a dynamic DNS entry for the master. With bind you can use nsupdate to update the DNS dynamically.

16.3.7 Setting Up Replication Using SSL

To use SSL for encrypting the transfer of the binary log required during replication, both the master and the slave must support SSL network connections. If either host does not support SSL connections (because it has not been compiled or configured for SSL), replication through an SSL connection is not possible.

Setting up replication using an SSL connection is similar to setting up a server and client using SSL. You must obtain (or create) a suitable security certificate that you can use on the master, and a similar certificate (from the same certificate authority) on each slave.

For more information on setting up a server and client for SSL connectivity, see Section 6.3.11.2, “Configuring MySQL for SSL”.

To enable SSL on the master you must create or obtain suitable certificates, and then add the following configuration options to the master's configuration within the `[mysqld]` section of the master's `my.cnf` file:

```
[mysqld]
ssl-ca=cacert.pem
ssl-cert=server-cert.pem
ssl-key=server-key.pem
```

The paths to the certificates may be relative or absolute; we recommend that you always use complete paths for this purpose.

The options are as follows:

- `ssl-ca` identifies the Certificate Authority (CA) certificate.
- `ssl-cert` identifies the server public key. This can be sent to the client and authenticated against the CA certificate that it has.
- `ssl-key` identifies the server private key.

On the slave, you have two options available for setting the SSL information. You can either add the slave certificates to the `[client]` section of the slave's `my.cnf` file, or you can explicitly specify the SSL information using the `CHANGE MASTER TO` statement:

- To add the slave certificates using an option file, add the following lines to the `[client]` section of the slave's `my.cnf` file:

  ```
  [client]
  ssl-ca=cacert.pem
  ssl-cert=client-cert.pem
  ssl-key=client-key.pem
  ```

  Restart the slave server, using the `--skip-slave-start` option to prevent the slave from connecting to the master. Use `CHANGE MASTER TO` to specify the master configuration, using the `MASTER_SSL` option to enable SSL connectivity:

  ```
  mysql> CHANGE MASTER TO
      -> MASTER_HOST='master_hostname',
      -> MASTER_USER='replicate',
      -> MASTER_PASSWORD='password',
      -> MASTER_SSL=1;
  ```

- To specify the SSL certificate options using the `CHANGE MASTER TO` statement, append the SSL options:

  ```
  mysql> CHANGE MASTER TO
      -> MASTER_HOST='master_hostname',
      -> MASTER_USER='replicate',
      -> MASTER_PASSWORD='password',
      -> MASTER_SSL=1,
      -> MASTER_SSL_CA = 'ca_file_name',
      -> MASTER_SSL_CAPATH = 'ca_directory_name',
      -> MASTER_SSL_CERT = 'cert_file_name',
      -> MASTER_SSL_KEY = 'key_file_name';
  ```

After the master information has been updated, start the slave replication process:

```
mysql> START SLAVE;
```

You can use the SHOW SLAVE STATUS statement to confirm that the SSL connection was established successfully.

For more information on the CHANGE MASTER TO statement, see Section 13.4.2.1, “CHANGE MASTER TO Syntax”.

If you want to enforce the use of SSL connections during replication, then create a user with the REPLICATION SLAVE privilege and use the REQUIRE SSL option for that user. For example:

```
mysql> CREATE USER 'repl'@'%.mydomain.com' IDENTIFIED BY 'slavepass';
mysql> GRANT REPLICATION SLAVE ON *.*
    -> TO 'repl'@'%.mydomain.com' REQUIRE SSL;
```

If the account already exists, you can add REQUIRE SSL to it with this statement:

```
mysql> GRANT USAGE ON *.*
    -> TO 'repl'@'%.mydomain.com' REQUIRE SSL;
```

16.3.8 Semisynchronous Replication

In addition to the built-in asynchronous replication, MySQL 5.7 supports an interface to semisynchronous replication that is implemented by plugins. This section discusses what semisynchronous replication is and how it works. The following sections cover the administrative interface to semisynchronous replication and how to install, configure, and monitor it.

MySQL replication by default is asynchronous. The master writes events to its binary log but does not know whether or when a slave has retrieved and processed them. With asynchronous replication, if the master crashes, transactions that it has committed might not have been transmitted to any slave. Consequently, failover from master to slave in this case may result in failover to a server that is missing transactions relative to the master.

Semisynchronous replication can be used as an alternative to asynchronous replication:

- A slave indicates whether it is semisynchronous-capable when it connects to the master.
- If semisynchronous replication is enabled on the master side and there is at least one semisynchronous slave, a thread that performs a transaction commit on the master blocks and waits until at least one semisynchronous slave acknowledges that it has received all events for the transaction, or until a timeout occurs.
- The slave acknowledges receipt of a transaction's events only after the events have been written to its relay log and flushed to disk.
- If a timeout occurs without any slave having acknowledged the transaction, the master reverts to asynchronous replication. When at least one semisynchronous slave catches up, the master returns to semisynchronous replication.
- Semisynchronous replication must be enabled on both the master and slave sides. If semisynchronous replication is disabled on the master, or enabled on the master but on no slaves, the master uses asynchronous replication.

While the master is blocking (waiting for acknowledgment from a slave), it does not return to the session that performed the transaction. When the block ends, the master returns to the session, which then can proceed to execute other statements. At this point, the transaction has committed on the master side, and receipt of its events has been acknowledged by at least one slave.

As of MySQL 5.7.3, the number of slave acknowledgments the master must receive per transaction before proceeding is configurable using the `rpl_semi_sync_master_wait_for_slave_count` system variable. The default value is 1.

Blocking also occurs after rollbacks that are written to the binary log, which occurs when a transaction that modifies nontransactional tables is rolled back. The rolled-back transaction is logged even though it has no effect for transactional tables because the modifications to the nontransactional tables cannot be rolled back and must be sent to slaves.

For statements that do not occur in transactional context (that is, when no transaction has been started with `START TRANSACTION` or `SET autocommit = 0`), autocommit is enabled and each statement commits implicitly. With semisynchronous replication, the master blocks for each such statement, just as it does for explicit transaction commits.

To understand what the “semi” in “semisynchronous replication” means, compare it with asynchronous and fully synchronous replication:

- With asynchronous replication, the master writes events to its binary log and slaves request them when they are ready. There is no guarantee that any event will ever reach any slave.

- With fully synchronous replication, when a master commits a transaction, all slaves also will have committed the transaction before the master returns to the session that performed the transaction. The drawback of this is that there might be a lot of delay to complete a transaction.

- Semisynchronous replication falls between asynchronous and fully synchronous replication. The master waits only until at least one slave has received and logged the events. It does not wait for all slaves to acknowledge receipt, and it requires only receipt, not that the events have been fully executed and committed on the slave side.

Compared to asynchronous replication, semisynchronous replication provides improved data integrity. When a commit returns successfully, it is known that the data exists in at least two places (on the master and at least one slave). If the master commits but a crash occurs while the master is waiting for acknowledgment from a slave, it is possible that the transaction may not have reached any slave.

Semisynchronous replication also places a rate limit on busy sessions by constraining the speed at which binary log events can be sent from master to slave. When one user is too busy, this will slow it down, which is useful in some deployment situations.

Semisynchronous replication does have some performance impact because commits are slower due to the need to wait for slaves. This is the tradeoff for increased data integrity. The amount of slowdown is at least the TCP/IP roundtrip time to send the commit to the slave and wait for the acknowledgment of receipt by the slave. This means that semisynchronous replication works best for close servers communicating over fast networks, and worst for distant servers communicating over slow networks.

The `rpl_semi_sync_master_wait_point` system variable controls the point at which a semisynchronous replication master waits for slave acknowledgment of transaction receipt before returning a status to the client that committed the transaction. These values are permitted:

- `AFTER_SYNC` (the default): The master writes each transaction to its binary log and the slave, and syncs the binary log to disk. The master waits for slave acknowledgment of transaction receipt after the sync.

Upon receiving acknowledgment, the master commits the transaction to the storage engine and returns a result to the client, which then can proceed.

- `AFTER_COMMIT`: The master writes each transaction to its binary log and the slave, syncs the binary log, and commits the transaction to the storage engine. The master waits for slave acknowledgment of transaction receipt after the commit. Upon receiving acknowledgment, the master returns a result to the client, which then can proceed.

The replication characteristics of these settings differ as follows:

- With `AFTER_SYNC`, all clients see the committed transaction at the same time: After it has been acknowledged by the slave and committed to the storage engine on the master. Thus, all clients see the same data on the master.

  In the event of master failure, all transactions committed on the master have been replicated to the slave (saved to its relay log). A crash of the master and failover to the slave is lossless because the slave is up to date.

- With `AFTER_COMMIT`, the client issuing the transaction gets a return status only after the server commits to the storage engine and receives slave acknowledgment. After the commit and before slave acknowledgment, other clients can see the committed transaction before the committing client.

  If something goes wrong such that the slave does not process the transaction, then in the event of a master crash and failover to the slave, it is possible that such clients will see a loss of data relative to what they saw on the master.

16.3.8.1 Semisynchronous Replication Administrative Interface

The administrative interface to semisynchronous replication has several components:

- Two plugins implement semisynchronous capability. There is one plugin for the master side and one for the slave side.

- System variables control plugin behavior. Some examples:

  - `rpl_semi_sync_master_enabled`

    Controls whether semisynchronous replication is enabled on the master. To enable or disable the plugin, set this variable to 1 or 0, respectively. The default is 0 (off).

  - `rpl_semi_sync_master_timeout`

    A value in milliseconds that controls how long the master waits on a commit for acknowledgment from a slave before timing out and reverting to asynchronous replication. The default value is 10000 (10 seconds).

  - `rpl_semi_sync_slave_enabled`

    Similar to `rpl_semi_sync_master_enabled`, but controls the slave plugin.

  All `rpl_semi_sync_xxx` system variables are described at Section 5.1.4, “Server System Variables”.

- Status variables enable semisynchronous replication monitoring. Some examples:

  - `Rpl_semi_sync_master_clients`

    The number of semisynchronous slaves.

- Rpl_semi_sync_master_status

  Whether semisynchronous replication currently is operational on the master. The value is 1 if the plugin has been enabled and a commit acknowledgment has not occurred. It is 0 if the plugin is not enabled or the master has fallen back to asynchronous replication due to commit acknowledgment timeout.

- Rpl_semi_sync_master_no_tx

  The number of commits that were not acknowledged successfully by a slave.

- Rpl_semi_sync_master_yes_tx

  The number of commits that were acknowledged successfully by a slave.

- Rpl_semi_sync_slave_status

  Whether semisynchronous replication currently is operational on the slave. This is 1 if the plugin has been enabled and the slave I/O thread is running, 0 otherwise.

All Rpl_semi_sync_*xxx* status variables are described at Section 5.1.6, "Server Status Variables".

The system and status variables are available only if the appropriate master or slave plugin has been installed with INSTALL PLUGIN.

16.3.8.2 Semisynchronous Replication Installation and Configuration

Semisynchronous replication is implemented using plugins, so the plugins must be installed into the server to make them available. After a plugin has been installed, you control it by means of the system variables associated with it. These system variables are unavailable until the associated plugin has been installed.

To use semisynchronous replication, the following requirements must be satisfied:

- MySQL 5.5 or higher must be installed.
- The capability of installing plugins requires a MySQL server that supports dynamic loading. To verify this, check that the value of the have_dynamic_loading system variable is YES. Binary distributions should support dynamic loading.
- Replication must already be working. For information on creating a master/slave relationship, see Section 16.1.1, "How to Set Up Replication".

To set up semisynchronous replication, use the following instructions. The INSTALL PLUGIN, SET GLOBAL, STOP SLAVE, and START SLAVE statements mentioned here require the SUPER privilege.

The semisynchronous replication plugins are included with MySQL distributions.

Unpack the component distribution, which contains files for the master side and the slave side.

Install the component files in the plugin directory of the appropriate server. Install the semisync_master* files in the plugin directory of the master server. Install the semisync_slave* files in the plugin directory of each slave server. The location of the plugin directory is available as the value of the server's plugin_dir system variable.

To load the plugins, use the INSTALL PLUGIN statement on the master and on each slave that is to be semisynchronous.

On the master:

```
mysql> INSTALL PLUGIN rpl_semi_sync_master SONAME 'semisync_master.so';
```

On each slave:

```
mysql> INSTALL PLUGIN rpl_semi_sync_slave SONAME 'semisync_slave.so';
```

The preceding commands use a plugin file name suffix of `.so`. A different suffix might apply on your system. If you are not sure about the plugin file name, look for the plugins in the server's plugin directory.

If an attempt to install a plugin results in an error on Linux similar to that shown here, you will need to install `libimf`:

```
mysql> INSTALL PLUGIN rpl_semi_sync_master SONAME 'semisync_master.so';
ERROR 1126 (HY000): Can't open shared library
'/usr/local/mysql/lib/plugin/semisync_master.so' (errno: 22 libimf.so: cannot open
shared object file: No such file or directory)
```

You can obtain `libimf` from http://dev.mysql.com/downloads/os-linux.html.

To see which plugins are installed, use the `SHOW PLUGINS` statement, or query the `INFORMATION_SCHEMA.PLUGINS` table.

After a semisynchronous replication plugin has been installed, it is disabled by default. The plugins must be enabled both on the master side and the slave side to enable semisynchronous replication. If only one side is enabled, replication will be asynchronous.

To control whether an installed plugin is enabled, set the appropriate system variables. You can set these variables at runtime using `SET GLOBAL`, or at server startup on the command line or in an option file.

At runtime, these master-side system variables are available:

```
mysql> SET GLOBAL rpl_semi_sync_master_enabled = {0|1};
mysql> SET GLOBAL rpl_semi_sync_master_timeout = N;
```

On the slave side, this system variable is available:

```
mysql> SET GLOBAL rpl_semi_sync_slave_enabled = {0|1};
```

For `rpl_semi_sync_master_enabled` or `rpl_semi_sync_slave_enabled`, the value should be 1 to enable semisynchronous replication or 0 to disable it. By default, these variables are set to 1.

For `rpl_semi_sync_master_timeout`, the value *`N`* is given in milliseconds. The default value is 10000 (10 seconds).

If you enable semisynchronous replication on a slave at runtime, you must also start the slave I/O thread (stopping it first if it is already running) to cause the slave to connect to the master and register as a semisynchronous slave:

```
mysql> STOP SLAVE IO_THREAD; START SLAVE IO_THREAD;
```

If the I/O thread is already running and you do not restart it, the slave continues to use asynchronous replication.

At server startup, the variables that control semisynchronous replication can be set as command-line options or in an option file. A setting listed in an option file takes effect each time the server starts. For example, you can set the variables in `my.cnf` files on the master and slave sides as follows.

On the master:

```
[mysqld]
rpl_semi_sync_master_enabled=1
rpl_semi_sync_master_timeout=1000 # 1 second
```

On each slave:

```
[mysqld]
rpl_semi_sync_slave_enabled=1
```

16.3.8.3 Semisynchronous Replication Monitoring

The plugins for the semisynchronous replication capability expose several system and status variables that you can examine to determine its configuration and operational state.

The system variable reflect how semisynchronous replication is configured. To check their values, use SHOW VARIABLES:

```
mysql> SHOW VARIABLES LIKE 'rpl_semi_sync%';
```

The status variables enable you to monitor the operation of semisynchronous replication. To check their values, use SHOW STATUS:

```
mysql> SHOW STATUS LIKE 'Rpl_semi_sync%';
```

When the master switches between asynchronous or semisynchronous replication due to commit-blocking timeout or a slave catching up, it sets the value of the Rpl_semi_sync_master_status status variable appropriately. Automatic fallback from semisynchronous to asynchronous replication on the master means that it is possible for the rpl_semi_sync_master_enabled system variable to have a value of 1 on the master side even when semisynchronous replication is in fact not operational at the moment. You can monitor the Rpl_semi_sync_master_status status variable to determine whether the master currently is using asynchronous or semisynchronous replication.

To see how many semisynchronous slaves are connected, check Rpl_semi_sync_master_clients.

The number of commits that have been acknowledged successfully or unsuccessfully by slaves are indicated by the Rpl_semi_sync_master_yes_tx and Rpl_semi_sync_master_no_tx variables.

On the slave side, Rpl_semi_sync_slave_status indicates whether semisynchronous replication currently is operational.

16.3.9 Delayed Replication

MySQL 5.7 supports delayed replication such that a slave server deliberately lags behind the master by at least a specified amount of time. The default delay is 0 seconds. Use the MASTER_DELAY option for CHANGE MASTER TO to set the delay to *N* seconds:

```
CHANGE MASTER TO MASTER_DELAY = N;
```

An event received from the master is not executed until at least *N* seconds later than its execution on the master. The exceptions are that there is no delay for format description events or log file rotation events, which affect only the internal state of the SQL thread.

Delayed replication can be used for several purposes:

- To protect against user mistakes on the master. A DBA can roll back a delayed slave to the time just before the disaster.
- To test how the system behaves when there is a lag. For example, in an application, a lag might be caused by a heavy load on the slave. However, it can be difficult to generate this load level. Delayed replication can simulate the lag without having to simulate the load. It can also be used to debug conditions related to a lagging slave.
- To inspect what the database looked like long ago, without having to reload a backup. For example, if the delay is one week and the DBA needs to see what the database looked like before the last few days' worth of development, the delayed slave can be inspected.

`START SLAVE` and `STOP SLAVE` take effect immediately and ignore any delay. `RESET SLAVE` resets the delay to 0.

`SHOW SLAVE STATUS` has three fields that provide information about the delay:

- `SQL_Delay`: A nonnegative integer indicating the number of seconds that the slave must lag the master.
- `SQL_Remaining_Delay`: When `Slave_SQL_Running_State` is `Waiting until MASTER_DELAY seconds after master executed event`, this field contains an integer indicating the number of seconds left of the delay. At other times, this field is `NULL`.
- `Slave_SQL_Running_State`: A string indicating the state of the SQL thread (analogous to `Slave_IO_State`). The value is identical to the `State` value of the SQL thread as displayed by `SHOW PROCESSLIST`.

When the slave SQL thread is waiting for the delay to elapse before executing an event, `SHOW PROCESSLIST` displays its `State` value as `Waiting until MASTER_DELAY seconds after master executed event`.

16.4 Replication Notes and Tips

16.4.1 Replication Features and Issues

The following sections provide information about what is supported and what is not in MySQL replication, and about specific issues and situations that may occur when replicating certain statements.

Statement-based replication depends on compatibility at the SQL level between the master and slave. In others, successful SBR requires that any SQL features used be supported by both the master and the slave servers. For example, if you use a feature on the master server that is available only in MySQL 5.7 (or later), you cannot replicate to a slave that uses MySQL 5.6 (or earlier).

Such incompatibilities also can occur within a release series when using pre-production releases of MySQL. For example, the `SLEEP()` function is available beginning with MySQL 5.0.12. If you use this function on the master, you cannot replicate to a slave that uses MySQL 5.0.11 or earlier.

For this reason, use Generally Available (GA) releases of MySQL for statement-based replication in a production setting, since we do not introduce new SQL statements or change their behavior within a given release series once that series reaches GA release status.

If you are planning to use statement-based replication between MySQL 5.7 and a previous MySQL release series, it is also a good idea to consult the edition of the *MySQL Reference Manual* corresponding to the earlier release series for information regarding the replication characteristics of that series.

With MySQL's statement-based replication, there may be issues with replicating stored routines or triggers. You can avoid these issues by using MySQL's row-based replication instead. For a detailed list of issues, see Section 18.7, “Binary Logging of Stored Programs”. For more information about row-based logging and row-based replication, see Section 5.2.4.1, “Binary Logging Formats”, and Section 16.1.2, “Replication Formats”.

For additional information specific to replication and InnoDB, see Section 14.2.15, “InnoDB and MySQL Replication”. For information relating to replication with MySQL Cluster, see MySQL Cluster Replication.

16.4.1.1 Replication and AUTO_INCREMENT

Statement-based replication of AUTO_INCREMENT, LAST_INSERT_ID(), and TIMESTAMP values is done correctly, subject to the following exceptions:

- When using statement-based replication prior to MySQL 5.7.1, AUTO_INCREMENT columns in tables on the slave must match the same columns on the master; that is, AUTO_INCREMENT columns must be replicated to AUTO_INCREMENT columns.

- A statement invoking a trigger or function that causes an update to an AUTO_INCREMENT column is not replicated correctly using statement-based replication. In MySQL 5.7, such statements are marked as unsafe. (Bug #45677)

- An INSERT into a table that has a composite primary key that includes an AUTO_INCREMENT column that is not the first column of this composite key is not safe for statement-based logging or replication. In MySQL 5.7 and later, such statements are marked as unsafe. (Bug #11754117, Bug #45670)

  This issue does not affect tables using the InnoDB storage engine, since an InnoDB table with an AUTO_INCREMENT column requires at least one key where the auto-increment column is the only or leftmost column.

- Adding an AUTO_INCREMENT column to a table with ALTER TABLE might not produce the same ordering of the rows on the slave and the master. This occurs because the order in which the rows are numbered depends on the specific storage engine used for the table and the order in which the rows were inserted. If it is important to have the same order on the master and slave, the rows must be ordered before assigning an AUTO_INCREMENT number. Assuming that you want to add an AUTO_INCREMENT column to a table t1 that has columns col1 and col2, the following statements produce a new table t2 identical to t1 but with an AUTO_INCREMENT column:

  ```
  CREATE TABLE t2 LIKE t1;
  ALTER TABLE t2 ADD id INT AUTO_INCREMENT PRIMARY KEY;
  INSERT INTO t2 SELECT * FROM t1 ORDER BY col1, col2;
  ```

  Important

  To guarantee the same ordering on both master and slave, the ORDER BY clause must name *all* columns of t1.

  The instructions just given are subject to the limitations of CREATE TABLE ... LIKE: Foreign key definitions are ignored, as are the DATA DIRECTORY and INDEX DIRECTORY table options. If a table definition includes any of those characteristics, create t2 using a CREATE TABLE statement that is identical to the one used to create t1, but with the addition of the AUTO_INCREMENT column.

  Regardless of the method used to create and populate the copy having the AUTO_INCREMENT column, the final step is to drop the original table and then rename the copy:

  ```
  DROP t1;
  ```

```
ALTER TABLE t2 RENAME t1;
```

See also Section C.5.7.1, “Problems with ALTER TABLE”.

16.4.1.2 Replication and BLACKHOLE Tables

The BLACKHOLE storage engine accepts data but discards it and does not store it. When performing binary logging, all inserts to such tables are always logged, regardless of the logging format in use. Updates and deletes are handled differently depending on whether statement based or row based logging is in use. With the statement based logging format, all statements affecting BLACKHOLE tables are logged, but their effects ignored. When using row-based logging, updates and deletes to such tables are simply skipped—they are not written to the binary log. In MySQL 5.7.2 and later, a warning is logged whenever this occurs (Bug #13004581)

For this reason we recommend when you replicate to tables using the BLACKHOLE storage engine that you have the binlog_format server variable set to STATEMENT, and not to either ROW or MIXED.

16.4.1.3 Replication and Character Sets

The following applies to replication between MySQL servers that use different character sets:

- If the master uses MySQL 4.1, you must *always* use the same *global* character set and collation on the master and the slave, regardless of the slave MySQL version. (These are controlled by the --character-set-server and --collation-server options.) Otherwise, you may get duplicate-key errors on the slave, because a key that is unique in the master character set might not be unique in the slave character set. Note that this is not a cause for concern when master and slave are both MySQL 5.0 or later.

- If the master is older than MySQL 4.1.3, the character set of any client should never be made different from its global value because this character set change is not known to the slave. In other words, clients should not use SET NAMES, SET CHARACTER SET, and so forth. If both the master and the slave are 4.1.3 or newer, clients can freely set session values for character set variables because these settings are written to the binary log and so are known to the slave. That is, clients can use SET NAMES or SET CHARACTER SET or can set variables such as collation_client or collation_server. However, clients are prevented from changing the *global* value of these variables; as stated previously, the master and slave must always have identical global character set values. This is true whether you are using statement-based or row-based replication.

- If the master has databases with a character set different from the global character_set_server value, you should design your CREATE TABLE statements so that they do not implicitly rely on the database default character set. A good workaround is to state the character set and collation explicitly in CREATE TABLE statements.

16.4.1.4 Replication of CREATE ... IF NOT EXISTS Statements

MySQL applies these rules when various CREATE ... IF NOT EXISTS statements are replicated:

- Every CREATE DATABASE IF NOT EXISTS statement is replicated, whether or not the database already exists on the master.

- Similarly, every CREATE TABLE IF NOT EXISTS statement without a SELECT is replicated, whether or not the table already exists on the master. This includes CREATE TABLE IF NOT EXISTS ... LIKE. Replication of CREATE TABLE IF NOT EXISTS ... SELECT follows somewhat different rules; see Section 16.4.1.5, “Replication of CREATE TABLE ... SELECT Statements”, for more information.

- CREATE EVENT IF NOT EXISTS is always replicated in MySQL 5.7, whether or not the event named in the statement already exists on the master.

See also Bug #45574.

16.4.1.5 Replication of CREATE TABLE ... SELECT Statements

This section discusses how MySQL replicates CREATE TABLE ... SELECT statements.

MySQL 5.7 does not allow a CREATE TABLE ... SELECT statement to make any changes in tables other than the table that is created by the statement. Some older versions of MySQL which permitted these statements to do so; this means that, when using statement-based replication between a MySQL 5.6 or later slave and a master running a previous version of MySQL, a CREATE TABLE ... SELECT statement causing changes in other tables on the master fails on the slave, causing replication to stop. To keep this from happening, you should use row-based replication, rewrite the offending statement before running it on the master, or upgrade the master to MySQL 5.7. (If you choose to upgrade the master, keep in mind that such a CREATE TABLE ... SELECT statement will fail following the upgrade unless it is rewritten to remove any side effects on other tables.) This is not an issue when using row-based replication, because the statement is logged as a CREATE TABLE statement with any changes to table data logged as row-insert events, rather than as the entire CREATE TABLE ... SELECT.

These behaviors are not dependent on MySQL version:

- CREATE TABLE ... SELECT always performs an implicit commit (Section 13.3.3, “Statements That Cause an Implicit Commit”).
- If destination table does not exist, logging occurs as follows. It does not matter whether IF NOT EXISTS is present.
  - STATEMENT or MIXED format: The statement is logged as written.
  - ROW format: The statement is logged as a CREATE TABLE statement followed by a series of insert-row events.
- If the statement fails, nothing is logged. This includes the case that the destination table exists and IF NOT EXISTS is not given.

When the destination table exists and IF NOT EXISTS is given, MySQL 5.7 ignores the statement completely; nothing is inserted or logged. The handling of such statements in this regard has changed considerably in previous MySQL releases; if you are replicating from a MySQL 5.5.6 or older master to a newer slave, see Replication of CREATE ... IF NOT EXISTS Statements, for more information.

16.4.1.6 Replication of CREATE SERVER, ALTER SERVER, and DROP SERVER

In MySQL 5.7, the statements CREATE SERVER, ALTER SERVER, and DROP SERVER are not written to the binary log, regardless of the binary logging format that is in use.

16.4.1.7 Replication of CURRENT_USER()

The following statements support use of the CURRENT_USER() function to take the place of the name of (and, possibly, the host for) an affected user or a definer; in such cases, CURRENT_USER() is expanded where and as needed:

- DROP USER
- RENAME USER
- GRANT
- REVOKE

- CREATE FUNCTION
- CREATE PROCEDURE
- CREATE TRIGGER
- CREATE EVENT
- CREATE VIEW
- ALTER EVENT
- ALTER VIEW
- SET PASSWORD

When CURRENT_USER() or CURRENT_USER is used as the definer in any of the statements CREATE FUNCTION, CREATE PROCEDURE, CREATE TRIGGER, CREATE EVENT, CREATE VIEW, or ALTER VIEW when binary logging is enabled, the function reference is expanded before it is written to the binary log, so that the statement refers to the same user on both the master and the slave when the statement is replicated. CURRENT_USER() or CURRENT_USER is also expanded prior to being written to the binary log when used in DROP USER, RENAME USER, GRANT, REVOKE, or ALTER EVENT.

16.4.1.8 Replication of DROP ... IF EXISTS Statements

The DROP DATABASE IF EXISTS, DROP TABLE IF EXISTS, and DROP VIEW IF EXISTS statements are always replicated, even if the database, table, or view to be dropped does not exist on the master. This is to ensure that the object to be dropped no longer exists on either the master or the slave, once the slave has caught up with the master.

DROP ... IF EXISTS statements for stored programs (stored procedures and functions, triggers, and events) are also replicated, even if the stored program to be dropped does not exist on the master.

16.4.1.9 Replication with Differing Table Definitions on Master and Slave

Source and target tables for replication do not have to be identical. A table on the master can have more or fewer columns than the slave's copy of the table. In addition, corresponding table columns on the master and the slave can use different data types, subject to certain conditions.

In all cases where the source and target tables do not have identical definitions, the database and table names must be the same on both the master and the slave. Additional conditions are discussed, with examples, in the following two sections.

Replication with More Columns on Master or Slave

You can replicate a table from the master to the slave such that the master and slave copies of the table have differing numbers of columns, subject to the following conditions:

- Columns common to both versions of the table must be defined in the same order on the master and the slave.

  (This is true even if both tables have the same number of columns.)

- Columns common to both versions of the table must be defined before any additional columns.

  This means that executing an ALTER TABLE statement on the slave where a new column is inserted into the table within the range of columns common to both tables causes replication to fail, as shown in the following example:

Suppose that a table t, existing on the master and the slave, is defined by the following CREATE TABLE statement:

```
CREATE TABLE t (
    c1 INT,
    c2 INT,
    c3 INT
);
```

Suppose that the ALTER TABLE statement shown here is executed on the slave:

```
ALTER TABLE t ADD COLUMN cnew1 INT AFTER c3;
```

The previous ALTER TABLE is permitted on the slave because the columns c1, c2, and c3 that are common to both versions of table t remain grouped together in both versions of the table, before any columns that differ.

However, the following ALTER TABLE statement cannot be executed on the slave without causing replication to break:

```
ALTER TABLE t ADD COLUMN cnew2 INT AFTER c2;
```

Replication fails after execution on the slave of the ALTER TABLE statement just shown, because the new column cnew2 comes between columns common to both versions of t.

- Each “extra” column in the version of the table having more columns must have a default value.

  A column's default value is determined by a number of factors, including its type, whether it is defined with a DEFAULT option, whether it is declared as NULL, and the server SQL mode in effect at the time of its creation; for more information, see Section 11.5, “Data Type Default Values”).

In addition, when the slave's copy of the table has more columns than the master's copy, each column common to the tables must use the same data type in both tables.

Examples. The following examples illustrate some valid and invalid table definitions:

More columns on the master. The following table definitions are valid and replicate correctly:

```
master> CREATE TABLE t1 (c1 INT, c2 INT, c3 INT);
slave>  CREATE TABLE t1 (c1 INT, c2 INT);
```

The following table definitions would raise Error 1532 (ER_BINLOG_ROW_RBR_TO_SBR) because the definitions of the columns common to both versions of the table are in a different order on the slave than they are on the master:

```
master> CREATE TABLE t1 (c1 INT, c2 INT, c3 INT);
slave>  CREATE TABLE t1 (c2 INT, c1 INT);
```

The following table definitions would also raise Error 1532 because the definition of the extra column on the master appears before the definitions of the columns common to both versions of the table:

```
master> CREATE TABLE t1 (c3 INT, c1 INT, c2 INT);
slave>  CREATE TABLE t1 (c1 INT, c2 INT);
```

More columns on the slave. The following table definitions are valid and replicate correctly:

```
master> CREATE TABLE t1 (c1 INT, c2 INT);
slave>  CREATE TABLE t1 (c1 INT, c2 INT, c3 INT);
```

The following definitions raise Error 1532 because the columns common to both versions of the table are not defined in the same order on both the master and the slave:

```
master> CREATE TABLE t1 (c1 INT, c2 INT);
slave>  CREATE TABLE t1 (c2 INT, c1 INT, c3 INT);
```

The following table definitions also raise Error 1532 because the definition for the extra column in the slave's version of the table appears before the definitions for the columns which are common to both versions of the table:

```
master> CREATE TABLE t1 (c1 INT, c2 INT);
slave>  CREATE TABLE t1 (c3 INT, c1 INT, c2 INT);
```

The following table definitions fail because the slave's version of the table has additional columns compared to the master's version, and the two versions of the table use different data types for the common column `c2`:

```
master> CREATE TABLE t1 (c1 INT, c2 BIGINT);
slave>  CREATE TABLE t1 (c1 INT, c2 INT, c3 INT);
```

Replication of Columns Having Different Data Types

Corresponding columns on the master's and the slave's copies of the same table ideally should have the same data type. However, beginning with MySQL 5.1.21, this is not always strictly enforced, as long as certain conditions are met.

All other things being equal, it is always possible to replicate from a column of a given data type to another column of the same type and same size or width, where applicable, or larger. For example, you can replicate from a `CHAR(10)` column to another `CHAR(10)`, or from a `CHAR(10)` column to a `CHAR(25)` column without any problems. In certain cases, it also possible to replicate from a column having one data type (on the master) to a column having a different data type (on the slave); when the data type of the master's version of the column is promoted to a type that is the same size or larger on the slave, this is known as *attribute promotion*.

Attribute promotion can be used with both statement-based and row-based replication, and is not dependent on the storage engine used by either the master or the slave. However, the choice of logging format does have an effect on the type conversions that are permitted; the particulars are discussed later in this section.

> **Important**
>
> Whether you use statement-based or row-based replication, the slave's copy of the table cannot contain more columns than the master's copy if you wish to employ attribute promotion.

Statement-based replication. When using statement-based replication, a simple rule of thumb to follow is, “If the statement run on the master would also execute successfully on the slave, it should also replicate successfully”. In other words, if the statement uses a value that is compatible with the type of a given column on the slave, the statement can be replicated. For example, you can insert any value that fits in a TINYINT column into a BIGINT column as well; it follows that, even if you change the type of a TINYINT column in the slave's copy of a table to BIGINT, any insert into that column on the master that succeeds should also succeed on the slave, since it is impossible to have a legal TINYINT value that is large enough to exceed a BIGINT column.

Prior to MySQL 5.7.1, when using statement-based replication, AUTO_INCREMENT columns were required to be the same on both the master and the slave; otherwise, updates could be applied to the wrong table on the slave. (Bug #12669186)

Row-based replication: attribute promotion and demotion. Row-based replication in MySQL 5.7 supports attribute promotion and demotion between smaller data types and larger types. It is also possible to specify whether or not to permit lossy (truncated) or non-lossy conversions of demoted column values, as explained later in this section.

Lossy and non-lossy conversions. In the event that the target type cannot represent the value being inserted, a decision must be made on how to handle the conversion. If we permit the conversion but truncate (or otherwise modify) the source value to achieve a “fit” in the target column, we make what is known as a *lossy conversion*. A conversion which does not require truncation or similar modifications to fit the source column value in the target column is a *non-lossy* conversion.

Type conversion modes (slave_type_conversions variable). The setting of the slave_type_conversions global server variable controls the type conversion mode used on the slave. This variable takes a set of values from the following table, which shows the effects of each mode on the slave's type-conversion behavior:

Mode	Effect
ALL_LOSSY	In this mode, type conversions that would mean loss of information are permitted. This does not imply that non-lossy conversions are permitted, merely that only cases requiring either lossy conversions or no conversion at all are permitted; for example, enabling *only* this mode permits an INT column to be converted to TINYINT (a lossy conversion), but not a TINYINT column to an INT column (non-lossy). Attempting the latter conversion in this case would cause replication to stop with an error on the slave.
ALL_NON_LOSSY	This mode permits conversions that do not require truncation or other special handling of the source value; that is, it permits conversions where the target type has a wider range than the source type. Setting this mode has no bearing on whether lossy conversions are permitted; this is controlled with the ALL_LOSSY mode. If only ALL_NON_LOSSY is set, but not ALL_LOSSY, then attempting a conversion that would result in the loss of data (such as INT to TINYINT, or CHAR(25) to VARCHAR(20)) causes the slave to stop with an error.
ALL_LOSSY,ALL_NON_LOSSY	When this mode is set, all supported type conversions are permitted, whether or not they are lossy conversions.
ALL_SIGNED	Treat promoted integer types as signed values (the default behavior).
ALL_UNSIGNED	Treat promoted integer types as unsigned values.

Mode	Effect
ALL_SIGNED,ALL_UNSIGNED	Treat promoted integer types as signed if possible, otherwise as unsigned.
[*empty*]	When slave_type_conversions is not set, no attribute promotion or demotion is permitted; this means that all columns in the source and target tables must be of the same types. This mode is the default.

When an integer type is promoted, its signedness is not preserved. By default, the slave treats all such values as signed. Beginning with MySQL 5.7.2, you can control this behavior using ALL_SIGNED, ALL_UNSIGNED, or both. (Bug#15831300) ALL_SIGNED tells the slave to treat all promoted integer types as signed; ALL_UNSIGNED instructs it to treat these as unsigned. Specifying both causes the slave to treat the value as signed if possible, otherwise to treat it as unsigned; the order in which they are listed is not significant. Neither ALL_SIGNED nor ALL_UNSIGNED has any effect if at least one of ALL_LOSSY or ALL_NONLOSSY is not also used.

Changing the type conversion mode requires restarting the slave with the new slave_type_conversions setting.

Supported conversions. Supported conversions between different but similar data types are shown in the following list:

- Between any of the integer types TINYINT, SMALLINT, MEDIUMINT, INT, and BIGINT.

  This includes conversions between the signed and unsigned versions of these types.

  Lossy conversions are made by truncating the source value to the maximum (or minimum) permitted by the target column. For insuring non-lossy conversions when going from unsigned to signed types, the target column must be large enough to accommodate the range of values in the source column. For example, you can demote TINYINT UNSIGNED non-lossily to SMALLINT, but not to TINYINT.

- Between any of the decimal types DECIMAL, FLOAT, DOUBLE, and NUMERIC.

  FLOAT to DOUBLE is a non-lossy conversion; DOUBLE to FLOAT can only be handled lossily. A conversion from DECIMAL(*M*,*D*) to DECIMAL(*M'*,*D'*) where *M'* => *M* and *D'* => *D* is non-lossy; for any case where *M'* < *M*, *D'* < *D*, or both, only a lossy conversion can be made.

  For any of the decimal types, if a value to be stored cannot be fit in the target type, the value is rounded down according to the rounding rules defined for the server elsewhere in the documentation. See Section 12.19.4, “Rounding Behavior”, for information about how this is done for decimal types.

- Between any of the string types CHAR, VARCHAR, and TEXT, including conversions between different widths.

  Conversion of a CHAR, VARCHAR, or TEXT to a CHAR, VARCHAR, or TEXT column the same size or larger is never lossy. Lossy conversion is handled by inserting only the first *N* characters of the string on the slave, where *N* is the width of the target column.

  Important

  Replication between columns using different character sets is not supported.

- Between any of the binary data types BINARY, VARBINARY, and BLOB, including conversions between different widths.

Conversion of a BINARY, VARBINARY, or BLOB to a BINARY, VARBINARY, or BLOB column the same size or larger is never lossy. Lossy conversion is handled by inserting only the first *N* bytes of the string on the slave, where *N* is the width of the target column.

- Between any 2 BIT columns of any 2 sizes.

  When inserting a value from a BIT(*M*) column into a BIT(*M'*) column, where *M'* > *M*, the most significant bits of the BIT(*M'*) columns are cleared (set to zero) and the *M* bits of the BIT(*M*) value are set as the least significant bits of the BIT(*M'*) column.

  When inserting a value from a source BIT(*M*) column into a target BIT(*M'*) column, where *M'* < *M*, the maximum possible value for the BIT(*M'*) column is assigned; in other words, an “all-set” value is assigned to the target column.

Conversions between types not in the previous list are not permitted.

Replication type conversions in MySQL 5.5.3 and earlier. Prior to MySQL 5.5.3, with row-based binary logging, you could not replicate between different INT subtypes, such as from TINYINT to BIGINT, because changes to columns of these types were represented differently from one another in the binary log when using row-based logging. (However, you could replicate from BLOB to TEXT using row-based replication because changes to BLOB and TEXT columns were represented using the same format in the binary log.)

Supported conversions for attribute promotion when using row-based replication prior to MySQL 5.5.3 are shown in the following table:

From (Master)	To (Slave)
BINARY	CHAR
BLOB	TEXT
CHAR	BINARY
DECIMAL	NUMERIC
NUMERIC	DECIMAL
TEXT	BLOB
VARBINARY	VARCHAR
VARCHAR	VARBINARY

Note

In all cases, the size or width of the column on the slave must be equal to or greater than that of the column on the master. For example, you could replicate from a CHAR(10) column on the master to a column that used BINARY(10) or BINARY(25) on the slave, but you could not replicate from a CHAR(10) column on the master to BINARY(5) column on the slave.

Any unique index (including primary keys) having a prefix must use a prefix of the same length on both master and slave; in such cases, differing prefix lengths are disallowed. It is possible to use a nonunique index whose prefix length differs between master and slave, but this can cause serious performance issues, particularly when the prefix used on the master is longer. This is due to the fact that 2 unique prefixes of a given length may no longer be unique at a shorter length; for example, the words *catalogue* and *catamount* have the 5-character prefixes *catal* and *catam*, respectively, but share the same 4-character prefix (*cata*). This can lead to queries that use such indexes executing less efficiently on the slave, when a shorter prefix is employed in the slave' definition of the same index than on the master.

For `DECIMAL` and `NUMERIC` columns, both the mantissa (*M*) and the number of decimals (*D*) must be the same size or larger on the slave as compared with the master. For example, replication from a `NUMERIC(5,4)` to a `DECIMAL(6,4)` worked, but not from a `NUMERIC(5,4)` to a `DECIMAL(5,3)`.

Prior to MySQL 5.5.3, MySQL replication did not support attribute promotion of any of the following data types to or from any other data type when using row-based replication:

- `INT` (including `TINYINT`, `SMALLINT`, `MEDIUMINT`, `BIGINT`).

  Promotion between `INT` subtypes—for example, from `SMALLINT` to `BIGINT`—was also not supported prior to MySQL 5.5.3.

- `SET` or `ENUM`.

- `FLOAT` or `DOUBLE`.

- All of the data types relating to dates, times, or both: `DATE`, `TIME`, `DATETIME`, `TIMESTAMP`, and `YEAR`.

16.4.1.10 Replication and `DIRECTORY` Table Options

If a `DATA DIRECTORY` or `INDEX DIRECTORY` table option is used in a `CREATE TABLE` statement on the master server, the table option is also used on the slave. This can cause problems if no corresponding directory exists in the slave host file system or if it exists but is not accessible to the slave server. This can be overridden by using the `NO_DIR_IN_CREATE` server SQL mode on the slave, which causes the slave to ignore the `DATA DIRECTORY` and `INDEX DIRECTORY` table options when replicating `CREATE TABLE` statements. The result is that `MyISAM` data and index files are created in the table's database directory.

For more information, see Section 5.1.7, “Server SQL Modes”.

16.4.1.11 Replication of Invoked Features

Replication of invoked features such as user-defined functions (UDFs) and stored programs (stored procedures and functions, triggers, and events) provides the following characteristics:

- The effects of the feature are always replicated.

- The following statements are replicated using statement-based replication:

  - `CREATE EVENT`
  - `ALTER EVENT`
  - `DROP EVENT`
  - `CREATE PROCEDURE`
  - `DROP PROCEDURE`
  - `CREATE FUNCTION`
  - `DROP FUNCTION`

  - CREATE TRIGGER
  - DROP TRIGGER

  However, the *effects* of features created, modified, or dropped using these statements are replicated using row-based replication.

  > **Note**
  >
  > Attempting to replicate invoked features using statement-based replication produces the warning Statement is not safe to log in statement format. For example, trying to replicate a UDF with statement-based replication generates this warning because it currently cannot be determined by the MySQL server whether the UDF is deterministic. If you are absolutely certain that the invoked feature's effects are deterministic, you can safely disregard such warnings.

- In the case of CREATE EVENT and ALTER EVENT:

  - The status of the event is set to SLAVESIDE_DISABLED on the slave regardless of the state specified (this does not apply to DROP EVENT).

  - The master on which the event was created is identified on the slave by its server ID. The ORIGINATOR column in INFORMATION_SCHEMA.EVENTS and the originator column in mysql.event store this information. See Section 19.7, "The INFORMATION_SCHEMA EVENTS Table", and Section 13.7.5.17, "SHOW EVENTS Syntax", for more information.

- The feature implementation resides on the slave in a renewable state so that if the master fails, the slave can be used as the master without loss of event processing.

To determine whether there are any scheduled events on a MySQL server that were created on a different server (that was acting as a replication master), query the INFORMATION_SCHEMA.EVENTS table in a manner similar to what is shown here:

```
SELECT EVENT_SCHEMA, EVENT_NAME
    FROM INFORMATION_SCHEMA.EVENTS
    WHERE STATUS = 'SLAVESIDE_DISABLED';
```

Alternatively, you can use the SHOW EVENTS statement, like this:

```
SHOW EVENTS
    WHERE STATUS = 'SLAVESIDE_DISABLED';
```

When promoting a replication slave having such events to a replication master, you must enable each event using ALTER EVENT *event_name* ENABLED, where *event_name* is the name of the event.

If more than one master was involved in creating events on this slave, and you wish to identify events that were created only on a given master having the server ID *master_id*, modify the previous query on the EVENTS table to include the ORIGINATOR column, as shown here:

```
SELECT EVENT_SCHEMA, EVENT_NAME, ORIGINATOR
    FROM INFORMATION_SCHEMA.EVENTS
    WHERE STATUS = 'SLAVESIDE_DISABLED'
    AND   ORIGINATOR = 'master_id'
```

You can employ ORIGINATOR with the SHOW EVENTS statement in a similar fashion:

```
SHOW EVENTS
    WHERE STATUS = 'SLAVESIDE_DISABLED'
    AND   ORIGINATOR = 'master_id'
```

Before enabling events that were replicated from the master, you should disable the MySQL Event Scheduler on the slave (using a statement such as `SET GLOBAL event_scheduler = OFF;`), run any necessary `ALTER EVENT` statements, restart the server, then re-enable the Event Scheduler on the slave afterward (using a statement such as `SET GLOBAL event_scheduler = ON;`)-

If you later demote the new master back to being a replication slave, you must disable manually all events enabled by the `ALTER EVENT` statements. You can do this by storing in a separate table the event names from the `SELECT` statement shown previously, or using `ALTER EVENT` statements to rename the events with a common prefix such as `replicated_` to identify them.

If you rename the events, then when demoting this server back to being a replication slave, you can identify the events by querying the `EVENTS` table, as shown here:

```
SELECT CONCAT(EVENT_SCHEMA, '.', EVENT_NAME) AS 'Db.Event'
      FROM INFORMATION_SCHEMA.EVENTS
      WHERE INSTR(EVENT_NAME, 'replicated_') = 1;
```

16.4.1.12 Replication and Floating-Point Values

With statement-based replication, values are converted from decimal to binary. Because conversions between decimal and binary representations of them may be approximate, comparisons involving floating-point values are inexact. This is true for operations that use floating-point values explicitly, or that use values that are converted to floating-point implicitly. Comparisons of floating-point values might yield different results on master and slave servers due to differences in computer architecture, the compiler used to build MySQL, and so forth. See Section 12.2, “Type Conversion in Expression Evaluation”, and Section C.5.5.8, “Problems with Floating-Point Values”.

16.4.1.13 Replication and Fractional Seconds Support

MySQL 5.7 permits fractional seconds for `TIME`, `DATETIME`, and `TIMESTAMP` values, with up to microseconds (6 digits) precision. See Section 11.3.6, “Fractional Seconds in Time Values”.

There may be problems replicating from a master server that understands fractional seconds to an older slave (MySQL 5.6.3 and earlier) that does not:

- For `CREATE TABLE` statements containing columns that have an *`fsp`* (fractional seconds precision) value greater than 0, replication will fail due to parser errors.

- Statements that use temporal data types with an *`fsp`* value of 0 will work for with statement-based logging but not row-based logging. In the latter case, the data types have binary formats and type codes on the master that differ from those on the slave.

- Some expression results will differ on master and slave. Examples: On the master, the `timestamp` system variable returns a value that includes a microseconds fractional part; on the slave, it returns an integer. On the master, functions that return a result that includes the current time (such as `CURTIME()`, `SYSDATE()`, or `UTC_TIMESTAMP()`) interpret an argument as an *`fsp`* value and the return value includes a fractional seconds part of that many digits. On the slave, these functions permit an argument but ignore it.

16.4.1.14 Replication and `FLUSH`

Some forms of the FLUSH statement are not logged because they could cause problems if replicated to a slave: FLUSH LOGS, FLUSH MASTER, FLUSH SLAVE, and FLUSH TABLES WITH READ LOCK. For a syntax example, see Section 13.7.6.3, "FLUSH Syntax". The FLUSH TABLES, ANALYZE TABLE, OPTIMIZE TABLE, and REPAIR TABLE statements are written to the binary log and thus replicated to slaves. This is not normally a problem because these statements do not modify table data.

However, this behavior can cause difficulties under certain circumstances. If you replicate the privilege tables in the mysql database and update those tables directly without using GRANT, you must issue a FLUSH PRIVILEGES on the slaves to put the new privileges into effect. In addition, if you use FLUSH TABLES when renaming a MyISAM table that is part of a MERGE table, you must issue FLUSH TABLES manually on the slaves. These statements are written to the binary log unless you specify NO_WRITE_TO_BINLOG or its alias LOCAL.

16.4.1.15 Replication and System Functions

Certain functions do not replicate well under some conditions:

- The USER(), CURRENT_USER() (or CURRENT_USER), UUID(), VERSION(), and LOAD_FILE() functions are replicated without change and thus do not work reliably on the slave unless row-based replication is enabled. (See Section 16.1.2, "Replication Formats".)

  USER() and CURRENT_USER() are automatically replicated using row-based replication when using MIXED mode, and generate a warning in STATEMENT mode. (See also Section 16.4.1.7, "Replication of CURRENT_USER()".) This is also true for VERSION() and RAND().

- For NOW(), the binary log includes the timestamp. This means that the value *as returned by the call to this function on the master* is replicated to the slave. This can lead to a possibly unexpected result when replicating between MySQL servers in different time zones. Suppose that the master is located in New York, the slave is located in Stockholm, and both servers are using local time. Suppose further that, on the master, you create a table mytable, perform an INSERT statement on this table, and then select from the table, as shown here:

  ```
  mysql> CREATE TABLE mytable (mycol TEXT);
  Query OK, 0 rows affected (0.06 sec)

  mysql> INSERT INTO mytable VALUES ( NOW() );
  Query OK, 1 row affected (0.00 sec)

  mysql> SELECT * FROM mytable;
  +---------------------+
  | mycol               |
  +---------------------+
  | 2009-09-01 12:00:00 |
  +---------------------+
  1 row in set (0.00 sec)
  ```

  Local time in Stockholm is 6 hours later than in New York; so, if you issue SELECT NOW() on the slave at that exact same instant, the value 2009-09-01 18:00:00 is returned. For this reason, if you select from the slave's copy of mytable after the CREATE TABLE and INSERT statements just shown have been replicated, you might expect mycol to contain the value 2009-09-01 18:00:00. However, this is not the case; when you select from the slave's copy of mytable, you obtain exactly the same result as on the master:

  ```
  mysql> SELECT * FROM mytable;
  +---------------------+
  | mycol               |
  ```

```
+---------------------+
| 2009-09-01 12:00:00 |
+---------------------+
1 row in set (0.00 sec)
```

Unlike NOW(), the SYSDATE() function is not replication-safe because it is not affected by SET TIMESTAMP statements in the binary log and is nondeterministic if statement-based logging is used. This is not a problem if row-based logging is used.

An alternative is to use the --sysdate-is-now option to cause SYSDATE() to be an alias for NOW(). This must be done on the master and the slave to work correctly. In such cases, a warning is still issued by this function, but can safely be ignored as long as --sysdate-is-now is used on both the master and the slave.

Beginning with MySQL 5.5.1, SYSDATE() is automatically replicated using row-based replication when using MIXED mode, and generates a warning in STATEMENT mode. (Bug #47995)

See also Section 16.4.1.30, "Replication and Time Zones".

- *The following restriction applies to statement-based replication only, not to row-based replication.* The GET_LOCK(), RELEASE_LOCK(), IS_FREE_LOCK(), and IS_USED_LOCK() functions that handle user-level locks are replicated without the slave knowing the concurrency context on the master. Therefore, these functions should not be used to insert into a master table because the content on the slave would differ. For example, do not issue a statement such as INSERT INTO mytable VALUES(GET_LOCK(...)).

  Beginning with MySQL 5.5.1, these functions are automatically replicated using row-based replication when using MIXED mode, and generate a warning in STATEMENT mode. (Bug #47995)

As a workaround for the preceding limitations when statement-based replication is in effect, you can use the strategy of saving the problematic function result in a user variable and referring to the variable in a later statement. For example, the following single-row INSERT is problematic due to the reference to the UUID() function:

```
INSERT INTO t VALUES(UUID());
```

To work around the problem, do this instead:

```
SET @my_uuid = UUID();
INSERT INTO t VALUES(@my_uuid);
```

That sequence of statements replicates because the value of @my_uuid is stored in the binary log as a user-variable event prior to the INSERT statement and is available for use in the INSERT.

The same idea applies to multiple-row inserts, but is more cumbersome to use. For a two-row insert, you can do this:

```
SET @my_uuid1 = UUID(); @my_uuid2 = UUID();
INSERT INTO t VALUES(@my_uuid1),(@my_uuid2);
```

However, if the number of rows is large or unknown, the workaround is difficult or impracticable. For example, you cannot convert the following statement to one in which a given individual user variable is associated with each row:

```
INSERT INTO t2 SELECT UUID(), * FROM t1;
```

Within a stored function, RAND() replicates correctly as long as it is invoked only once during the execution of the function. (You can consider the function execution timestamp and random number seed as implicit inputs that are identical on the master and slave.)

The FOUND_ROWS() and ROW_COUNT() functions are not replicated reliably using statement-based replication. A workaround is to store the result of the function call in a user variable, and then use that in the INSERT statement. For example, if you wish to store the result in a table named mytable, you might normally do so like this:

```
SELECT SQL_CALC_FOUND_ROWS FROM mytable LIMIT 1;
INSERT INTO mytable VALUES( FOUND_ROWS() );
```

However, if you are replicating mytable, you should use SELECT ... INTO, and then store the variable in the table, like this:

```
SELECT SQL_CALC_FOUND_ROWS INTO @found_rows FROM mytable LIMIT 1;
INSERT INTO mytable VALUES(@found_rows);
```

In this way, the user variable is replicated as part of the context, and applied on the slave correctly.

These functions are automatically replicated using row-based replication when using MIXED mode, and generate a warning in STATEMENT mode. (Bug #12092, Bug #30244)

Prior to MySQL 5.7.3, the value of LAST_INSERT_ID() was not replicated correctly if any filtering options such as --replicate-ignore-db and --replicate-do-table were enabled on the slave. (Bug #17234370, BUG# 69861)

16.4.1.16 Replication and LIMIT

Statement-based replication of LIMIT clauses in DELETE, UPDATE, and INSERT ... SELECT statements is unsafe since the order of the rows affected is not defined. (Such statements can be replicated correctly with statement-based replication only if they also contain an ORDER BY clause.) When such a statement is encountered:

- When using STATEMENT mode, a warning that the statement is not safe for statement-based replication is now issued.

  Currently, when using STATEMENT mode, warnings are issued for DML statements containing LIMIT even when they also have an ORDER BY clause (and so are made deterministic). This is a known issue. (Bug #42851)

- When using MIXED mode, the statement is now automatically replicated using row-based mode.

16.4.1.17 Replication and LOAD DATA INFILE

The LOAD DATA INFILE statement was not always replicated correctly to a slave running MySQL 5.5.0 or earlier from a master running MySQL 4.0 or earlier. When using statement-based replication, the LOAD DATA INFILE statement CONCURRENT option was not replicated. This issue was fixed in MySQL 5.5.0. This issue does not have any impact on CONCURRENT option handling when using row-based replication in MySQL 5.1 or later. (Bug #34628)

In MySQL 5.7, LOAD DATA INFILE is considered unsafe (see Section 16.1.2.3, “Determination of Safe and Unsafe Statements in Binary Logging”). It causes a warning when using statement-based logging format, and is logged using row-based format when using mixed-format logging.

16.4.1.18 Replication and REPAIR TABLE

When used on a corrupted or otherwise damaged table, it is possible for the REPAIR TABLE statement to delete rows that cannot be recovered. However, any such modifications of table data performed by this statement are not replicated, which can cause master and slave to lose synchronization. For this reason, in the event that a table on the master becomes damaged and you use REPAIR TABLE to repair it, you should first stop replication (if it is still running) before using REPAIR TABLE, then afterward compare the master's and slave's copies of the table and be prepared to correct any discrepancies manually, before restarting replication.

16.4.1.19 Replication and Master or Slave Shutdowns

It is safe to shut down a master server and restart it later. When a slave loses its connection to the master, the slave tries to reconnect immediately and retries periodically if that fails. The default is to retry every 60 seconds. This may be changed with the CHANGE MASTER TO statement. A slave also is able to deal with network connectivity outages. However, the slave notices the network outage only after receiving no data from the master for slave_net_timeout seconds. If your outages are short, you may want to decrease slave_net_timeout. See Section 5.1.4, “Server System Variables”.

An unclean shutdown (for example, a crash) on the master side can result in the master binary log having a final position less than the most recent position read by the slave, due to the master binary log file not being flushed. This can cause the slave not to be able to replicate when the master comes back up. Setting sync_binlog=1 in the master my.cnf file helps to minimize this problem because it causes the master to flush its binary log more frequently.

Shutting down a slave cleanly is safe because it keeps track of where it left off. However, be careful that the slave does not have temporary tables open; see Section 16.4.1.22, “Replication and Temporary Tables”. Unclean shutdowns might produce problems, especially if the disk cache was not flushed to disk before the problem occurred:

- For transactions, the slave commits and then updates relay-log.info. If a crash occurs between these two operations, relay log processing will have proceeded further than the information file indicates and the slave will re-execute the events from the last transaction in the relay log after it has been restarted.

- A similar problem can occur if the slave updates relay-log.info but the server host crashes before the write has been flushed to disk. To minimize the chance of this occurring, set sync_relay_log_info=1 in the slave my.cnf file. The default value of sync_relay_log_info is 0, which does not cause writes to be forced to disk; the server relies on the operating system to flush the file from time to time.

The fault tolerance of your system for these types of problems is greatly increased if you have a good uninterruptible power supply.

16.4.1.20 Replication and max_allowed_packet

max_allowed_packet sets an upper limit on the size of any single message between the MySQL server and clients, including replication slaves. If you are replicating large column values (such as might be found in TEXT or BLOB columns) and max_allowed_packet is too small on the master, the master fails with an error, and the slave shuts down the I/O thread. If max_allowed_packet is too small on the slave, this also causes the slave to stop the I/O thread.

Row-based replication currently sends all columns and column values for updated rows from the master to the slave, including values of columns that were not actually changed by the update. This means that, when you are replicating large column values using row-based replication, you must take care to set max_allowed_packet large enough to accommodate the largest row in any table to be replicated, even if you are replicating updates only, or you are inserting only relatively small values.

16.4.1.21 Replication and MEMORY Tables

When a master server shuts down and restarts, its MEMORY tables become empty. To replicate this effect to slaves, the first time that the master uses a given MEMORY table after startup, it logs an event that notifies slaves that the table must to be emptied by writing a DELETE statement for that table to the binary log.

When a slave server shuts down and restarts, its MEMORY tables become empty. This causes the slave to be out of synchrony with the master and may lead to other failures or cause the slave to stop:

- Row-format updates and deletes received from the master may fail with Can't find record in '*memory_table*'.
- Statements such as INSERT INTO ... SELECT FROM *memory_table* may insert a different set of rows on the master and slave.

The safe way to restart a slave that is replicating MEMORY tables is to first drop or delete all rows from the MEMORY tables on the master and wait until those changes have replicated to the slave. Then it is safe to restart the slave.

An alternative restart method may apply in some cases. When binlog_format=ROW, you can prevent the slave from stopping if you set slave_exec_mode=IDEMPOTENT before you start the slave again. This allows the slave to continue to replicate, but its MEMORY tables will still be different from those on the master. This can be okay if the application logic is such that the contents of MEMORY tables can be safely lost (for example, if the MEMORY tables are used for caching). slave_exec_mode=IDEMPOTENT applies globally to all tables, so it may hide other replication errors in non-MEMORY tables.

The size of MEMORY tables is limited by the value of the max_heap_table_size system variable, which is not replicated (see Section 16.4.1.34, "Replication and Variables"). A change in max_heap_table_size takes effect for MEMORY tables that are created or updated using ALTER TABLE ... ENGINE = MEMORY or TRUNCATE TABLE following the change, or for all MEMORY tables following a server restart. If you increase the value of this variable on the master without doing so on the slave, it becomes possible for a table on the master to grow larger than its counterpart on the slave, leading to inserts that succeed on the master but fail on the slave with Table is full errors. This is a known issue (Bug #48666). In such cases, you must set the global value of max_heap_table_size on the slave as well as on the master, then restart replication. It is also recommended that you restart both the master and slave MySQL servers, to insure that the new value takes complete (global) effect on each of them.

See Section 14.4, "The MEMORY Storage Engine", for more information about MEMORY tables.

16.4.1.22 Replication and Temporary Tables

The discussion in the following paragraphs does not apply when binlog_format=ROW because, in that case, temporary tables are not replicated; this means that there are never any temporary tables on the slave to be lost in the event of an unplanned shutdown by the slave. The remainder of this section applies only when using statement-based or mixed-format replication. Loss of replicated temporary tables on the slave can be an issue, whenever binlog_format is STATEMENT or MIXED, for statements involving temporary tables that can be logged safely using statement-based format. For more information about row-based replication and temporary tables, see RBL, RBR, and temporary tables.

Safe slave shutdown when using temporary tables. Temporary tables are replicated except in the case where you stop the slave server (not just the slave threads) and you have replicated temporary tables that are open for use in updates that have not yet been executed on the slave. If you stop the slave server, the temporary tables needed by those updates are no longer available when the slave is restarted. To avoid this problem, do not shut down the slave while it has temporary tables open. Instead, use the following procedure:

1. Issue a `STOP SLAVE SQL_THREAD` statement.
2. Use `SHOW STATUS` to check the value of the `Slave_open_temp_tables` variable.
3. If the value is not 0, restart the slave SQL thread with `START SLAVE SQL_THREAD` and repeat the procedure later.
4. When the value is 0, issue a `mysqladmin shutdown` command to stop the slave.

Temporary tables and replication options. By default, all temporary tables are replicated; this happens whether or not there are any matching `--replicate-do-db`, `--replicate-do-table`, or `--replicate-wild-do-table` options in effect. However, the `--replicate-ignore-table` and `--replicate-wild-ignore-table` options are honored for temporary tables.

A recommended practice when using statement-based or mixed-format replication is to designate a prefix for exclusive use in naming temporary tables that you do not want replicated, then employ a `--replicate-wild-ignore-table` option to match that prefix. For example, you might give all such tables names beginning with `norep` (such as `norepmytable`, `norepyourtable`, and so on), then use `--replicate-wild-ignore-table=norep%` to prevent them from being replicated.

16.4.1.23 Replication of the `mysql` System Database

Data modification statements made to tables in the `mysql` database are replicated according to the value of `binlog_format`; if this value is `MIXED`, these statements are replicated using row-based format. However, statements that would normally update this information indirectly—such `GRANT`, `REVOKE`, and statements manipulating triggers, stored routines, and views—are replicated to slaves using statement-based replication.

16.4.1.24 Replication and the Query Optimizer

It is possible for the data on the master and slave to become different if a statement is written in such a way that the data modification is nondeterministic; that is, left up the query optimizer. (In general, this is not a good practice, even outside of replication.) Examples of nondeterministic statements include `DELETE` or `UPDATE` statements that use `LIMIT` with no `ORDER BY` clause; see Section 16.4.1.16, “Replication and `LIMIT`”, for a detailed discussion of these.

16.4.1.25 Replication and Reserved Words

You can encounter problems when you attempt to replicate from an older master to a newer slave and you make use of identifiers on the master that are reserved words in the newer MySQL version running on the slave. An example of this is using a table column named `current_user` on a 4.0 master that is replicating to a 4.1 or higher slave because `CURRENT_USER` is a reserved word beginning in MySQL 4.1. Replication can fail in such cases with Error 1064 `You have an error in your SQL syntax...`, *even if a database or table named using the reserved word or a table having a column named using the reserved word is excluded from replication*. This is due to the fact that each SQL event must be parsed by the slave prior to execution, so that the slave knows which database object or objects would be affected; only after the event is parsed can the slave apply any filtering rules defined by `--replicate-do-db`, `--replicate-do-table`, `--replicate-ignore-db`, and `--replicate-ignore-table`.

To work around the problem of database, table, or column names on the master which would be regarded as reserved words by the slave, do one of the following:

- Use one or more ALTER TABLE statements on the master to change the names of any database objects where these names would be considered reserved words on the slave, and change any SQL statements that use the old names to use the new names instead.
- In any SQL statements using these database object names, write the names as quoted identifiers using backtick characters (`).

For listings of reserved words by MySQL version, see Reserved Words, in the *MySQL Server Version Reference*. For identifier quoting rules, see Section 9.2, “Schema Object Names”.

16.4.1.26 Slave Errors During Replication

If a statement produces the same error (identical error code) on both the master and the slave, the error is logged, but replication continues.

If a statement produces different errors on the master and the slave, the slave SQL thread terminates, and the slave writes a message to its error log and waits for the database administrator to decide what to do about the error. This includes the case that a statement produces an error on the master or the slave, but not both. To address the issue, connect to the slave manually and determine the cause of the problem. SHOW SLAVE STATUS is useful for this. Then fix the problem and run START SLAVE. For example, you might need to create a nonexistent table before you can start the slave again.

If this error code validation behavior is not desirable, some or all errors can be masked out (ignored) with the --slave-skip-errors option.

For nontransactional storage engines such as MyISAM, it is possible to have a statement that only partially updates a table and returns an error code. This can happen, for example, on a multiple-row insert that has one row violating a key constraint, or if a long update statement is killed after updating some of the rows. If that happens on the master, the slave expects execution of the statement to result in the same error code. If it does not, the slave SQL thread stops as described previously.

If you are replicating between tables that use different storage engines on the master and slave, keep in mind that the same statement might produce a different error when run against one version of the table, but not the other, or might cause an error for one version of the table, but not the other. For example, since MyISAM ignores foreign key constraints, an INSERT or UPDATE statement accessing an InnoDB table on the master might cause a foreign key violation but the same statement performed on a MyISAM version of the same table on the slave would produce no such error, causing replication to stop.

16.4.1.27 Replication of Server-Side Help Tables

The server maintains tables in the mysql database that store information for the HELP statement (see Section 13.8.3, “HELP Syntax”. These tables can be loaded manually as described at Section 5.1.10, “Server-Side Help”.

Help table content is derived from the MySQL Reference Manual. There are versions of the manual specific to each MySQL release series, so help content is specific to each series as well. Normally, you load a version of help content that matches the server version. This has implications for replication. For example, you would load MySQL 5.5 help content into a MySQL 5.5 master server, but not necessarily replicate that content to a MySQL 5.6 slave server for which 5.6 help content is more appropriate.

This section describes how to manage help table content upgrades when your servers participate in replication. Server versions are one factor in this task. Another is that help table structure may differ between the master and the slave.

Assume that help content is stored in a file named fill_help_tables.sql. In MySQL distributions, this file is located under the share or share/mysql directory, and the most recent version is always available for download from http://dev.mysql.com/doc/index-other.html.

To upgrade help tables, using the following procedure. Connection parameters are not shown for the `mysql` commands discussed here; in all cases, connect to the server using an account such as `root` that has privileges for modifying tables in the `mysql` database.

1. Upgrade your servers by running `mysql_upgrade`, first on the slaves and then on the master. This is the usual principle of upgrading slaves first.

2. Decide whether you want to replicate help table content from the master to its slaves. If not, load the content on the master and each slave individually. Otherwise, check for and resolve any incompatibilities between help table structure on the master and its slaves, then load the content into the master and let it replicate to the slaves.

   More detail about these two methods of loading help table content follows.

Loading Help Table Content Without Replication to Slaves

To load help table content without replication, run this command on the master and each slave individually, using a `fill_help_tables.sql` file containing content appropriate to the server version (enter the command on one line):

```
mysql --init-command="SET sql_log_bin=0"
  mysql < fill_help_tables.sql
```

Use the `--init-command` option on each server, including the slaves, in case a slave also acts as a master to other slaves in your replication topology. The `SET` statement suppresses binary logging. After the command has been run on each server to be upgraded, you are done.

> **Note**
>
> As of MySQL 5.7.5, the `fill_help_tables.sql` file includes the `SET` statement to cause the file contents not to replicate. Thus, for 5.7. and up, the command is simpler:
>
> ```
> mysql mysql < fill_help_tables.sql
> ```

Loading Help Table Content With Replication to Slaves

> **Note**
>
> As mentioned previously, `fill_help_tables.sql` in MySQL 5.7.5 and up includes a `SET` statement to suppress binary logging of the file contents. If you want to replicate help table contents for MySQL 5.7.5 or later, you must edit `fill_help_tables.sql` to remove the `SET` statement. This should rarely be desireable because help table contents are specific to the version of the server into which they are loaded, which may differ for master and slave.

If you do want to replicate help table content, check for help table incompatibilities between your master and its slaves. The `url` column in the `help_category` and `help_topic` tables was originally `CHAR(128)`, but is `TEXT` in newer MySQL versions to accommodate longer URLs. To check help table structure, use this statement:

```
SELECT TABLE_NAME, COLUMN_NAME, COLUMN_TYPE
FROM INFORMATION_SCHEMA.COLUMNS
WHERE TABLE_SCHEMA = 'mysql'
AND COLUMN_NAME = 'url';
```

For tables with the old structure, the statement produces this result:

```
+---------------+-------------+-------------+
| TABLE_NAME    | COLUMN_NAME | COLUMN_TYPE |
+---------------+-------------+-------------+
| help_category | url         | char(128)   |
| help_topic    | url         | char(128)   |
+---------------+-------------+-------------+
```

For tables with the new structure, the statement produces this result:

```
+---------------+-------------+-------------+
| TABLE_NAME    | COLUMN_NAME | COLUMN_TYPE |
+---------------+-------------+-------------+
| help_category | url         | text        |
| help_topic    | url         | text        |
+---------------+-------------+-------------+
```

If the master and slave both have the old structure or both have the new structure, they are compatible and you can replicate help table content by executing this command on the master:

```
mysql mysql < fill_help_tables.sql
```

The table content will load into the master, then replicate to the slaves.

If the master and slave have incompatible help tables (one server has the old structure and the other has the new), you have a choice between not replicating help table content after all, or making the table structures compatible so that you can replicate the content.

- If you decide not to replicate the content after all, upgrade the master and slaves individually using `mysql` with the `--init-command` option, as described previously.

- If instead you decide to make the table structures compatible, upgrade the tables on the server that has the old structure. Suppose that your master server has the old table structure. Upgrade its tables to the new structure manually by executing these statements (binary logging is disabled here to prevent replication of the changes to the slaves, which already have the new structure):

  ```
  SET sql_log_bin=0;
  ALTER TABLE mysql.help_category ALTER COLUMN url TEXT;
  ALTER TABLE mysql.help_topic ALTER COLUMN url TEXT;
  ```

  Then run this command on the master:

  ```
  mysql mysql < fill_help_tables.sql
  ```

  The table content will load into the master, then replicate to the slaves.

16.4.1.28 Replication and Server SQL Mode

Using different server SQL mode settings on the master and the slave may cause the same `INSERT` statements to be handled differently on the master and the slave, leading the master and slave to diverge. For best results, you should always use the same server SQL mode on the master and on the slave. This advice applies whether you are using statement-based or row-based replication.

If you are replicating partitioned tables, using different SQL modes on the master and the slave is likely to cause issues. At a minimum, this is likely to cause the distribution of data among partitions to be different in the master's and slave's copies of a given table. It may also cause inserts into partitioned tables that succeed on the master to fail on the slave.

For more information, see Section 5.1.7, “Server SQL Modes”.

As of MySQL 5.7.4, the deprecated ERROR_FOR_DIVISION_BY_ZERO, NO_ZERO_DATE, and NO_ZERO_IN_DATE SQL modes do nothing. Instead, their previous effects are included in the effects of strict SQL mode (STRICT_ALL_TABLES or STRICT_TRANS_TABLES). In other words, strict mode now means the same thing as the previous meaning of strict mode plus the ERROR_FOR_DIVISION_BY_ZERO, NO_ZERO_DATE, and NO_ZERO_IN_DATE modes. This change reduces the number of SQL modes with an effect dependent on strict mode and makes them part of strict mode itself.

To prepare for these SQL mode changes, it is advisable *before* upgrading to read SQL Mode Changes in MySQL 5.7. That discussion provides guidelines to assess whether your applications will be affected by these changes.

The deprecated ERROR_FOR_DIVISION_BY_ZERO, NO_ZERO_DATE, and NO_ZERO_IN_DATE SQL modes are still recognized so that statements that name them do not produce an error, but will be removed in a future version of MySQL. To make advance preparation for versions of MySQL in which these modes do not exist, applications should be modified to not refer to those mode names.

16.4.1.29 Replication Retries and Timeouts

The global system variable slave_transaction_retries affects replication as follows: If the slave SQL thread fails to execute a transaction because of an InnoDB deadlock or because it exceeded the InnoDB innodb_lock_wait_timeout value, or the NDBCLUSTER TransactionDeadlockDetectionTimeout or TransactionInactiveTimeout value, the slave automatically retries the transaction slave_transaction_retries times before stopping with an error. The default value is 10. The total retry count can be seen in the output of SHOW STATUS; see Section 5.1.6, “Server Status Variables”.

16.4.1.30 Replication and Time Zones

The same system time zone should be set for both master and slave. Otherwise, statements depending on the local time on the master are not replicated properly, such as statements that use the NOW() or FROM_UNIXTIME() functions. You can set the time zone in which MySQL server runs by using the --timezone=*timezone_name* option of the mysqld_safe script or by setting the TZ environment variable. See also Section 16.4.1.15, “Replication and System Functions”.

If the master is MySQL 4.1 or earlier, both master and slave should also use the same default connection time zone. That is, the --default-time-zone parameter should have the same value for both master and slave.

CONVERT_TZ(...,...,@@session.time_zone) is properly replicated only if both master and slave are running MySQL 5.0.4 or newer.

16.4.1.31 Replication and Transactions

Mixing transactional and nontransactional statements within the same transaction. In general, you should avoid transactions that update both transactional and nontransactional tables in a replication environment. You should also avoid using any statement that accesses both transactional (or temporary) and nontransactional tables and writes to any of them.

As of MySQL 5.5.2, the server uses these rules for binary logging:

- If the initial statements in a transaction are nontransactional, they are written to the binary log immediately. The remaining statements in the transaction are cached and not written to the binary log until the transaction is committed. (If the transaction is rolled back, the cached statements are written to the binary log only if they make nontransactional changes that cannot be rolled back. Otherwise, they are discarded.)

- For statement-based logging, logging of nontransactional statements is affected by the `binlog_direct_non_transactional_updates` system variable. When this variable is `OFF` (the default), logging is as just described. When this variable is `ON`, logging occurs immediately for nontransactional statements occurring anywhere in the transaction (not just initial nontransactional statements). Other statements are kept in the transaction cache and logged when the transaction commits. `binlog_direct_non_transactional_updates` has no effect for row-format or mixed-format binary logging.

Transactional, nontransactional, and mixed statements.
To apply those rules, the server considers a statement nontransactional if it changes only nontransactional tables, and transactional if it changes only transactional tables. In MySQL 5.7, a statement that references both nontransactional and transactional tables and updates *any* of the tables involved, is considered a “mixed” statement. (In previous MySQL release series, a statement that changed both nontransactional and transactional tables was considered mixed.) Mixed statements, like transactional statements, are cached and logged when the transaction commits.

A mixed statement that updates a transactional table is considered unsafe if the statement also performs either of the following actions:

- Updates or reads a transactional table

- Reads a nontransactional table and the transaction isolation level is less than REPEATABLE_READ

A mixed statement following the update of a transactional table within a transaction is considered unsafe if it performs either of the following actions:

- Updates any table and reads from any temporary table

- Updates a nontransactional table and binlog_direct_non_trans_update is OFF

For more information, see Section 16.1.2.3, “Determination of Safe and Unsafe Statements in Binary Logging”.

Note

A mixed statement is unrelated to mixed binary logging format.

In situations where transactions mix updates to transactional and nontransactional tables, the order of statements in the binary log is correct, and all needed statements are written to the binary log even in case of a `ROLLBACK`. However, when a second connection updates the nontransactional table before the first connection transaction is complete, statements can be logged out of order because the second connection update is written immediately after it is performed, regardless of the state of the transaction being performed by the first connection.

Using different storage engines on master and slave. It is possible to replicate transactional tables on the master using nontransactional tables on the slave. For example, you can replicate an `InnoDB` master table as a `MyISAM` slave table. However, if you do this, there are problems if the slave is stopped in the middle of a `BEGIN` ... `COMMIT` block because the slave restarts at the beginning of the `BEGIN` block.

In MySQL 5.7, it is also safe to replicate transactions from `MyISAM` tables on the master to transactional tables—such as tables that use the `InnoDB` storage engine—on the slave. In such cases (beginning with MySQL 5.5.0), an AUTOCOMMIT=1 statement issued on the master is replicated, thus enforcing AUTOCOMMIT mode on the slave.

When the storage engine type of the slave is nontransactional, transactions on the master that mix updates of transactional and nontransactional tables should be avoided because they can cause inconsistency of the data between the master transactional table and the slave nontransactional table. That is, such transactions can lead to master storage engine-specific behavior with the possible effect of replication going out of synchrony. MySQL does not issue a warning about this currently, so extra care should be taken when replicating transactional tables from the master to nontransactional tables on the slaves.

Changing the binary logging format within transactions. Beginning with MySQL 5.5.3, the binlog_format system variable is read-only as long as a transaction is in progress. (Bug #47863)

Every transaction (including autocommit transactions) is recorded in the binary log as though it starts with a BEGIN statement, and ends with either a COMMIT or a ROLLBACK statement. In MySQL 5.7, this true is even for statements affecting tables that use a nontransactional storage engine (such as MyISAM).

16.4.1.32 Replication and Triggers

With statement-based replication, triggers executed on the master also execute on the slave. With row-based replication, triggers executed on the master do not execute on the slave. Instead, the row changes on the master resulting from trigger execution are replicated and applied on the slave.

This behavior is by design. If under row-based replication the slave applied the triggers as well as the row changes caused by them, the changes would in effect be applied twice on the slave, leading to different data on the master and the slave.

If you want triggers to execute on both the master and the slave—perhaps because you have different triggers on the master and slave—you must use statement-based replication. However, to enable slave-side triggers, it is not necessary to use statement-based replication exclusively. It is sufficient to switch to statement-based replication only for those statements where you want this effect, and to use row-based replication the rest of the time.

A statement invoking a trigger (or function) that causes an update to an AUTO_INCREMENT column is not replicated correctly using statement-based replication. MySQL 5.7 marks such statements as unsafe. (Bug #45677)

A trigger can have triggers for different combinations of trigger event (INSERT, UPDATE, DELETE) and action time (BEFORE, AFTER), but before MySQL 5.7.2 cannot have multiple triggers that have the same trigger event and action time. MySQL 5.7.2 lifts this limitation and multiple triggers are permitted. This change has replication implications for upgrades and downgrades.

For brevity, “multiple triggers” here is shorthand for “multiple triggers that have the same trigger event and action time.”

Upgrades. Suppose that you upgrade an old server that does not support multiple triggers to MySQL 5.7.2 or newer. If the new server is a replication master and has old slaves that do not support multiple triggers, an error occurs on those slaves if a trigger is created on the master for a table that already has a trigger with the same trigger event and action time. To avoid this problem, upgrade the slaves first, then upgrade the master.

Downgrades. If you downgrade a server that supports multiple triggers to an older version that does not, the downgrade has these effects:

- For each table that has triggers, all trigger definitions remain in the .TRG file for the table. However, if there are multiple triggers with the same trigger event and action time, the server executes only one of them when the trigger event occurs. For information about .TRG files, see Table Trigger Storage.

- If triggers for the table are added or dropped subsequent to the downgrade, the server rewrites the table's .TRG file. The rewritten file retains only one trigger per combination of trigger event and action time; the others are lost.

To avoid these problems, modify your triggers before downgrading. For each table that has multiple triggers per combination of trigger event and action time, convert each such set of triggers to a single trigger as follows:

1. For each trigger, create a stored routine that contains all the code in the trigger. Values accessed using NEW and OLD can be passed to the routine using parameters. If the trigger needs a single result value from the code, you can put the code in a stored function and have the function return the value. If the trigger needs multiple result values from the code, you can put the code in a stored procedure and return the values using OUT parameters.

2. Drop all triggers for the table.

3. Create one new trigger for the table that invokes the stored routines just created. The effect for this trigger is thus the same as the multiple triggers it replaces.

16.4.1.33 Replication and TRUNCATE TABLE

TRUNCATE TABLE is normally regarded as a DML statement, and so would be expected to be logged and replicated using row-based format when the binary logging mode is ROW or MIXED. However this caused issues when logging or replicating, in STATEMENT or MIXED mode, tables that used transactional storage engines such as InnoDB when the transaction isolation level was READ COMMITTED or READ UNCOMMITTED, which precludes statement-based logging.

TRUNCATE TABLE is treated for purposes of logging and replication as DDL rather than DML so that it can be logged and replicated as a statement. However, the effects of the statement as applicable to InnoDB and other transactional tables on replication slaves still follow the rules described in Section 13.1.27, “TRUNCATE TABLE Syntax” governing such tables. (Bug #36763)

16.4.1.34 Replication and Variables

System variables are not replicated correctly when using STATEMENT mode, except for the following variables when they are used with session scope:

- auto_increment_increment
- auto_increment_offset
- character_set_client
- character_set_connection
- character_set_database
- character_set_server
- collation_connection
- collation_database
- collation_server
- foreign_key_checks
- identity

- last_insert_id
- lc_time_names
- pseudo_thread_id
- sql_auto_is_null
- time_zone
- timestamp
- unique_checks

When MIXED mode is used, the variables in the preceding list, when used with session scope, cause a switch from statement-based to row-based logging. See Section 5.2.4.3, “Mixed Binary Logging Format”.

sql_mode is also replicated except for the NO_DIR_IN_CREATE mode; the slave always preserves its own value for NO_DIR_IN_CREATE, regardless of changes to it on the master. This is true for all replication formats.

However, when mysqlbinlog parses a SET @@sql_mode = *mode* statement, the full *mode* value, including NO_DIR_IN_CREATE, is passed to the receiving server. For this reason, replication of such a statement may not be safe when STATEMENT mode is in use.

The default_storage_engine and storage_engine system variables are not replicated, regardless of the logging mode; this is intended to facilitate replication between different storage engines.

The read_only system variable is not replicated. In addition, the enabling this variable has different effects with regard to temporary tables, table locking, and the SET PASSWORD statement in different MySQL versions.

The max_heap_table_size system variable is not replicated. Increasing the value of this variable on the master without doing so on the slave can lead eventually to Table is full errors on the slave when trying to execute INSERT statements on a MEMORY table on the master that is thus permitted to grow larger than its counterpart on the slave. For more information, see Section 16.4.1.21, “Replication and MEMORY Tables”.

In statement-based replication, session variables are not replicated properly when used in statements that update tables. For example, the following sequence of statements will not insert the same data on the master and the slave:

```
SET max_join_size=1000;
INSERT INTO mytable VALUES(@@max_join_size);
```

This does not apply to the common sequence:

```
SET time_zone=...;
INSERT INTO mytable VALUES(CONVERT_TZ(..., ..., @@time_zone));
```

Replication of session variables is not a problem when row-based replication is being used, in which case, session variables are always replicated safely. See Section 16.1.2, “Replication Formats”.

In MySQL 5.7, the following session variables are written to the binary log and honored by the replication slave when parsing the binary log, regardless of the logging format:

- sql_mode

- foreign_key_checks
- unique_checks
- character_set_client
- collation_connection
- collation_database
- collation_server
- sql_auto_is_null

> **Important**
>
> Even though session variables relating to character sets and collations are written to the binary log, replication between different character sets is not supported.

To help reduce possible confusion, we recommend that you always use the same setting for the lower_case_table_names system variable on both master and slave, especially when you are running MySQL on platforms with case-sensitive file systems.

16.4.1.35 Replication and Views

Views are always replicated to slaves. Views are filtered by their own name, not by the tables they refer to. This means that a view can be replicated to the slave even if the view contains a table that would normally be filtered out by replication-ignore-table rules. Care should therefore be taken to ensure that views do not replicate table data that would normally be filtered for security reasons.

Replication from a table to a samed-named view is supported using statement-based logging, but not when using row-based logging. In MySQL 5.7.1 and later, trying to do so when row-based logging is in effect causes an error. (Bug #11752707, Bug #43975)

16.4.2 Replication Compatibility Between MySQL Versions

MySQL supports replication from one major version to the next higher major version. For example, you can replicate from a master running MySQL 5.0 to a slave running MySQL 5.1, from a master running MySQL 5.1 to a slave running MySQL 5.5, and so on.

However, one may encounter difficulties when replicating from an older master to a newer slave if the master uses statements or relies on behavior no longer supported in the version of MySQL used on the slave. For example, in MySQL 5.5, CREATE TABLE ... SELECT statements are permitted to change tables other than the one being created, but are no longer allowed to do so in MySQL 5.6 (see Section 16.4.1.5, “Replication of CREATE TABLE ... SELECT Statements”).

The use of more than 2 MySQL Server versions is not supported in replication setups involving multiple masters, regardless of the number of master or slave MySQL servers. This restriction applies not only to major versions, but to minor versions within the same major version as well. For example, if you are using a chained or circular replication setup, you cannot use MySQL 5.7.1, MySQL 5.7.2, and MySQL 5.7.4 concurrently, although you could use any 2 of these releases together.

In some cases, it is also possible to replicate between a master and a slave that is more than one major version newer than the master. However, there are known issues with trying to replicate from a master running MySQL 4.1 or earlier to a slave running MySQL 5.1 or later. To work around such problems, you can insert a MySQL server running an intermediate version between the two; for example, rather than replicating directly from a MySQL 4.1 master to a MySQL 5.1 slave, it is possible to replicate from a MySQL 4.1 server to a MySQL 5.0 server, and then from the MySQL 5.0 server to a MySQL 5.1 server.

> **Important**
>
> It is strongly recommended to use the most recent release available within a given MySQL major version because replication (and other) capabilities are continually being improved. It is also recommended to upgrade masters and slaves that use early releases of a major version of MySQL to GA (production) releases when the latter become available for that major version.

Replication from newer masters to older slaves may be possible, but is generally not supported. This is due to a number of factors:

- **Binary log format changes.** The binary log format can change between major releases. While we attempt to maintain backward compatibility, this is not always possible. For example, the binary log format implemented in MySQL 5.0 changed considerably from that used in previous versions, especially with regard to handling of character sets, `LOAD DATA INFILE`, and time zones. This means that replication from a MySQL 5.0 (or later) master to a MySQL 4.1 (or earlier) slave is generally not supported.

  This also has significant implications for upgrading replication servers; see Section 16.4.3, “Upgrading a Replication Setup”, for more information.

- **Use of row-based replication.** Row-based replication was implemented in MySQL 5.1.5, so you cannot replicate using row-based replication from any MySQL 5.7 or later master to a slave older than MySQL 5.1.5.

  For more information about row-based replication, see Section 16.1.2, “Replication Formats”.

- **SQL incompatibilities.** You cannot replicate from a newer master to an older slave using statement-based replication if the statements to be replicated use SQL features available on the master but not on the slave.

  However, if both the master and the slave support row-based replication, and there are no data definition statements to be replicated that depend on SQL features found on the master but not on the slave, you can use row-based replication to replicate the effects of data modification statements even if the DDL run on the master is not supported on the slave.

For more information on potential replication issues, see Section 16.4.1, “Replication Features and Issues”.

16.4.3 Upgrading a Replication Setup

When you upgrade servers that participate in a replication setup, the procedure for upgrading depends on the current server versions and the version to which you are upgrading.

This section applies to upgrading replication from older versions of MySQL to MySQL 5.7. A 4.0 server should be 4.0.3 or newer.

When you upgrade a master to 5.7 from an earlier MySQL release series, you should first ensure that all the slaves of this master are using the same 5.7.x release. If this is not the case, you should first upgrade the slaves. To upgrade each slave, shut it down, upgrade it to the appropriate 5.7.x version, restart it, and restart replication. Relay logs created by the slave after the upgrade are in 5.7 format.

Changes affecting operations in strict SQL mode may result in replication failure on an updated slave. For example, as of MySQL 5.7.2, the server restricts insertion of a `DEFAULT` value of 0 for temporal data types in strict mode (STRICT_TRANS_TABLES or STRICT_ALL_TABLES). A resulting incompatibility for replication if you use statement-based logging (binlog_format=STATEMENT) is that if a slave is upgraded, a nonupgraded master will execute statements without error that may fail on the slave and replication will stop. To deal with this, stop all new statements on the master and wait until the slaves catch up. Then upgrade the slaves. Alternatively, if you cannot stop new statements, temporarily change to row-based logging on the master (binlog_format=ROW) and wait until all slaves have processed all binary logs produced up to the point of this change. Then upgrade the slaves.

After the slaves have been upgraded, shut down the master, upgrade it to the same 5.7.x release as the slaves, and restart it. If you had temporarily changed the master to row-based logging, change it back to statement-based logging. The 5.7 master is able to read the old binary logs written prior to the upgrade and to send them to the 5.7 slaves. The slaves recognize the old format and handle it properly. Binary logs created by the master subsequent to the upgrade are in 5.7 format. These too are recognized by the 5.7 slaves.

In other words, when upgrading to MySQL 5.7, the slaves must be MySQL 5.7 before you can upgrade the master to 5.7. Note that downgrading from 5.7 to older versions does not work so simply: You must ensure that any 5.7 binary log or relay log has been fully processed, so that you can remove it before proceeding with the downgrade.

Downgrading a replication setup to a previous version cannot be done once you have switched from statement-based to row-based replication, and after the first row-based statement has been written to the binlog. See Section 16.1.2, “Replication Formats”.

Some upgrades may require that you drop and re-create database objects when you move from one MySQL series to the next. For example, collation changes might require that table indexes be rebuilt. Such operations, if necessary, will be detailed at Section 2.10.1.2, “Upgrading from MySQL 5.6 to 5.7”. It is safest to perform these operations separately on the slaves and the master, and to disable replication of these operations from the master to the slave. To achieve this, use the following procedure:

1. Stop all the slaves and upgrade them. Restart them with the --skip-slave-start option so that they do not connect to the master. Perform any table repair or rebuilding operations needed to re-create database objects, such as use of REPAIR TABLE or ALTER TABLE, or dumping and reloading tables or triggers.

2. Disable the binary log on the master. To do this without restarting the master, execute a SET sql_log_bin = 0 statement. Alternatively, stop the master and restart it without the --log-bin option. If you restart the master, you might also want to disallow client connections. For example, if all clients connect using TCP/IP, use the --skip-networking option when you restart the master.

3. With the binary log disabled, perform any table repair or rebuilding operations needed to re-create database objects. The binary log must be disabled during this step to prevent these operations from being logged and sent to the slaves later.

4. Re-enable the binary log on the master. If you set sql_log_bin to 0 earlier, execute a SET sql_log_bin = 1 statement. If you restarted the master to disable the binary log, restart it with --log-bin, and without --skip-networking so that clients and slaves can connect.

5. Restart the slaves, this time without the --skip-slave-start option.

If you are upgrading an existing replication setup from a version of MySQL that does not support global transaction identifiers to a version that does, you should not enable GTIDs on either the master or the slave before making sure that the setup meets all the requirements for GTID-based replication. See Section 16.1.3.2, “Setting Up Replication Using GTIDs”, which contains information about converting existing replication setups to use GTID-based replication.

16.4.4 Troubleshooting Replication

If you have followed the instructions but your replication setup is not working, the first thing to do is *check the error log for messages*. Many users have lost time by not doing this soon enough after encountering problems.

If you cannot tell from the error log what the problem was, try the following techniques:

- Verify that the master has binary logging enabled by issuing a `SHOW MASTER STATUS` statement. If logging is enabled, `Position` is nonzero. If binary logging is not enabled, verify that you are running the master with the `--log-bin` option.
- Verify that the master and slave both were started with the `--server-id` [2162] option and that the ID value is unique on each server.
- Verify that the slave is running. Use `SHOW SLAVE STATUS` to check whether the `Slave_IO_Running` and `Slave_SQL_Running` values are both `Yes`. If not, verify the options that were used when starting the slave server. For example, `--skip-slave-start` prevents the slave threads from starting until you issue a `START SLAVE` statement.
- If the slave is running, check whether it established a connection to the master. Use `SHOW PROCESSLIST`, find the I/O and SQL threads and check their `State` column to see what they display. See Section 16.2.1, “Replication Implementation Details”. If the I/O thread state says `Connecting to master`, check the following:
  - Verify the privileges for the user being used for replication on the master.
  - Check that the host name of the master is correct and that you are using the correct port to connect to the master. The port used for replication is the same as used for client network communication (the default is `3306`). For the host name, ensure that the name resolves to the correct IP address.
  - Check that networking has not been disabled on the master or slave. Look for the `skip-networking` option in the configuration file. If present, comment it out or remove it.
  - If the master has a firewall or IP filtering configuration, ensure that the network port being used for MySQL is not being filtered.
  - Check that you can reach the master by using `ping` or `traceroute`/`tracert` to reach the host.
- If the slave was running previously but has stopped, the reason usually is that some statement that succeeded on the master failed on the slave. This should never happen if you have taken a proper snapshot of the master, and never modified the data on the slave outside of the slave thread. If the slave stops unexpectedly, it is a bug or you have encountered one of the known replication limitations described in Section 16.4.1, “Replication Features and Issues”. If it is a bug, see Section 16.4.5, “How to Report Replication Bugs or Problems”, for instructions on how to report it.
- If a statement that succeeded on the master refuses to run on the slave, try the following procedure if it is not feasible to do a full database resynchronization by deleting the slave's databases and copying a new snapshot from the master:
  1. Determine whether the affected table on the slave is different from the master table. Try to understand how this happened. Then make the slave's table identical to the master's and run `START SLAVE`.
  2. If the preceding step does not work or does not apply, try to understand whether it would be safe to make the update manually (if needed) and then ignore the next statement from the master.

3. If you decide that the slave can skip the next statement from the master, issue the following statements:

   ```
   mysql> SET GLOBAL sql_slave_skip_counter = N;
   mysql> START SLAVE;
   ```

   The value of *N* should be 1 if the next statement from the master does not use AUTO_INCREMENT or LAST_INSERT_ID(). Otherwise, the value should be 2. The reason for using a value of 2 for statements that use AUTO_INCREMENT or LAST_INSERT_ID() is that they take two events in the binary log of the master.

   See also Section 13.4.2.5, "SET GLOBAL sql_slave_skip_counter Syntax".

4. If you are sure that the slave started out perfectly synchronized with the master, and that no one has updated the tables involved outside of the slave thread, then presumably the discrepancy is the result of a bug. If you are running the most recent version of MySQL, please report the problem. If you are running an older version, try upgrading to the latest production release to determine whether the problem persists.

16.4.5 How to Report Replication Bugs or Problems

When you have determined that there is no user error involved, and replication still either does not work at all or is unstable, it is time to send us a bug report. We need to obtain as much information as possible from you to be able to track down the bug. Please spend some time and effort in preparing a good bug report.

If you have a repeatable test case that demonstrates the bug, please enter it into our bugs database using the instructions given in Section 1.7, "How to Report Bugs or Problems". If you have a "phantom" problem (one that you cannot duplicate at will), use the following procedure:

1. Verify that no user error is involved. For example, if you update the slave outside of the slave thread, the data goes out of synchrony, and you can have unique key violations on updates. In this case, the slave thread stops and waits for you to clean up the tables manually to bring them into synchrony. *This is not a replication problem. It is a problem of outside interference causing replication to fail.*

2. Run the slave with the --log-slave-updates and --log-bin options. These options cause the slave to log the updates that it receives from the master into its own binary logs.

3. Save all evidence before resetting the replication state. If we have no information or only sketchy information, it becomes difficult or impossible for us to track down the problem. The evidence you should collect is:

   - All binary log files from the master

   - All binary log files from the slave

   - The output of SHOW MASTER STATUS from the master at the time you discovered the problem

   - The output of SHOW SLAVE STATUS from the slave at the time you discovered the problem

   - Error logs from the master and the slave

4. Use mysqlbinlog to examine the binary logs. The following should be helpful to find the problem statement. *log_file* and *log_pos* are the Master_Log_File and Read_Master_Log_Pos values from SHOW SLAVE STATUS.

```
shell> mysqlbinlog --start-position=log_pos log_file | head
```

After you have collected the evidence for the problem, try to isolate it as a separate test case first. Then enter the problem with as much information as possible into our bugs database using the instructions at Section 1.7, “How to Report Bugs or Problems”.

Chapter 17 Partitioning

Table of Contents

This chapter discusses MySQL's implementation of *user-defined partitioning*. You can determine whether your MySQL Server supports partitioning by checking the output of the SHOW PLUGINS statement, as shown here:

```
mysql> SHOW PLUGINS;
+------------+----------+----------------+---------+---------+
| Name       | Status   | Type           | Library | License |
+------------+----------+----------------+---------+---------+
| binlog     | ACTIVE   | STORAGE ENGINE | NULL    | GPL     |
| partition  | ACTIVE   | STORAGE ENGINE | NULL    | GPL     |
| ARCHIVE    | ACTIVE   | STORAGE ENGINE | NULL    | GPL     |
| BLACKHOLE  | ACTIVE   | STORAGE ENGINE | NULL    | GPL     |
| CSV        | ACTIVE   | STORAGE ENGINE | NULL    | GPL     |
| FEDERATED  | DISABLED | STORAGE ENGINE | NULL    | GPL     |
| MEMORY     | ACTIVE   | STORAGE ENGINE | NULL    | GPL     |
| InnoDB     | ACTIVE   | STORAGE ENGINE | NULL    | GPL     |
| MRG_MYISAM | ACTIVE   | STORAGE ENGINE | NULL    | GPL     |
| MyISAM     | ACTIVE   | STORAGE ENGINE | NULL    | GPL     |
| ndbcluster | DISABLED | STORAGE ENGINE | NULL    | GPL     |
+------------+----------+----------------+---------+---------+
11 rows in set (0.00 sec)
```

You can also check the INFORMATION_SCHEMA.PLUGINS table with a query similar to this one:

```
mysql> SELECT
    ->     PLUGIN_NAME as Name,
    ->     PLUGIN_VERSION as Version,
    ->     PLUGIN_STATUS as Status
    -> FROM INFORMATION_SCHEMA.PLUGINS
    -> WHERE PLUGIN_TYPE='STORAGE ENGINE';
+--------------------+---------+--------+
| Name               | Version | Status |
+--------------------+---------+--------+
| binlog             | 1.0     | ACTIVE |
| CSV                | 1.0     | ACTIVE |
| MEMORY             | 1.0     | ACTIVE |
| MRG_MYISAM         | 1.0     | ACTIVE |
| MyISAM             | 1.0     | ACTIVE |
| PERFORMANCE_SCHEMA | 0.1     | ACTIVE |
| BLACKHOLE          | 1.0     | ACTIVE |
| ARCHIVE            | 3.0     | ACTIVE |
| InnoDB             | 5.6     | ACTIVE |
| partition          | 1.0     | ACTIVE |
+--------------------+---------+--------+
10 rows in set (0.00 sec)
```

In either case, if you do not see the `partition` plugin listed with the value `ACTIVE` for the `Status` column in the output (shown in bold text in each of the examples just given), then your version of MySQL was not built with partitioning support.

MySQL 5.7 Community binaries provided by Oracle include partitioning support. For information about partitioning support offered in commercial MySQL Server binaries, see *MySQL Enterprise Server 5.1* on the MySQL Web site at http://www.mysql.com/products/enterprise/server.html.

To enable partitioning if you are compiling MySQL 5.7 from source, the build must be configured with the `-DWITH_PARTITION_STORAGE_ENGINE` option. For more information, see Section 2.8, “Installing MySQL from Source”.

If your MySQL binary is built with partitioning support, nothing further needs to be done to enable it (for example, no special entries are required in your `my.cnf` file).

If you want to disable partitioning support, you can start the MySQL Server with the `--skip-partition` option, in which case the value of `have_partitioning` is `DISABLED`. When partitioning support is disabled, you can see any existing partitioned tables and drop them (although doing this is not advised), but you cannot otherwise manipulate them or access their data.

See Section 17.1, “Overview of Partitioning in MySQL”, for an introduction to partitioning and partitioning concepts.

MySQL supports several types of partitioning as well as subpartitioning; see Section 17.2, “Partitioning Types”, and Section 17.2.6, “Subpartitioning”.

Section 17.3, “Partition Management”, covers methods of adding, removing, and altering partitions in existing partitioned tables.

Section 17.3.4, “Maintenance of Partitions”, discusses table maintenance commands for use with partitioned tables.

The `PARTITIONS` table in the `INFORMATION_SCHEMA` database provides information about partitions and partitioned tables. See Section 19.14, “The `INFORMATION_SCHEMA PARTITIONS` Table”, for more information; for some examples of queries against this table, see Section 17.2.7, “How MySQL Partitioning Handles `NULL`”.

For known issues with partitioning in MySQL 5.7, see Section 17.6, “Restrictions and Limitations on Partitioning”.

You may also find the following resources to be useful when working with partitioned tables.

Additional Resources. Other sources of information about user-defined partitioning in MySQL include the following:

- MySQL Partitioning Forum

  This is the official discussion forum for those interested in or experimenting with MySQL Partitioning technology. It features announcements and updates from MySQL developers and others. It is monitored by members of the Partitioning Development and Documentation Teams.

- Mikael Ronström's Blog

  MySQL Partitioning Architect and Lead Developer Mikael Ronström frequently posts articles here concerning his work with MySQL Partitioning and MySQL Cluster.

- PlanetMySQL

  A MySQL news site featuring MySQL-related blogs, which should be of interest to anyone using my MySQL. We encourage you to check here for links to blogs kept by those working with MySQL Partitioning, or to have your own blog added to those covered.

MySQL 5.7 binaries are available from http://dev.mysql.com/downloads/mysql/5.7.html. However, for the latest partitioning bugfixes and feature additions, you can obtain the source from our Bazaar repository. To enable partitioning, the build must be configured with the `-DWITH_PARTITION_STORAGE_ENGINE` option. For more information about building MySQL, see Section 2.8, “Installing MySQL from Source”. If you have problems compiling a partitioning-enabled MySQL 5.7 build, check the MySQL Partitioning Forum and ask for assistance there if you do not find a solution to your problem already posted.

17.1 Overview of Partitioning in MySQL

This section provides a conceptual overview of partitioning in MySQL 5.7.

For information on partitioning restrictions and feature limitations, see Section 17.6, “Restrictions and Limitations on Partitioning”.

The SQL standard does not provide much in the way of guidance regarding the physical aspects of data storage. The SQL language itself is intended to work independently of any data structures or media underlying the schemas, tables, rows, or columns with which it works. Nonetheless, most advanced database management systems have evolved some means of determining the physical location to be used for storing specific pieces of data in terms of the file system, hardware or even both. In MySQL, the `InnoDB` storage engine has long supported the notion of a tablespace, and the MySQL Server, even prior to the introduction of partitioning, could be configured to employ different physical directories for storing different databases (see Section 8.11.3.1, “Using Symbolic Links”, for an explanation of how this is done).

Partitioning takes this notion a step further, by enabling you to distribute portions of individual tables across a file system according to rules which you can set largely as needed. In effect, different portions of a table are stored as separate tables in different locations. The user-selected rule by which the division of data is accomplished is known as a *partitioning function*, which in MySQL can be the modulus, simple matching against a set of ranges or value lists, an internal hashing function, or a linear hashing function. The function is selected according to the partitioning type specified by the user, and takes as its parameter the value of a user-supplied expression. This expression can be a column value, a function acting on one or more column values, or a set of one or more column values, depending on the type of partitioning that is used.

In the case of `RANGE`, `LIST`, and [`LINEAR`] `HASH` partitioning, the value of the partitioning column is passed to the partitioning function, which returns an integer value representing the number of the partition in which that particular record should be stored. This function must be nonconstant and nonrandom. It may not contain any queries, but may use an SQL expression that is valid in MySQL, as long as that expression returns either NULL or an integer *intval* such that

```
-MAXVALUE <= intval <= MAXVALUE
```

(MAXVALUE is used to represent the least upper bound for the type of integer in question. -MAXVALUE represents the greatest lower bound.)

For [LINEAR] KEY, RANGE COLUMNS, and LIST COLUMNS partitioning, the partitioning expression consists of a list of one or more columns.

For [LINEAR] KEY partitioning, the partitioning function is supplied by MySQL.

For more information about permitted partitioning column types and partitioning functions, see Section 17.2, “Partitioning Types”, as well as Section 13.1.14, “CREATE TABLE Syntax”, which provides partitioning syntax descriptions and additional examples. For information about restrictions on partitioning functions, see Section 17.6.3, “Partitioning Limitations Relating to Functions”.

This is known as *horizontal partitioning*—that is, different rows of a table may be assigned to different physical partitions. MySQL 5.7 does not support *vertical partitioning*, in which different columns of a table are assigned to different physical partitions. There are not at this time any plans to introduce vertical partitioning into MySQL 5.7.

For information about determining whether your MySQL Server binary supports user-defined partitioning, see Chapter 17, *Partitioning*.

For creating partitioned tables, you can use most storage engines that are supported by your MySQL server; the MySQL partitioning engine runs in a separate layer and can interact with any of these. In MySQL 5.7, all partitions of the same partitioned table must use the same storage engine; for example, you cannot use MyISAM for one partition and InnoDB for another. However, there is nothing preventing you from using different storage engines for different partitioned tables on the same MySQL server or even in the same database.

MySQL partitioning cannot be used with the MERGE, CSV, or FEDERATED storage engines.

To employ a particular storage engine for a partitioned table, it is necessary only to use the [STORAGE] ENGINE option just as you would for a nonpartitioned table. However, you should keep in mind that [STORAGE] ENGINE (and other table options) need to be listed *before* any partitioning options are used in a CREATE TABLE statement. This example shows how to create a table that is partitioned by hash into 6 partitions and which uses the InnoDB storage engine:

```
CREATE TABLE ti (id INT, amount DECIMAL(7,2), tr_date DATE)
    ENGINE=INNODB
    PARTITION BY HASH( MONTH(tr_date) )
    PARTITIONS 6;
```

Each PARTITION clause can include a [STORAGE] ENGINE option, but in MySQL 5.7 this has no effect.

Important

Partitioning applies to all data and indexes of a table; you cannot partition only the data and not the indexes, or *vice versa*, nor can you partition only a portion of the table.

Data and indexes for each partition can be assigned to a specific directory using the DATA DIRECTORY and INDEX DIRECTORY options for the PARTITION clause of the CREATE TABLE statement used to create the partitioned table.

The DATA DIRECTORY and INDEX DIRECTORY options have no effect when defining partitions for tables using the InnoDB storage engine.

DATA DIRECTORY and INDEX DIRECTORY are not supported for individual partitions or subpartitions on Windows. These options are ignored on Windows, except that a warning is generated.

In addition, MAX_ROWS and MIN_ROWS can be used to determine the maximum and minimum numbers of rows, respectively, that can be stored in each partition. See Section 17.3, “Partition Management”, for more information on these options.

Some advantages of partitioning are listed here:

- Partitioning makes it possible to store more data in one table than can be held on a single disk or file system partition.

- Data that loses its usefulness can often be easily removed from a partitioned table by dropping the partition (or partitions) containing only that data. Conversely, the process of adding new data can in some cases be greatly facilitated by adding one or more new partitions for storing specifically that data.

- Some queries can be greatly optimized in virtue of the fact that data satisfying a given WHERE clause can be stored only on one or more partitions, which automatically excludes any remaining partitions from the search. Because partitions can be altered after a partitioned table has been created, you can reorganize your data to enhance frequent queries that may not have been often used when the partitioning scheme was first set up. This ability to exclude non-matching partitions (and thus any rows they contain) is often referred to as *partition pruning*. For more information, see Section 17.4, “Partition Pruning”.

  In addition, MySQL 5.7 supports explicit partition selection for queries. For example, SELECT * FROM t PARTITION (p0,p1) WHERE c < 5 selects only those rows in partitions p0 and p1 that match the WHERE condition. In this case, MySQL does not check any other partitions of table t; this can greatly speed up queries when you already know which partition or partitions you wish to examine. Partition selection is also supported for the data modification statements DELETE, INSERT, REPLACE, UPDATE, and LOAD DATA, LOAD XML. See the descriptions of these statements for more information and examples.

Other benefits usually associated with partitioning include those in the following list. These features are not currently implemented in MySQL Partitioning, but are high on our list of priorities.

- Queries involving aggregate functions such as SUM() and COUNT() can easily be parallelized. A simple example of such a query might be SELECT salesperson_id, COUNT(orders) as order_total FROM sales GROUP BY salesperson_id;. By “parallelized,” we mean that the query can be run simultaneously on each partition, and the final result obtained merely by summing the results obtained for all partitions.

- Achieving greater query throughput in virtue of spreading data seeks over multiple disks.

Be sure to check this section and chapter frequently for updates as MySQL Partitioning development continues.

17.2 Partitioning Types

This section discusses the types of partitioning which are available in MySQL 5.7. These include the types listed here:

- **RANGE partitioning.** This type of partitioning assigns rows to partitions based on column values falling within a given range. See Section 17.2.1, “RANGE Partitioning”. For information about an extension to this type, RANGE COLUMNS, see Section 17.2.3.1, “RANGE COLUMNS partitioning”.

- **LIST partitioning.** Similar to partitioning by RANGE, except that the partition is selected based on columns matching one of a set of discrete values. See Section 17.2.2, "LIST Partitioning". For information about an extension to this type, LIST COLUMNS, see Section 17.2.3.2, "LIST COLUMNS partitioning".

- **HASH partitioning.** With this type of partitioning, a partition is selected based on the value returned by a user-defined expression that operates on column values in rows to be inserted into the table. The function may consist of any expression valid in MySQL that yields a nonnegative integer value. An extension to this type, LINEAR HASH, is also available. See Section 17.2.4, "HASH Partitioning".

- **KEY partitioning.** This type of partitioning is similar to partitioning by HASH, except that only one or more columns to be evaluated are supplied, and the MySQL server provides its own hashing function. These columns can contain other than integer values, since the hashing function supplied by MySQL guarantees an integer result regardless of the column data type. An extension to this type, LINEAR KEY, is also available. See Section 17.2.5, "KEY Partitioning".

A very common use of database partitioning is to segregate data by date. Some database systems support explicit date partitioning, which MySQL does not implement in 5.7. However, it is not difficult in MySQL to create partitioning schemes based on DATE, TIME, or DATETIME columns, or based on expressions making use of such columns.

When partitioning by KEY or LINEAR KEY, you can use a DATE, TIME, or DATETIME column as the partitioning column without performing any modification of the column value. For example, this table creation statement is perfectly valid in MySQL:

```
CREATE TABLE members (
    firstname VARCHAR(25) NOT NULL,
    lastname VARCHAR(25) NOT NULL,
    username VARCHAR(16) NOT NULL,
    email VARCHAR(35),
    joined DATE NOT NULL
)
PARTITION BY KEY(joined)
PARTITIONS 6;
```

In MySQL 5.7, it is also possible to use a DATE or DATETIME column as the partitioning column using RANGE COLUMNS and LIST COLUMNS partitioning.

MySQL's other partitioning types, however, require a partitioning expression that yields an integer value or NULL. If you wish to use date-based partitioning by RANGE, LIST, HASH, or LINEAR HASH, you can simply employ a function that operates on a DATE, TIME, or DATETIME column and returns such a value, as shown here:

```
CREATE TABLE members (
    firstname VARCHAR(25) NOT NULL,
    lastname VARCHAR(25) NOT NULL,
    username VARCHAR(16) NOT NULL,
    email VARCHAR(35),
    joined DATE NOT NULL
)
PARTITION BY RANGE( YEAR(joined) ) (
    PARTITION p0 VALUES LESS THAN (1960),
    PARTITION p1 VALUES LESS THAN (1970),
    PARTITION p2 VALUES LESS THAN (1980),
    PARTITION p3 VALUES LESS THAN (1990),
    PARTITION p4 VALUES LESS THAN MAXVALUE
);
```

Additional examples of partitioning using dates may be found in the following sections of this chapter:

- Section 17.2.1, “RANGE Partitioning”
- Section 17.2.4, “HASH Partitioning”
- Section 17.2.4.1, “LINEAR HASH Partitioning”

For more complex examples of date-based partitioning, see the following sections:

- Section 17.4, “Partition Pruning”
- Section 17.2.6, “Subpartitioning”

MySQL partitioning is optimized for use with the TO_DAYS(), YEAR(), and TO_SECONDS() functions. However, you can use other date and time functions that return an integer or NULL, such as WEEKDAY(), DAYOFYEAR(), or MONTH(). See Section 12.7, “Date and Time Functions”, for more information about such functions.

It is important to remember—regardless of the type of partitioning that you use—that partitions are always numbered automatically and in sequence when created, starting with 0. When a new row is inserted into a partitioned table, it is these partition numbers that are used in identifying the correct partition. For example, if your table uses 4 partitions, these partitions are numbered 0, 1, 2, and 3. For the RANGE and LIST partitioning types, it is necessary to ensure that there is a partition defined for each partition number. For HASH partitioning, the user function employed must return an integer value greater than 0. For KEY partitioning, this issue is taken care of automatically by the hashing function which the MySQL server employs internally.

Names of partitions generally follow the rules governing other MySQL identifiers, such as those for tables and databases. However, you should note that partition names are not case-sensitive. For example, the following CREATE TABLE statement fails as shown:

```
mysql> CREATE TABLE t2 (val INT)
    -> PARTITION BY LIST(val)(
    ->     PARTITION mypart VALUES IN (1,3,5),
    ->     PARTITION MyPart VALUES IN (2,4,6)
    -> );
ERROR 1488 (HY000): Duplicate partition name mypart
```

Failure occurs because MySQL sees no difference between the partition names mypart and MyPart.

When you specify the number of partitions for the table, this must be expressed as a positive, nonzero integer literal with no leading zeros, and may not be an expression such as 0.8E+01 or 6-2, even if it evaluates to an integer value. Decimal fractions are not permitted.

In the sections that follow, we do not necessarily provide all possible forms for the syntax that can be used for creating each partition type; this information may be found in Section 13.1.14, “CREATE TABLE Syntax”.

17.2.1 RANGE Partitioning

A table that is partitioned by range is partitioned in such a way that each partition contains rows for which the partitioning expression value lies within a given range. Ranges should be contiguous but not overlapping, and are defined using the VALUES LESS THAN operator. For the next few examples, suppose that you are creating a table such as the following to hold personnel records for a chain of 20 video stores, numbered 1 through 20:

```
CREATE TABLE employees (
    id INT NOT NULL,
    fname VARCHAR(30),
    lname VARCHAR(30),
    hired DATE NOT NULL DEFAULT '1970-01-01',
    separated DATE NOT NULL DEFAULT '9999-12-31',
    job_code INT NOT NULL,
    store_id INT NOT NULL
);
```

Note

The `employees` table used here has no primary or unique keys. While the examples work as shown for purposes of the present discussion, you should keep in mind that tables are extremely likely in practice to have primary keys, unique keys, or both, and that allowable choices for partitioning columns depend on the columns used for these keys, if any are present. For a discussion of these issues, see Section 17.6.1, “Partitioning Keys, Primary Keys, and Unique Keys”.

This table can be partitioned by range in a number of ways, depending on your needs. One way would be to use the `store_id` column. For instance, you might decide to partition the table 4 ways by adding a `PARTITION BY RANGE` clause as shown here:

```
CREATE TABLE employees (
    id INT NOT NULL,
    fname VARCHAR(30),
    lname VARCHAR(30),
    hired DATE NOT NULL DEFAULT '1970-01-01',
    separated DATE NOT NULL DEFAULT '9999-12-31',
    job_code INT NOT NULL,
    store_id INT NOT NULL
)
PARTITION BY RANGE (store_id) (
    PARTITION p0 VALUES LESS THAN (6),
    PARTITION p1 VALUES LESS THAN (11),
    PARTITION p2 VALUES LESS THAN (16),
    PARTITION p3 VALUES LESS THAN (21)
);
```

In this partitioning scheme, all rows corresponding to employees working at stores 1 through 5 are stored in partition `p0`, to those employed at stores 6 through 10 are stored in partition `p1`, and so on. Note that each partition is defined in order, from lowest to highest. This is a requirement of the `PARTITION BY RANGE` syntax; you can think of it as being analogous to a series of `if ... elseif ...` statements in C or Java in this regard.

It is easy to determine that a new row containing the data `(72, 'Michael', 'Widenius', '1998-06-25', NULL, 13)` is inserted into partition `p2`, but what happens when your chain adds a 21st store? Under this scheme, there is no rule that covers a row whose `store_id` is greater than 20, so an error results because the server does not know where to place it. You can keep this from occurring by using a “catchall” `VALUES LESS THAN` clause in the `CREATE TABLE` statement that provides for all values greater than the highest value explicitly named:

```
CREATE TABLE employees (
    id INT NOT NULL,
    fname VARCHAR(30),
    lname VARCHAR(30),
    hired DATE NOT NULL DEFAULT '1970-01-01',
    separated DATE NOT NULL DEFAULT '9999-12-31',
    job_code INT NOT NULL,
    store_id INT NOT NULL
)
PARTITION BY RANGE (store_id) (
    PARTITION p0 VALUES LESS THAN (6),
    PARTITION p1 VALUES LESS THAN (11),
    PARTITION p2 VALUES LESS THAN (16),
    PARTITION p3 VALUES LESS THAN MAXVALUE
);
```

Note

Another way to avoid an error when no matching value is found is to use the IGNORE keyword as part of the INSERT statement. For an example, see Section 17.2.2, “LIST Partitioning”. Also see Section 13.2.5, “INSERT Syntax”, for general information about IGNORE.

MAXVALUE represents an integer value that is always greater than the largest possible integer value (in mathematical language, it serves as a *least upper bound*). Now, any rows whose store_id column value is greater than or equal to 16 (the highest value defined) are stored in partition p3. At some point in the future—when the number of stores has increased to 25, 30, or more—you can use an ALTER TABLE statement to add new partitions for stores 21-25, 26-30, and so on (see Section 17.3, “Partition Management”, for details of how to do this).

In much the same fashion, you could partition the table based on employee job codes—that is, based on ranges of job_code column values. For example—assuming that two-digit job codes are used for regular (in-store) workers, three-digit codes are used for office and support personnel, and four-digit codes are used for management positions—you could create the partitioned table using the following statement:

```
CREATE TABLE employees (
    id INT NOT NULL,
    fname VARCHAR(30),
    lname VARCHAR(30),
    hired DATE NOT NULL DEFAULT '1970-01-01',
    separated DATE NOT NULL DEFAULT '9999-12-31',
    job_code INT NOT NULL,
    store_id INT NOT NULL
)
PARTITION BY RANGE (job_code) (
    PARTITION p0 VALUES LESS THAN (100),
    PARTITION p1 VALUES LESS THAN (1000),
    PARTITION p2 VALUES LESS THAN (10000)
);
```

In this instance, all rows relating to in-store workers would be stored in partition p0, those relating to office and support staff in p1, and those relating to managers in partition p2.

It is also possible to use an expression in VALUES LESS THAN clauses. However, MySQL must be able to evaluate the expression's return value as part of a LESS THAN (<) comparison.

Rather than splitting up the table data according to store number, you can use an expression based on one of the two DATE columns instead. For example, let us suppose that you wish to partition based on the year that each employee left the company; that is, the value of YEAR(separated). An example of a CREATE TABLE statement that implements such a partitioning scheme is shown here:

```
CREATE TABLE employees (
    id INT NOT NULL,
    fname VARCHAR(30),
    lname VARCHAR(30),
    hired DATE NOT NULL DEFAULT '1970-01-01',
    separated DATE NOT NULL DEFAULT '9999-12-31',
    job_code INT,
    store_id INT
)
PARTITION BY RANGE ( YEAR(separated) ) (
    PARTITION p0 VALUES LESS THAN (1991),
    PARTITION p1 VALUES LESS THAN (1996),
    PARTITION p2 VALUES LESS THAN (2001),
    PARTITION p3 VALUES LESS THAN MAXVALUE
);
```

In this scheme, for all employees who left before 1991, the rows are stored in partition p0; for those who left in the years 1991 through 1995, in p1; for those who left in the years 1996 through 2000, in p2; and for any workers who left after the year 2000, in p3.

It is also possible to partition a table by RANGE, based on the value of a TIMESTAMP column, using the UNIX_TIMESTAMP() function, as shown in this example:

```
CREATE TABLE quarterly_report_status (
    report_id INT NOT NULL,
    report_status VARCHAR(20) NOT NULL,
    report_updated TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
)
PARTITION BY RANGE ( UNIX_TIMESTAMP(report_updated) ) (
    PARTITION p0 VALUES LESS THAN ( UNIX_TIMESTAMP('2008-01-01 00:00:00') ),
    PARTITION p1 VALUES LESS THAN ( UNIX_TIMESTAMP('2008-04-01 00:00:00') ),
    PARTITION p2 VALUES LESS THAN ( UNIX_TIMESTAMP('2008-07-01 00:00:00') ),
    PARTITION p3 VALUES LESS THAN ( UNIX_TIMESTAMP('2008-10-01 00:00:00') ),
    PARTITION p4 VALUES LESS THAN ( UNIX_TIMESTAMP('2009-01-01 00:00:00') ),
    PARTITION p5 VALUES LESS THAN ( UNIX_TIMESTAMP('2009-04-01 00:00:00') ),
    PARTITION p6 VALUES LESS THAN ( UNIX_TIMESTAMP('2009-07-01 00:00:00') ),
    PARTITION p7 VALUES LESS THAN ( UNIX_TIMESTAMP('2009-10-01 00:00:00') ),
    PARTITION p8 VALUES LESS THAN ( UNIX_TIMESTAMP('2010-01-01 00:00:00') ),
    PARTITION p9 VALUES LESS THAN (MAXVALUE)
);
```

Any other expressions involving TIMESTAMP values are not permitted. (See Bug #42849.)

Range partitioning is particularly useful when one or more of the following conditions is true:

- You want or need to delete "old" data. If you are using the partitioning scheme shown immediately above, you can simply use ALTER TABLE employees DROP PARTITION p0; to delete all rows relating to employees who stopped working for the firm prior to 1991. (See Section 13.1.6, "ALTER TABLE Syntax", and Section 17.3, "Partition Management", for more information.) For a table with a great many rows, this can be much more efficient than running a DELETE query such as DELETE FROM employees WHERE YEAR(separated) <= 1990;.

- You want to use a column containing date or time values, or containing values arising from some other series.

- You frequently run queries that depend directly on the column used for partitioning the table. For example, when executing a query such as EXPLAIN PARTITIONS SELECT COUNT(*) FROM employees WHERE separated BETWEEN '2000-01-01' AND '2000-12-31' GROUP BY store_id;, MySQL can quickly determine that only partition p2 needs to be scanned because the remaining partitions cannot contain any records satisfying the WHERE clause. See Section 17.4, "Partition Pruning", for more information about how this is accomplished.

A variant on this type of partitioning is RANGE COLUMNS partitioning. Partitioning by RANGE COLUMNS makes it possible to employ multiple columns for defining partitioning ranges that apply both to placement of rows in partitions and for determining the inclusion or exclusion of specific partitions when performing partition pruning. See Section 17.2.3.1, "RANGE COLUMNS partitioning", for more information.

Partitioning schemes based on time intervals. If you wish to implement a partitioning scheme based on ranges or intervals of time in MySQL 5.7, you have two options:

1. Partition the table by `RANGE`, and for the partitioning expression, employ a function operating on a `DATE`, `TIME`, or `DATETIME` column and returning an integer value, as shown here:

   ```
   CREATE TABLE members (
       firstname VARCHAR(25) NOT NULL,
       lastname VARCHAR(25) NOT NULL,
       username VARCHAR(16) NOT NULL,
       email VARCHAR(35),
       joined DATE NOT NULL
   )
   PARTITION BY RANGE( YEAR(joined) ) (
       PARTITION p0 VALUES LESS THAN (1960),
       PARTITION p1 VALUES LESS THAN (1970),
       PARTITION p2 VALUES LESS THAN (1980),
       PARTITION p3 VALUES LESS THAN (1990),
       PARTITION p4 VALUES LESS THAN MAXVALUE
   );
   ```

   In MySQL 5.7, it is also possible to partition a table by `RANGE` based on the value of a `TIMESTAMP` column, using the `UNIX_TIMESTAMP()` function, as shown in this example:

   ```
   CREATE TABLE quarterly_report_status (
       report_id INT NOT NULL,
       report_status VARCHAR(20) NOT NULL,
       report_updated TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
   )
   PARTITION BY RANGE ( UNIX_TIMESTAMP(report_updated) ) (
       PARTITION p0 VALUES LESS THAN ( UNIX_TIMESTAMP('2008-01-01 00:00:00') ),
       PARTITION p1 VALUES LESS THAN ( UNIX_TIMESTAMP('2008-04-01 00:00:00') ),
       PARTITION p2 VALUES LESS THAN ( UNIX_TIMESTAMP('2008-07-01 00:00:00') ),
       PARTITION p3 VALUES LESS THAN ( UNIX_TIMESTAMP('2008-10-01 00:00:00') ),
       PARTITION p4 VALUES LESS THAN ( UNIX_TIMESTAMP('2009-01-01 00:00:00') ),
       PARTITION p5 VALUES LESS THAN ( UNIX_TIMESTAMP('2009-04-01 00:00:00') ),
       PARTITION p6 VALUES LESS THAN ( UNIX_TIMESTAMP('2009-07-01 00:00:00') ),
       PARTITION p7 VALUES LESS THAN ( UNIX_TIMESTAMP('2009-10-01 00:00:00') ),
       PARTITION p8 VALUES LESS THAN ( UNIX_TIMESTAMP('2010-01-01 00:00:00') ),
       PARTITION p9 VALUES LESS THAN (MAXVALUE)
   );
   ```

   In MySQL 5.7, any other expressions involving `TIMESTAMP` values are not permitted. (See Bug #42849.)

   > **Note**
   >
   > It is also possible in MySQL 5.7 to use `UNIX_TIMESTAMP(timestamp_column)` as a partitioning expression for tables that are partitioned by `LIST`. However, it is usually not practical to do so.

2. Partition the table by `RANGE COLUMNS`, using a `DATE` or `DATETIME` column as the partitioning column. For example, the `members` table could be defined using the `joined` column directly, as shown here:

   ```
   CREATE TABLE members (
       firstname VARCHAR(25) NOT NULL,
       lastname VARCHAR(25) NOT NULL,
       username VARCHAR(16) NOT NULL,
       email VARCHAR(35),
       joined DATE NOT NULL
   )
   ```

```
PARTITION BY RANGE COLUMNS(joined) (
    PARTITION p0 VALUES LESS THAN ('1960-01-01'),
    PARTITION p1 VALUES LESS THAN ('1970-01-01'),
    PARTITION p2 VALUES LESS THAN ('1980-01-01'),
    PARTITION p3 VALUES LESS THAN ('1990-01-01'),
    PARTITION p4 VALUES LESS THAN MAXVALUE
);
```

Note

The use of partitioning columns employing date or time types other than DATE or DATETIME is not supported with RANGE COLUMNS.

17.2.2 LIST Partitioning

List partitioning in MySQL is similar to range partitioning in many ways. As in partitioning by RANGE, each partition must be explicitly defined. The chief difference between the two types of partitioning is that, in list partitioning, each partition is defined and selected based on the membership of a column value in one of a set of value lists, rather than in one of a set of contiguous ranges of values. This is done by using PARTITION BY LIST(*expr*) where *expr* is a column value or an expression based on a column value and returning an integer value, and then defining each partition by means of a VALUES IN (*value_list*), where *value_list* is a comma-separated list of integers.

Note

In MySQL 5.7, it is possible to match against only a list of integers (and possibly NULL—see Section 17.2.7, “How MySQL Partitioning Handles NULL”) when partitioning by LIST.

However, other column types may be used in value lists when employing LIST COLUMN partitioning, which is described later in this section.

Unlike the case with partitions defined by range, list partitions do not need to be declared in any particular order. For more detailed syntactical information, see Section 13.1.14, “CREATE TABLE Syntax”.

For the examples that follow, we assume that the basic definition of the table to be partitioned is provided by the CREATE TABLE statement shown here:

```
CREATE TABLE employees (
    id INT NOT NULL,
    fname VARCHAR(30),
    lname VARCHAR(30),
    hired DATE NOT NULL DEFAULT '1970-01-01',
    separated DATE NOT NULL DEFAULT '9999-12-31',
    job_code INT,
    store_id INT
);
```

(This is the same table used as a basis for the examples in Section 17.2.1, “RANGE Partitioning”.)

Suppose that there are 20 video stores distributed among 4 franchises as shown in the following table.

Region	Store ID Numbers
North	3, 5, 6, 9, 17
East	1, 2, 10, 11, 19, 20
West	4, 12, 13, 14, 18
Central	7, 8, 15, 16

To partition this table in such a way that rows for stores belonging to the same region are stored in the same partition, you could use the CREATE TABLE statement shown here:

```
CREATE TABLE employees (
    id INT NOT NULL,
    fname VARCHAR(30),
    lname VARCHAR(30),
    hired DATE NOT NULL DEFAULT '1970-01-01',
    separated DATE NOT NULL DEFAULT '9999-12-31',
    job_code INT,
    store_id INT
)
PARTITION BY LIST(store_id) (
    PARTITION pNorth VALUES IN (3,5,6,9,17),
    PARTITION pEast VALUES IN (1,2,10,11,19,20),
    PARTITION pWest VALUES IN (4,12,13,14,18),
    PARTITION pCentral VALUES IN (7,8,15,16)
);
```

This makes it easy to add or drop employee records relating to specific regions to or from the table. For instance, suppose that all stores in the West region are sold to another company. In MySQL 5.7, all rows relating to employees working at stores in that region can be deleted with the query ALTER TABLE employees TRUNCATE PARTITION pWest, which can be executed much more efficiently than the equivalent DELETE statement DELETE FROM employees WHERE store_id IN (4,12,13,14,18);. (Using ALTER TABLE employees DROP PARTITION pWest would also delete all of these rows, but would also remove the partition pWest from the definition of the table; you would need to use an ALTER TABLE ... ADD PARTITION statement to restore the table's original partitioning scheme.)

As with RANGE partitioning, it is possible to combine LIST partitioning with partitioning by hash or key to produce a composite partitioning (subpartitioning). See Section 17.2.6, “Subpartitioning”.

Unlike the case with RANGE partitioning, there is no “catch-all” such as MAXVALUE; all expected values for the partitioning expression should be covered in PARTITION ... VALUES IN (...) clauses. An INSERT statement containing an unmatched partitioning column value fails with an error, as shown in this example:

```
mysql> CREATE TABLE h2 (
    ->   c1 INT,
    ->   c2 INT
    -> )
    -> PARTITION BY LIST(c1) (
    ->   PARTITION p0 VALUES IN (1, 4, 7),
    ->   PARTITION p1 VALUES IN (2, 5, 8)
    -> );
Query OK, 0 rows affected (0.11 sec)

mysql> INSERT INTO h2 VALUES (3, 5);
ERROR 1525 (HY000): Table has no partition for value 3
```

When inserting multiple rows using a single INSERT statement the behavior depends on whether the table uses a transactional storage engine. For an InnoDB table, the statement is considered a single transaction, so the presence of any unmatched values causes the statement to fail completely, and no rows are inserted. For a table using a nontransactional storage engine such as MyISAM, any rows coming before the row containing the unmatched value are inserted, but any coming after it are not.

You can cause this type of error to be ignored by using the IGNORE keyword. If you do so, rows containing unmatched partitioning column values are not inserted, but any rows with matching values *are* inserted, and no errors are reported:

```
mysql> TRUNCATE h2;
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM h2;
Empty set (0.00 sec)

mysql> INSERT IGNORE INTO h2 VALUES (2, 5), (6, 10), (7, 5), (3, 1), (1, 9);
Query OK, 3 rows affected (0.00 sec)
Records: 5  Duplicates: 2  Warnings: 0

mysql> SELECT * FROM h2;
+------+------+
| c1   | c2   |
+------+------+
|    7 |    5 |
|    1 |    9 |
|    2 |    5 |
+------+------+
3 rows in set (0.00 sec)
```

MySQL 5.7 provides support for `LIST COLUMNS` partitioning. This is a variant of `LIST` partitioning that enables you to use columns of types other than integer types for partitioning columns, as well as to use multiple columns as partitioning keys. For more information, see Section 17.2.3.2, "`LIST COLUMNS` partitioning".

17.2.3 COLUMNS Partitioning

The next two sections discuss *`COLUMNS` partitioning*, which are variants on `RANGE` and `LIST` partitioning. `COLUMNS` partitioning enables the use of multiple columns in partitioning keys. All of these columns are taken into account both for the purpose of placing rows in partitions and for the determination of which partitions are to be checked for matching rows in partition pruning.

In addition, both `RANGE COLUMNS` partitioning and `LIST COLUMNS` partitioning support the use of non-integer columns for defining value ranges or list members. The permitted data types are shown in the following list:

- All integer types: `TINYINT`, `SMALLINT`, `MEDIUMINT`, `INT` (`INTEGER`), and `BIGINT`. (This is the same as with partitioning by `RANGE` and `LIST`.)

  Other numeric data types (such as `DECIMAL` or `FLOAT`) are not supported as partitioning columns.

- `DATE` and `DATETIME`.

  Columns using other data types relating to dates or times are not supported as partitioning columns.

- The following string types: `CHAR`, `VARCHAR`, `BINARY`, and `VARBINARY`.

  `TEXT` and `BLOB` columns are not supported as partitioning columns.

The discussions of `RANGE COLUMNS` and `LIST COLUMNS` partitioning in the next two sections assume that you are already familiar with partitioning based on ranges and lists as supported in MySQL 5.1 and later; for more information about these, see Section 17.2.1, "RANGE Partitioning", and Section 17.2.2, "`LIST` Partitioning", respectively.

17.2.3.1 RANGE COLUMNS partitioning

Range columns partitioning is similar to range partitioning, but enables you to define partitions using ranges based on multiple column values. In addition, you can define the ranges using columns of types other than integer types.

RANGE COLUMNS partitioning differs significantly from RANGE partitioning in the following ways:

- RANGE COLUMNS does not accept expressions, only names of columns.
- RANGE COLUMNS accepts a list of one or more columns.

  RANGE COLUMNS partitions are based on comparisons between *tuples* (lists of column values) rather than comparisons between scalar values. Placement of rows in RANGE COLUMNS partitions is also based on comparisons between tuples; this is discussed further later in this section.
- RANGE COLUMNS partitioning columns are not restricted to integer columns; string, DATE and DATETIME columns can also be used as partitioning columns. (See Section 17.2.3, "COLUMNS Partitioning", for details.)

The basic syntax for creating a table partitioned by RANGE COLUMNS is shown here:

```
CREATE TABLE table_name
PARTITIONED BY RANGE COLUMNS(column_list) (
    PARTITION partition_name VALUES LESS THAN (value_list)[,
    PARTITION partition_name VALUES LESS THAN (value_list)][,
    ...]
)

column_list:
    column_name[, column_name][, ...]

value_list:
    value[, value][, ...]
```

Note

Not all CREATE TABLE options that can be used when creating partitioned tables are shown here. For complete information, see Section 13.1.14, "CREATE TABLE Syntax".

In the syntax just shown, *column_list* is a list of one or more columns (sometimes called a *partitioning column list*), and *value_list* is a list of values (that is, it is a *partition definition value list*). A *value_list* must be supplied for each partition definition, and each *value_list* must have the same number of values as the *column_list* has columns. Generally speaking, if you use *N* columns in the COLUMNS clause, then each VALUES LESS THAN clause must also be supplied with a list of *N* values.

The elements in the partitioning column list and in the value list defining each partition must occur in the same order. In addition, each element in the value list must be of the same data type as the corresponding element in the column list. However, the order of the column names in the partitioning column list and the value lists does not have to be the same as the order of the table column definitions in the main part of the CREATE TABLE statement. As with table partitioned by RANGE, you can use MAXVALUE to represent a value such that any legal value inserted into a given column is always less than this value. Here is an example of a CREATE TABLE statement that helps to illustrate all of these points:

```
mysql> CREATE TABLE rcx (
    ->     a INT,
    ->     b INT,
    ->     c CHAR(3),
    ->     d INT
    -> )
    -> PARTITION BY RANGE COLUMNS(a,d,c) (
    ->     PARTITION p0 VALUES LESS THAN (5,10,'ggg'),
    ->     PARTITION p1 VALUES LESS THAN (10,20,'mmmm'),
    ->     PARTITION p2 VALUES LESS THAN (15,30,'sss'),
    ->     PARTITION p3 VALUES LESS THAN (MAXVALUE,MAXVALUE,MAXVALUE)
    -> );
Query OK, 0 rows affected (0.15 sec)
```

Table rcx contains the columns a, b, c, d. The partitioning column list supplied to the COLUMNS clause uses 3 of these columns, in the order a, d, c. Each value list used to define a partition contains 3 values in the same order; that is, each value list tuple has the form (INT, INT, CHAR(3)), which corresponds to the data types used by columns a, d, and c (in that order).

Placement of rows into partitions is determined by comparing the tuple from a row to be inserted that matches the column list in the COLUMNS clause with the tuples used in the VALUES LESS THAN clauses to define partitions of the table. Because we are comparing tuples (that is, lists or sets of values) rather than scalar values, the semantics of VALUES LESS THAN as used with RANGE COLUMNS partitions differs somewhat from the case with simple RANGE partitions. In RANGE partitioning, a row generating an expression value that is equal to a limiting value in a VALUES LESS THAN is never placed in the corresponding partition; however, when using RANGE COLUMNS partitioning, it is sometimes possible for a row whose partitioning column list's first element is equal in value to the that of the first element in a VALUES LESS THAN value list to be placed in the corresponding partition.

Consider the RANGE partitioned table created by this statement:

```
CREATE TABLE r1 (
    a INT,
    b INT
)
PARTITION BY RANGE (a)  (
    PARTITION p0 VALUES LESS THAN (5),
    PARTITION p1 VALUES LESS THAN (MAXVALUE)
);
```

If we insert 3 rows into this table such that the column value for a is 5 for each row, all 3 rows are stored in partition p1 because the a column value is in each case not less than 5, as we can see by executing the proper query against the INFORMATION_SCHEMA.PARTITIONS table:

```
mysql> INSERT INTO r1 VALUES (5,10), (5,11), (5,12);
Query OK, 3 rows affected (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> SELECT PARTITION_NAME,TABLE_ROWS
    ->     FROM INFORMATION_SCHEMA.PARTITIONS
    ->     WHERE TABLE_NAME = 'r1';
+----------------+------------+
| PARTITION_NAME | TABLE_ROWS |
+----------------+------------+
| p0             |          0 |
| p1             |          3 |
+----------------+------------+
2 rows in set (0.00 sec)
```

Now consider a similar table rc1 that uses RANGE COLUMNS partitioning with both columns a and b referenced in the COLUMNS clause, created as shown here:

```
CREATE TABLE rc1 (
    a INT,
    b INT
)
PARTITION BY RANGE COLUMNS(a, b) (
    PARTITION p0 VALUES LESS THAN (5, 12),
    PARTITION p3 VALUES LESS THAN (MAXVALUE, MAXVALUE)
);
```

If we insert exactly the same rows into rc1 as we just inserted into r1, the distribution of the rows is quite different:

```
mysql> INSERT INTO rc1 VALUES (5,10), (5,11), (5,12);
Query OK, 3 rows affected (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> SELECT PARTITION_NAME,TABLE_ROWS
    ->     FROM INFORMATION_SCHEMA.PARTITIONS
    ->     WHERE TABLE_NAME = 'rc1';
+--------------+----------------+------------+
| TABLE_SCHEMA | PARTITION_NAME | TABLE_ROWS |
+--------------+----------------+------------+
| p            | p0             |          2 |
| p            | p1             |          1 |
+--------------+----------------+------------+
2 rows in set (0.00 sec)
```

This is because we are comparing rows rather than scalar values. We can compare the row values inserted with the limiting row value from the VALUES THAN LESS THAN clause used to define partition p0 in table rc1, like this:

```
mysql> SELECT (5,10) < (5,12), (5,11) < (5,12), (5,12) < (5,12);
+-----------------+-----------------+-----------------+
| (5,10) < (5,12) | (5,11) < (5,12) | (5,12) < (5,12) |
+-----------------+-----------------+-----------------+
|               1 |               1 |               0 |
+-----------------+-----------------+-----------------+
1 row in set (0.00 sec)
```

The 2 tuples (5,10) and (5,11) evaluate as less than (5,12), so they are stored in partition p0. Since 5 is not less than 5 and 12 is not less than 12, (5,12) is considered not less than (5,12), and is stored in partition p1.

The SELECT statement in the preceding example could also have been written using explicit row constructors, like this:

```
SELECT ROW(5,10) < ROW(5,12), ROW(5,11) < ROW(5,12), ROW(5,12) < ROW(5,12);
```

For more information about the use of row constructors in MySQL, see Section 13.2.10.5, “Row Subqueries”.

For a table partitioned by RANGE COLUMNS using only a single partitioning column, the storing of rows in partitions is the same as that of an equivalent table that is partitioned by RANGE. The following CREATE TABLE statement creates a table partitioned by RANGE COLUMNS using 1 partitioning column:

```
CREATE TABLE rx (
    a INT,
    b INT
)
PARTITION BY RANGE COLUMNS (a)  (
    PARTITION p0 VALUES LESS THAN (5),
    PARTITION p1 VALUES LESS THAN (MAXVALUE)
);
```

If we insert the rows (5,10), (5,11), and (5,12) into this table, we can see that their placement is the same as it is for the table r we created and populated earlier:

```
mysql> INSERT INTO rx VALUES (5,10), (5,11), (5,12);
```

```
Query OK, 3 rows affected (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> SELECT PARTITION_NAME,TABLE_ROWS
    ->     FROM INFORMATION_SCHEMA.PARTITIONS
    ->     WHERE TABLE_NAME = 'rx';
+--------------+----------------+------------+
| TABLE_SCHEMA | PARTITION_NAME | TABLE_ROWS |
+--------------+----------------+------------+
| p            | p0             |          0 |
| p            | p1             |          3 |
+--------------+----------------+------------+
2 rows in set (0.00 sec)
```

It is also possible to create tables partitioned by RANGE COLUMNS where limiting values for one or more columns are repeated in successive partition definitions. You can do this as long as the tuples of column values used to define the partitions are strictly increasing. For example, each of the following CREATE TABLE statements is valid:

```
CREATE TABLE rc2 (
    a INT,
    b INT
)
PARTITION BY RANGE COLUMNS(a,b) (
    PARTITION p0 VALUES LESS THAN (0,10),
    PARTITION p1 VALUES LESS THAN (10,20),
    PARTITION p2 VALUES LESS THAN (10,30),
    PARTITION p3 VALUES LESS THAN (MAXVALUE,MAXVALUE)
 );

CREATE TABLE rc3 (
    a INT,
    b INT
)
PARTITION BY RANGE COLUMNS(a,b) (
    PARTITION p0 VALUES LESS THAN (0,10),
    PARTITION p1 VALUES LESS THAN (10,20),
    PARTITION p2 VALUES LESS THAN (10,30),
    PARTITION p3 VALUES LESS THAN (10,35),
    PARTITION p4 VALUES LESS THAN (20,40),
    PARTITION p5 VALUES LESS THAN (MAXVALUE,MAXVALUE)
 );
```

The following statement also succeeds, even though it might appear at first glance that it would not, since the limiting value of column b is 25 for partition p0 and 20 for partition p1, and the limiting value of column c is 100 for partition p1 and 50 for partition p2:

```
CREATE TABLE rc4 (
    a INT,
    b INT,
    c INT
)
PARTITION BY RANGE COLUMNS(a,b,c) (
    PARTITION p0 VALUES LESS THAN (0,25,50),
    PARTITION p1 VALUES LESS THAN (10,20,100),
    PARTITION p2 VALUES LESS THAN (10,30,50)
    PARTITION p3 VALUES LESS THAN (MAXVALUE,MAXVALUE,MAXVALUE)
 );
```

When designing tables partitioned by RANGE COLUMNS, you can always test successive partition definitions by comparing the desired tuples using the mysql client, like this:

```
mysql> SELECT (0,25,50) < (10,20,100), (10,20,100) < (10,30,50);
+-------------------------+--------------------------+
| (0,25,50) < (10,20,100) | (10,20,100) < (10,30,50) |
+-------------------------+--------------------------+
|                       1 |                        1 |
+-------------------------+--------------------------+
1 row in set (0.00 sec)
```

If a CREATE TABLE statement contains partition definitions that are not in strictly increasing order, it fails with an error, as shown in this example:

```
mysql> CREATE TABLE rcf (
    ->     a INT,
    ->     b INT,
    ->     c INT
    -> )
    -> PARTITION BY RANGE COLUMNS(a,b,c) (
    ->     PARTITION p0 VALUES LESS THAN (0,25,50),
    ->     PARTITION p1 VALUES LESS THAN (20,20,100),
    ->     PARTITION p2 VALUES LESS THAN (10,30,50),
    ->     PARTITION p3 VALUES LESS THAN (MAXVALUE,MAXVALUE,MAXVALUE)
    ->  );
ERROR 1493 (HY000): VALUES LESS THAN value must be strictly increasing for each partition
```

When you get such an error, you can deduce which partition definitions are invalid by making “less than” comparisons between their column lists. In this case, the problem is with the definition of partition p2 because the tuple used to define it is not less than the tuple used to define partition p3, as shown here:

```
mysql> SELECT (0,25,50) < (20,20,100), (20,20,100) < (10,30,50);
+-------------------------+--------------------------+
| (0,25,50) < (20,20,100) | (20,20,100) < (10,30,50) |
+-------------------------+--------------------------+
|                       1 |                        0 |
+-------------------------+--------------------------+
1 row in set (0.00 sec)
```

It is also possible for MAXVALUE to appear for the same column in more than one VALUES LESS THAN clause when using RANGE COLUMNS. However, the limiting values for individual columns in successive partition definitions should otherwise be increasing, there should be no more than one partition defined where MAXVALUE is used as the upper limit for all column values, and this partition definition should appear last in the list of PARTITION ... VALUES LESS THAN clauses. In addition, you cannot use MAXVALUE as the limiting value for the first column in more than one partition definition.

As stated previously, it is also possible with RANGE COLUMNS partitioning to use non-integer columns as partitioning columns. (See Section 17.2.3, “COLUMNS Partitioning”, for a complete listing of these.) Consider a table named employees (which is not partitioned), created using the following statement:

```
CREATE TABLE employees (
    id INT NOT NULL,
    fname VARCHAR(30),
    lname VARCHAR(30),
    hired DATE NOT NULL DEFAULT '1970-01-01',
    separated DATE NOT NULL DEFAULT '9999-12-31',
    job_code INT NOT NULL,
    store_id INT NOT NULL
);
```

Using RANGE COLUMNS partitioning, you can create a version of this table that stores each row in one of four partitions based on the employee's last name, like this:

```
CREATE TABLE employees_by_lname (
    id INT NOT NULL,
    fname VARCHAR(30),
    lname VARCHAR(30),
    hired DATE NOT NULL DEFAULT '1970-01-01',
    separated DATE NOT NULL DEFAULT '9999-12-31',
    job_code INT NOT NULL,
    store_id INT NOT NULL
)
PARTITION BY RANGE COLUMNS (lname)  (
    PARTITION p0 VALUES LESS THAN ('g'),
    PARTITION p1 VALUES LESS THAN ('m'),
    PARTITION p2 VALUES LESS THAN ('t'),
    PARTITION p3 VALUES LESS THAN (MAXVALUE)
);
```

Alternatively, you could cause the `employees` table as created previously to be partitioned using this scheme by executing the following `ALTER TABLE` statement:

```
ALTER TABLE employees PARTITION BY RANGE COLUMNS (lname)  (
    PARTITION p0 VALUES LESS THAN ('g'),
    PARTITION p1 VALUES LESS THAN ('m'),
    PARTITION p2 VALUES LESS THAN ('t'),
    PARTITION p3 VALUES LESS THAN (MAXVALUE)
);
```

Note

Because different character sets and collations have different sort orders, the character sets and collations in use may effect which partition of a table partitioned by `RANGE COLUMNS` a given row is stored in when using string columns as partitioning columns. In addition, changing the character set or collation for a given database, table, or column after such a table is created may cause changes in how rows are distributed. For example, when using a case-sensitive collation, `'and'` sorts before `'Andersen'`, but when using a collation that is case insensitive, the reverse is true.

For information about how MySQL handles character sets and collations, see Section 10.1, “Character Set Support”.

Similarly, you can cause the `employees` table to be partitioned in such a way that each row is stored in one of several partitions based on the decade in which the corresponding employee was hired using the `ALTER TABLE` statement shown here:

```
ALTER TABLE employees PARTITION BY RANGE COLUMNS (hired)  (
    PARTITION p0 VALUES LESS THAN ('1970-01-01'),
    PARTITION p1 VALUES LESS THAN ('1980-01-01'),
    PARTITION p2 VALUES LESS THAN ('1990-01-01'),
    PARTITION p3 VALUES LESS THAN ('2000-01-01'),
    PARTITION p4 VALUES LESS THAN ('2010-01-01'),
    PARTITION p5 VALUES LESS THAN (MAXVALUE)
);
```

See Section 13.1.14, “`CREATE TABLE` Syntax”, for additional information about `PARTITION BY RANGE COLUMNS` syntax.

17.2.3.2 `LIST COLUMNS` partitioning

MySQL 5.7 provides support for `LIST COLUMNS` partitioning. This is a variant of `LIST` partitioning that enables the use of multiple columns as partition keys, and for columns of data types other than integer types to be used as partitioning columns; you can use string types, DATE, and DATETIME columns. (For more information about permitted data types for COLUMNS partitioning columns, see Section 17.2.3, “COLUMNS Partitioning”.)

Suppose that you have a business that has customers in 12 cities which, for sales and marketing purposes, you organize into 4 regions of 3 cities each as shown in the following table:

Region	Cities
1	Oskarshamn, Högsby, Mönsterås
2	Vimmerby, Hultsfred, Västervik
3	Nässjö, Eksjö, Vetlanda
4	Uppvidinge, Alvesta, Växjo

With LIST COLUMNS partitioning, you can create a table for customer data that assigns a row to any of 4 partitions corresponding to these regions based on the name of the city where a customer resides, as shown here:

```
CREATE TABLE customers_1 (
    first_name VARCHAR(25),
    last_name VARCHAR(25),
    street_1 VARCHAR(30),
    street_2 VARCHAR(30),
    city VARCHAR(15),
    renewal DATE
)
PARTITION BY LIST COLUMNS(city) (
    PARTITION pRegion_1 VALUES IN('Oskarshamn', 'Högsby', 'Mönsterås'),
    PARTITION pRegion_2 VALUES IN('Vimmerby', 'Hultsfred', 'Västervik'),
    PARTITION pRegion_3 VALUES IN('Nässjö', 'Eksjö', 'Vetlanda'),
    PARTITION pRegion_4 VALUES IN('Uppvidinge', 'Alvesta', 'Växjo')
);
```

As with partitioning by RANGE COLUMNS, you do not need to use expressions in the COLUMNS() clause to convert column values into integers. (In fact, the use of expressions other than column names is not permitted with COLUMNS().)

It is also possible to use DATE and DATETIME columns, as shown in the following example that uses the same name and columns as the customers_1 table shown previously, but employs LIST COLUMNS partitioning based on the renewal column to store rows in one of 4 partitions depending on the week in February 2010 the customer's account is scheduled to renew:

```
CREATE TABLE customers_2 (
    first_name VARCHAR(25),
    last_name VARCHAR(25),
    street_1 VARCHAR(30),
    street_2 VARCHAR(30),
    city VARCHAR(15),
    renewal DATE
)
PARTITION BY LIST COLUMNS(renewal) (
    PARTITION pWeek_1 VALUES IN('2010-02-01', '2010-02-02', '2010-02-03',
        '2010-02-04', '2010-02-05', '2010-02-06', '2010-02-07'),
    PARTITION pWeek_2 VALUES IN('2010-02-08', '2010-02-09', '2010-02-10',
        '2010-02-11', '2010-02-12', '2010-02-13', '2010-02-14'),
    PARTITION pWeek_3 VALUES IN('2010-02-15', '2010-02-16', '2010-02-17',
        '2010-02-18', '2010-02-19', '2010-02-20', '2010-02-21'),
    PARTITION pWeek_4 VALUES IN('2010-02-22', '2010-02-23', '2010-02-24',
        '2010-02-25', '2010-02-26', '2010-02-27', '2010-02-28')
);
```

This works, but becomes cumbersome to define and maintain if the number of dates involved grows very large; in such cases, it is usually more practical to employ RANGE or RANGE COLUMNS partitioning instead. In this case, since the column we wish to use as the partitioning key is a DATE column, we use RANGE COLUMNS partitioning, as shown here:

```
CREATE TABLE customers_3 (
    first_name VARCHAR(25),
    last_name VARCHAR(25),
    street_1 VARCHAR(30),
    street_2 VARCHAR(30),
    city VARCHAR(15),
    renewal DATE
)
PARTITION BY RANGE COLUMNS(renewal) (
    PARTITION pWeek_1 VALUES LESS THAN('2010-02-09'),
    PARTITION pWeek_2 VALUES LESS THAN('2010-02-15'),
    PARTITION pWeek_3 VALUES LESS THAN('2010-02-22'),
    PARTITION pWeek_4 VALUES LESS THAN('2010-03-01')
);
```

See Section 17.2.3.1, “RANGE COLUMNS partitioning”, for more information.

In addition (as with RANGE COLUMNS partitioning), you can use multiple columns in the COLUMNS() clause.

See Section 13.1.14, “CREATE TABLE Syntax”, for additional information about PARTITION BY LIST COLUMNS() syntax.

17.2.4 HASH Partitioning

Partitioning by HASH is used primarily to ensure an even distribution of data among a predetermined number of partitions. With range or list partitioning, you must specify explicitly into which partition a given column value or set of column values is to be stored; with hash partitioning, MySQL takes care of this for you, and you need only specify a column value or expression based on a column value to be hashed and the number of partitions into which the partitioned table is to be divided.

To partition a table using HASH partitioning, it is necessary to append to the CREATE TABLE statement a PARTITION BY HASH (*expr*) clause, where *expr* is an expression that returns an integer. This can simply be the name of a column whose type is one of MySQL's integer types. In addition, you will most likely want to follow this with a PARTITIONS *num* clause, where *num* is a positive integer representing the number of partitions into which the table is to be divided.

For example, the following statement creates a table that uses hashing on the store_id column and is divided into 4 partitions:

```
CREATE TABLE employees (
    id INT NOT NULL,
    fname VARCHAR(30),
    lname VARCHAR(30),
    hired DATE NOT NULL DEFAULT '1970-01-01',
    separated DATE NOT NULL DEFAULT '9999-12-31',
    job_code INT,
    store_id INT
)
PARTITION BY HASH(store_id)
```

```
PARTITIONS 4;
```

If you do not include a PARTITIONS clause, the number of partitions defaults to 1.

Using the PARTITIONS keyword without a number following it results in a syntax error.

You can also use an SQL expression that returns an integer for *expr*. For instance, you might want to partition based on the year in which an employee was hired. This can be done as shown here:

```
CREATE TABLE employees (
    id INT NOT NULL,
    fname VARCHAR(30),
    lname VARCHAR(30),
    hired DATE NOT NULL DEFAULT '1970-01-01',
    separated DATE NOT NULL DEFAULT '9999-12-31',
    job_code INT,
    store_id INT
)
PARTITION BY HASH( YEAR(hired) )
PARTITIONS 4;
```

expr must return a nonconstant, nonrandom integer value (in other words, it should be varying but deterministic), and must not contain any prohibited constructs as described in Section 17.6, “Restrictions and Limitations on Partitioning”. You should also keep in mind that this expression is evaluated each time a row is inserted or updated (or possibly deleted); this means that very complex expressions may give rise to performance issues, particularly when performing operations (such as batch inserts) that affect a great many rows at one time.

The most efficient hashing function is one which operates upon a single table column and whose value increases or decreases consistently with the column value, as this allows for “pruning” on ranges of partitions. That is, the more closely that the expression varies with the value of the column on which it is based, the more efficiently MySQL can use the expression for hash partitioning.

For example, where date_col is a column of type DATE, then the expression TO_DAYS(date_col) is said to vary directly with the value of date_col, because for every change in the value of date_col, the value of the expression changes in a consistent manner. The variance of the expression YEAR(date_col) with respect to date_col is not quite as direct as that of TO_DAYS(date_col), because not every possible change in date_col produces an equivalent change in YEAR(date_col). Even so, YEAR(date_col) is a good candidate for a hashing function, because it varies directly with a portion of date_col and there is no possible change in date_col that produces a disproportionate change in YEAR(date_col).

By way of contrast, suppose that you have a column named int_col whose type is INT. Now consider the expression POW(5-int_col,3) + 6. This would be a poor choice for a hashing function because a change in the value of int_col is not guaranteed to produce a proportional change in the value of the expression. Changing the value of int_col by a given amount can produce by widely different changes in the value of the expression. For example, changing int_col from 5 to 6 produces a change of -1 in the value of the expression, but changing the value of int_col from 6 to 7 produces a change of -7 in the expression value.

In other words, the more closely the graph of the column value *versus* the value of the expression follows a straight line as traced by the equation y=*c*x where *c* is some nonzero constant, the better the expression is suited to hashing. This has to do with the fact that the more nonlinear an expression is, the more uneven the distribution of data among the partitions it tends to produce.

In theory, pruning is also possible for expressions involving more than one column value, but determining which of such expressions are suitable can be quite difficult and time-consuming. For this reason, the use of hashing expressions involving multiple columns is not particularly recommended.

When PARTITION BY HASH is used, MySQL determines which partition of *num* partitions to use based on the modulus of the result of the user function. In other words, for an expression *expr*, the partition in which the record is stored is partition number *N*, where *N* = MOD(*expr*, *num*). Suppose that table t1 is defined as follows, so that it has 4 partitions:

```
CREATE TABLE t1 (col1 INT, col2 CHAR(5), col3 DATE)
    PARTITION BY HASH( YEAR(col3) )
    PARTITIONS 4;
```

If you insert a record into t1 whose col3 value is '2005-09-15', then the partition in which it is stored is determined as follows:

```
MOD(YEAR('2005-09-01'),4)
=  MOD(2005,4)
=  1
```

MySQL 5.7 also supports a variant of HASH partitioning known as *linear hashing* which employs a more complex algorithm for determining the placement of new rows inserted into the partitioned table. See Section 17.2.4.1, “LINEAR HASH Partitioning”, for a description of this algorithm.

The user function is evaluated each time a record is inserted or updated. It may also—depending on the circumstances—be evaluated when records are deleted.

Note

If a table to be partitioned has a UNIQUE key, then any columns supplied as arguments to the HASH user function or to the KEY's *column_list* must be part of that key.

17.2.4.1 LINEAR HASH Partitioning

MySQL also supports linear hashing, which differs from regular hashing in that linear hashing utilizes a linear powers-of-two algorithm whereas regular hashing employs the modulus of the hashing function's value.

Syntactically, the only difference between linear-hash partitioning and regular hashing is the addition of the LINEAR keyword in the PARTITION BY clause, as shown here:

```
CREATE TABLE employees (
    id INT NOT NULL,
    fname VARCHAR(30),
    lname VARCHAR(30),
    hired DATE NOT NULL DEFAULT '1970-01-01',
    separated DATE NOT NULL DEFAULT '9999-12-31',
    job_code INT,
    store_id INT
)
PARTITION BY LINEAR HASH( YEAR(hired) )
PARTITIONS 4;
```

Given an expression *expr*, the partition in which the record is stored when linear hashing is used is partition number *N* from among *num* partitions, where *N* is derived according to the following algorithm:

1. Find the next power of 2 greater than *num*. We call this value *V*; it can be calculated as:

   ```
   V = POWER(2, CEILING(LOG(2, num)))
   ```

(Suppose that *num* is 13. Then `LOG(2,13)` is 3.7004397181411. `CEILING(3.7004397181411)` is 4, and *V* = `POWER(2,4)`, which is 16.)

2. Set *N* = *F*(*column_list*) & (*V* - 1).

3. While *N* >= *num*:

   - Set *V* = CEIL(*V* / 2)

   - Set *N* = *N* & (*V* - 1)

Suppose that the table `t1`, using linear hash partitioning and having 6 partitions, is created using this statement:

```
CREATE TABLE t1 (col1 INT, col2 CHAR(5), col3 DATE)
    PARTITION BY LINEAR HASH( YEAR(col3) )
    PARTITIONS 6;
```

Now assume that you want to insert two records into `t1` having the `col3` column values `'2003-04-14'` and `'1998-10-19'`. The partition number for the first of these is determined as follows:

```
V = POWER(2, CEILING( LOG(2,6) )) = 8
N = YEAR('2003-04-14') & (8 - 1)
   = 2003 & 7
   = 3

(3 >= 6 is FALSE: record stored in partition #3)
```

The number of the partition where the second record is stored is calculated as shown here:

```
V = 8
N = YEAR('1998-10-19') & (8-1)
  = 1998 & 7
  = 6

(6 >= 6 is TRUE: additional step required)

N = 6 & CEILING(8 / 2)
  = 6 & 3
  = 2

(2 >= 6 is FALSE: record stored in partition #2)
```

The advantage in partitioning by linear hash is that the adding, dropping, merging, and splitting of partitions is made much faster, which can be beneficial when dealing with tables containing extremely large amounts (terabytes) of data. The disadvantage is that data is less likely to be evenly distributed between partitions as compared with the distribution obtained using regular hash partitioning.

17.2.5 KEY Partitioning

Partitioning by key is similar to partitioning by hash, except that where hash partitioning employs a user-defined expression, the hashing function for key partitioning is supplied by the MySQL server. This internal hashing function is based on the same algorithm as `PASSWORD()`.

The syntax rules for `CREATE TABLE ... PARTITION BY KEY` are similar to those for creating a table that is partitioned by hash. The major differences are listed here:

- KEY is used rather than HASH.
- KEY takes only a list of zero or more column names. Any columns used as the partitioning key must comprise part or all of the table's primary key, if the table has one. Where no column name is specified as the partitioning key, the table's primary key is used, if there is one. For example, the following CREATE TABLE statement is valid in MySQL 5.7:

  ```
  CREATE TABLE k1 (
      id INT NOT NULL PRIMARY KEY,
      name VARCHAR(20)
  )
  PARTITION BY KEY()
  PARTITIONS 2;
  ```

  If there is no primary key but there is a unique key, then the unique key is used for the partitioning key:

  ```
  CREATE TABLE k1 (
      id INT NOT NULL,
      name VARCHAR(20),
      UNIQUE KEY (id)
  )
  PARTITION BY KEY()
  PARTITIONS 2;
  ```

  However, if the unique key column were not defined as NOT NULL, then the previous statement would fail.

  In both of these cases, the partitioning key is the id column, even though it is not shown in the output of SHOW CREATE TABLE or in the PARTITION_EXPRESSION column of the INFORMATION_SCHEMA.PARTITIONS table.

  Unlike the case with other partitioning types, columns used for partitioning by KEY are not restricted to integer or NULL values. For example, the following CREATE TABLE statement is valid:

  ```
  CREATE TABLE tm1 (
      s1 CHAR(32) PRIMARY KEY
  )
  PARTITION BY KEY(s1)
  PARTITIONS 10;
  ```

  The preceding statement would *not* be valid, were a different partitioning type to be specified. (In this case, simply using PARTITION BY KEY() would also be valid and have the same effect as PARTITION BY KEY(s1), since s1 is the table's primary key.)

  For additional information about this issue, see Section 17.6, “Restrictions and Limitations on Partitioning”.

  Important

  For a key-partitioned table, you cannot execute an ALTER TABLE DROP PRIMARY KEY, as doing so generates the error ERROR 1466 (HY000): Field in list of fields for partition function not found in table.

It is also possible to partition a table by linear key. Here is a simple example:

```
CREATE TABLE tk (
    col1 INT NOT NULL,
    col2 CHAR(5),
    col3 DATE
)
PARTITION BY LINEAR KEY (col1)
PARTITIONS 3;
```

Using LINEAR has the same effect on KEY partitioning as it does on HASH partitioning, with the partition number being derived using a powers-of-two algorithm rather than modulo arithmetic. See Section 17.2.4.1, "LINEAR HASH Partitioning", for a description of this algorithm and its implications.

17.2.6 Subpartitioning

Subpartitioning—also known as *composite partitioning*—is the further division of each partition in a partitioned table. Consider the following CREATE TABLE statement:

```
CREATE TABLE ts (id INT, purchased DATE)
    PARTITION BY RANGE( YEAR(purchased) )
    SUBPARTITION BY HASH( TO_DAYS(purchased) )
    SUBPARTITIONS 2 (
        PARTITION p0 VALUES LESS THAN (1990),
        PARTITION p1 VALUES LESS THAN (2000),
        PARTITION p2 VALUES LESS THAN MAXVALUE
    );
```

Table ts has 3 RANGE partitions. Each of these partitions—p0, p1, and p2—is further divided into 2 subpartitions. In effect, the entire table is divided into 3 * 2 = 6 partitions. However, due to the action of the PARTITION BY RANGE clause, the first 2 of these store only those records with a value less than 1990 in the purchased column.

In MySQL 5.7, it is possible to subpartition tables that are partitioned by RANGE or LIST. Subpartitions may use either HASH or KEY partitioning. This is also known as *composite partitioning*.

Note

SUBPARTITION BY HASH and SUBPARTITION BY KEY generally follow the same syntax rules as PARTITION BY HASH and PARTITION BY KEY, respectively. An exception to this is that SUBPARTITION BY KEY (unlike PARTITION BY KEY) does not currently support a default column, so the column used for this purpose must be specified, even if the table has an explicit primary key. This is a known issue which we are working to address; see Issues with subpartitions, for more information and an example.

It is also possible to define subpartitions explicitly using SUBPARTITION clauses to specify options for individual subpartitions. For example, a more verbose fashion of creating the same table ts as shown in the previous example would be:

```
CREATE TABLE ts (id INT, purchased DATE)
    PARTITION BY RANGE( YEAR(purchased) )
    SUBPARTITION BY HASH( TO_DAYS(purchased) ) (
        PARTITION p0 VALUES LESS THAN (1990) (
            SUBPARTITION s0,
            SUBPARTITION s1
        ),
        PARTITION p1 VALUES LESS THAN (2000) (
            SUBPARTITION s2,
            SUBPARTITION s3
        ),
        PARTITION p2 VALUES LESS THAN MAXVALUE (
            SUBPARTITION s4,
            SUBPARTITION s5
        )
    );
```

Some syntactical items of note are listed here:

- Each partition must have the same number of subpartitions.
- If you explicitly define any subpartitions using SUBPARTITION on any partition of a partitioned table, you must define them all. In other words, the following statement will fail:

  ```
  CREATE TABLE ts (id INT, purchased DATE)
      PARTITION BY RANGE( YEAR(purchased) )
      SUBPARTITION BY HASH( TO_DAYS(purchased) ) (
          PARTITION p0 VALUES LESS THAN (1990) (
              SUBPARTITION s0,
              SUBPARTITION s1
          ),
          PARTITION p1 VALUES LESS THAN (2000),
          PARTITION p2 VALUES LESS THAN MAXVALUE (
              SUBPARTITION s2,
              SUBPARTITION s3
          )
      );
  ```

  This statement would still fail even if it included a SUBPARTITIONS 2 clause.

- Each SUBPARTITION clause must include (at a minimum) a name for the subpartition. Otherwise, you may set any desired option for the subpartition or allow it to assume its default setting for that option.
- Subpartition names must be unique across the entire table. For example, the following CREATE TABLE statement is valid in MySQL 5.7:

  ```
  CREATE TABLE ts (id INT, purchased DATE)
      PARTITION BY RANGE( YEAR(purchased) )
      SUBPARTITION BY HASH( TO_DAYS(purchased) ) (
          PARTITION p0 VALUES LESS THAN (1990) (
              SUBPARTITION s0,
              SUBPARTITION s1
          ),
          PARTITION p1 VALUES LESS THAN (2000) (
              SUBPARTITION s2,
              SUBPARTITION s3
          ),
          PARTITION p2 VALUES LESS THAN MAXVALUE (
              SUBPARTITION s4,
              SUBPARTITION s5
          )
      );
  ```

Subpartitions can be used with especially large tables to distribute data and indexes across many disks. Suppose that you have 6 disks mounted as /disk0, /disk1, /disk2, and so on. Now consider the following example:

```
CREATE TABLE ts (id INT, purchased DATE)
    PARTITION BY RANGE( YEAR(purchased) )
    SUBPARTITION BY HASH( TO_DAYS(purchased) ) (
        PARTITION p0 VALUES LESS THAN (1990) (
```

```
            SUBPARTITION s0
                DATA DIRECTORY = '/disk0/data'
                INDEX DIRECTORY = '/disk0/idx',
            SUBPARTITION s1
                DATA DIRECTORY = '/disk1/data'
                INDEX DIRECTORY = '/disk1/idx'
        ),
        PARTITION p1 VALUES LESS THAN (2000) (
            SUBPARTITION s2
                DATA DIRECTORY = '/disk2/data'
                INDEX DIRECTORY = '/disk2/idx',
            SUBPARTITION s3
                DATA DIRECTORY = '/disk3/data'
                INDEX DIRECTORY = '/disk3/idx'
        ),
        PARTITION p2 VALUES LESS THAN MAXVALUE (
            SUBPARTITION s4
                DATA DIRECTORY = '/disk4/data'
                INDEX DIRECTORY = '/disk4/idx',
            SUBPARTITION s5
                DATA DIRECTORY = '/disk5/data'
                INDEX DIRECTORY = '/disk5/idx'
        )
    );
```

In this case, a separate disk is used for the data and for the indexes of each `RANGE`. Many other variations are possible; another example might be:

```
CREATE TABLE ts (id INT, purchased DATE)
    PARTITION BY RANGE(YEAR(purchased))
    SUBPARTITION BY HASH( TO_DAYS(purchased) ) (
        PARTITION p0 VALUES LESS THAN (1990) (
            SUBPARTITION s0a
                DATA DIRECTORY = '/disk0'
                INDEX DIRECTORY = '/disk1',
            SUBPARTITION s0b
                DATA DIRECTORY = '/disk2'
                INDEX DIRECTORY = '/disk3'
        ),
        PARTITION p1 VALUES LESS THAN (2000) (
            SUBPARTITION s1a
                DATA DIRECTORY = '/disk4/data'
                INDEX DIRECTORY = '/disk4/idx',
            SUBPARTITION s1b
                DATA DIRECTORY = '/disk5/data'
                INDEX DIRECTORY = '/disk5/idx'
        ),
        PARTITION p2 VALUES LESS THAN MAXVALUE (
            SUBPARTITION s2a,
            SUBPARTITION s2b
        )
    );
```

Here, the storage is as follows:

- Rows with `purchased` dates from before 1990 take up a vast amount of space, so are split up 4 ways, with a separate disk dedicated to the data and to the indexes for each of the two subpartitions (`s0a` and `s0b`) making up partition `p0`. In other words:
    - The data for subpartition `s0a` is stored on `/disk0`.
    - The indexes for subpartition `s0a` are stored on `/disk1`.
    - The data for subpartition `s0b` is stored on `/disk2`.

  - The indexes for subpartition s0b are stored on /disk3.

- Rows containing dates ranging from 1990 to 1999 (partition p1) do not require as much room as those from before 1990. These are split between 2 disks (/disk4 and /disk5) rather than 4 disks as with the legacy records stored in p0:

  - Data and indexes belonging to p1's first subpartition (s1a) are stored on /disk4—the data in /disk4/data, and the indexes in /disk4/idx.

  - Data and indexes belonging to p1's second subpartition (s1b) are stored on /disk5—the data in /disk5/data, and the indexes in /disk5/idx.

- Rows reflecting dates from the year 2000 to the present (partition p2) do not take up as much space as required by either of the two previous ranges. Currently, it is sufficient to store all of these in the default location.

  In future, when the number of purchases for the decade beginning with the year 2000 grows to a point where the default location no longer provides sufficient space, the corresponding rows can be moved using an ALTER TABLE ... REORGANIZE PARTITION statement. See Section 17.3, “Partition Management”, for an explanation of how this can be done.

The DATA DIRECTORY and INDEX DIRECTORY options are not permitted in partition definitions when the NO_DIR_IN_CREATE server SQL mode is in effect. In MySQL 5.7, these options are also not permitted when defining subpartitions (Bug #42954).

17.2.7 How MySQL Partitioning Handles NULL

Partitioning in MySQL does nothing to disallow NULL as the value of a partitioning expression, whether it is a column value or the value of a user-supplied expression. Even though it is permitted to use NULL as the value of an expression that must otherwise yield an integer, it is important to keep in mind that NULL is not a number. MySQL's partitioning implementation treats NULL as being less than any non-NULL value, just as ORDER BY does.

This means that treatment of NULL varies between partitioning of different types, and may produce behavior which you do not expect if you are not prepared for it. This being the case, we discuss in this section how each MySQL partitioning type handles NULL values when determining the partition in which a row should be stored, and provide examples for each.

Handling of NULL with RANGE partitioning. If you insert a row into a table partitioned by RANGE such that the column value used to determine the partition is NULL, the row is inserted into the lowest partition. Consider these two tables in a database named p, created as follows:

```
mysql> CREATE TABLE t1 (
    ->     c1 INT,
    ->     c2 VARCHAR(20)
    -> )
    -> PARTITION BY RANGE(c1) (
    ->     PARTITION p0 VALUES LESS THAN (0),
    ->     PARTITION p1 VALUES LESS THAN (10),
    ->     PARTITION p2 VALUES LESS THAN MAXVALUE
    -> );
Query OK, 0 rows affected (0.09 sec)

mysql> CREATE TABLE t2 (
    ->     c1 INT,
    ->     c2 VARCHAR(20)
    -> )
    -> PARTITION BY RANGE(c1) (
    ->     PARTITION p0 VALUES LESS THAN (-5),
    ->     PARTITION p1 VALUES LESS THAN (0),
    ->     PARTITION p2 VALUES LESS THAN (10),
    ->     PARTITION p3 VALUES LESS THAN MAXVALUE
    -> );
Query OK, 0 rows affected (0.09 sec)
```

You can see the partitions created by these two CREATE TABLE statements using the following query against the PARTITIONS table in the INFORMATION_SCHEMA database:

```
mysql> SELECT TABLE_NAME, PARTITION_NAME, TABLE_ROWS, AVG_ROW_LENGTH, DATA_LENGTH
     >   FROM INFORMATION_SCHEMA.PARTITIONS
     >   WHERE TABLE_SCHEMA = 'p' AND TABLE_NAME LIKE 't_';
+------------+----------------+------------+----------------+-------------+
| TABLE_NAME | PARTITION_NAME | TABLE_ROWS | AVG_ROW_LENGTH | DATA_LENGTH |
+------------+----------------+------------+----------------+-------------+
| t1         | p0             |          0 |              0 |           0 |
| t1         | p1             |          0 |              0 |           0 |
| t1         | p2             |          0 |              0 |           0 |
| t2         | p0             |          0 |              0 |           0 |
| t2         | p1             |          0 |              0 |           0 |
| t2         | p2             |          0 |              0 |           0 |
| t2         | p3             |          0 |              0 |           0 |
+------------+----------------+------------+----------------+-------------+
7 rows in set (0.00 sec)
```

(For more information about this table, see Section 19.14, “The INFORMATION_SCHEMA PARTITIONS Table”.) Now let us populate each of these tables with a single row containing a NULL in the column used as the partitioning key, and verify that the rows were inserted using a pair of SELECT statements:

```
mysql> INSERT INTO t1 VALUES (NULL, 'mothra');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO t2 VALUES (NULL, 'mothra');
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM t1;
+------+--------+
| id   | name   |
+------+--------+
| NULL | mothra |
+------+--------+
1 row in set (0.00 sec)

mysql> SELECT * FROM t2;
+------+--------+
| id   | name   |
+------+--------+
| NULL | mothra |
+------+--------+
1 row in set (0.00 sec)
```

You can see which partitions are used to store the inserted rows by rerunning the previous query against INFORMATION_SCHEMA.PARTITIONS and inspecting the output:

```
mysql> SELECT TABLE_NAME, PARTITION_NAME, TABLE_ROWS, AVG_ROW_LENGTH, DATA_LENGTH
     >   FROM INFORMATION_SCHEMA.PARTITIONS
     >   WHERE TABLE_SCHEMA = 'p' AND TABLE_NAME LIKE 't_';
+------------+----------------+------------+----------------+-------------+
| TABLE_NAME | PARTITION_NAME | TABLE_ROWS | AVG_ROW_LENGTH | DATA_LENGTH |
+------------+----------------+------------+----------------+-------------+
| t1         | p0             |          1 |             20 |          20 |
| t1         | p1             |          0 |              0 |           0 |
| t1         | p2             |          0 |              0 |           0 |
| t2         | p0             |          1 |             20 |          20 |
| t2         | p1             |          0 |              0 |           0 |
| t2         | p2             |          0 |              0 |           0 |
| t2         | p3             |          0 |              0 |           0 |
+------------+----------------+------------+----------------+-------------+
7 rows in set (0.01 sec)
```

You can also demonstrate that these rows were stored in the lowest partition of each table by dropping these partitions, and then re-running the SELECT statements:

```
mysql> ALTER TABLE t1 DROP PARTITION p0;
Query OK, 0 rows affected (0.16 sec)

mysql> ALTER TABLE t2 DROP PARTITION p0;
Query OK, 0 rows affected (0.16 sec)

mysql> SELECT * FROM t1;
Empty set (0.00 sec)

mysql> SELECT * FROM t2;
Empty set (0.00 sec)
```

(For more information on ALTER TABLE ... DROP PARTITION, see Section 13.1.6, “ALTER TABLE Syntax”.)

NULL is also treated in this way for partitioning expressions that use SQL functions. Suppose that we define a table using a CREATE TABLE statement such as this one:

```
CREATE TABLE tndate (
    id INT,
    dt DATE
)
PARTITION BY RANGE( YEAR(dt) ) (
    PARTITION p0 VALUES LESS THAN (1990),
    PARTITION p1 VALUES LESS THAN (2000),
    PARTITION p2 VALUES LESS THAN MAXVALUE
);
```

As with other MySQL functions, YEAR(NULL) returns NULL. A row with a dt column value of NULL is treated as though the partitioning expression evaluated to a value less than any other value, and so is inserted into partition p0.

Handling of NULL with LIST partitioning. A table that is partitioned by LIST admits NULL values if and only if one of its partitions is defined using that value-list that contains NULL. The converse of this is that a table partitioned by LIST which does not explicitly use NULL in a value list rejects rows resulting in a NULL value for the partitioning expression, as shown in this example:

```
mysql> CREATE TABLE ts1 (
    ->     c1 INT,
    ->     c2 VARCHAR(20)
    -> )
    -> PARTITION BY LIST(c1) (
    ->     PARTITION p0 VALUES IN (0, 3, 6),
    ->     PARTITION p1 VALUES IN (1, 4, 7),
    ->     PARTITION p2 VALUES IN (2, 5, 8)
    -> );
Query OK, 0 rows affected (0.01 sec)
```

```
mysql> INSERT INTO ts1 VALUES (9, 'mothra');
ERROR 1504 (HY000): Table has no partition for value 9

mysql> INSERT INTO ts1 VALUES (NULL, 'mothra');
ERROR 1504 (HY000): Table has no partition for value NULL
```

Only rows having a c1 value between 0 and 8 inclusive can be inserted into ts1. NULL falls outside this range, just like the number 9. We can create tables ts2 and ts3 having value lists containing NULL, as shown here:

```
mysql> CREATE TABLE ts2 (
    ->     c1 INT,
    ->     c2 VARCHAR(20)
    -> )
    -> PARTITION BY LIST(c1) (
    ->     PARTITION p0 VALUES IN (0, 3, 6),
    ->     PARTITION p1 VALUES IN (1, 4, 7),
    ->     PARTITION p2 VALUES IN (2, 5, 8),
    ->     PARTITION p3 VALUES IN (NULL)
    -> );
Query OK, 0 rows affected (0.01 sec)

mysql> CREATE TABLE ts3 (
    ->     c1 INT,
    ->     c2 VARCHAR(20)
    -> )
    -> PARTITION BY LIST(c1) (
    ->     PARTITION p0 VALUES IN (0, 3, 6),
    ->     PARTITION p1 VALUES IN (1, 4, 7, NULL),
    ->     PARTITION p2 VALUES IN (2, 5, 8)
    -> );
Query OK, 0 rows affected (0.01 sec)
```

When defining value lists for partitioning, you can (and should) treat NULL just as you would any other value. For example, both VALUES IN (NULL) and VALUES IN (1, 4, 7, NULL) are valid, as are VALUES IN (1, NULL, 4, 7), VALUES IN (NULL, 1, 4, 7), and so on. You can insert a row having NULL for column c1 into each of the tables ts2 and ts3:

```
mysql> INSERT INTO ts2 VALUES (NULL, 'mothra');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO ts3 VALUES (NULL, 'mothra');
Query OK, 1 row affected (0.00 sec)
```

By issuing the appropriate query against INFORMATION_SCHEMA.PARTITIONS, you can determine which partitions were used to store the rows just inserted (we assume, as in the previous examples, that the partitioned tables were created in the p database):

```
mysql> SELECT TABLE_NAME, PARTITION_NAME, TABLE_ROWS, AVG_ROW_LENGTH, DATA_LENGTH
     >   FROM INFORMATION_SCHEMA.PARTITIONS
     >   WHERE TABLE_SCHEMA = 'p' AND TABLE_NAME LIKE 'ts_';
+------------+----------------+------------+----------------+-------------+
| TABLE_NAME | PARTITION_NAME | TABLE_ROWS | AVG_ROW_LENGTH | DATA_LENGTH |
+------------+----------------+------------+----------------+-------------+
| ts2        | p0             |          0 |              0 |           0 |
| ts2        | p1             |          0 |              0 |           0 |
| ts2        | p2             |          0 |              0 |           0 |
| ts2        | p3             |          1 |             20 |          20 |
| ts3        | p0             |          0 |              0 |           0 |
| ts3        | p1             |          1 |             20 |          20 |
| ts3        | p2             |          0 |              0 |           0 |
+------------+----------------+------------+----------------+-------------+
7 rows in set (0.01 sec)
```

As shown earlier in this section, you can also verify which partitions were used for storing the rows by deleting these partitions and then performing a SELECT.

Handling of NULL with HASH and KEY partitioning. NULL is handled somewhat differently for tables partitioned by HASH or KEY. In these cases, any partition expression that yields a NULL value is treated as though its return value were zero. We can verify this behavior by examining the effects on the file system of creating a table partitioned by HASH and populating it with a record containing appropriate values. Suppose that you have a table th (also in the p database) created using the following statement:

```
mysql> CREATE TABLE th (
    ->     c1 INT,
    ->     c2 VARCHAR(20)
    -> )
    -> PARTITION BY HASH(c1)
    -> PARTITIONS 2;
Query OK, 0 rows affected (0.00 sec)
```

The partitions belonging to this table can be viewed using the query shown here:

```
mysql> SELECT TABLE_NAME,PARTITION_NAME,TABLE_ROWS,AVG_ROW_LENGTH,DATA_LENGTH
     >   FROM INFORMATION_SCHEMA.PARTITIONS
     >   WHERE TABLE_SCHEMA = 'p' AND TABLE_NAME ='th';
+------------+----------------+------------+----------------+-------------+
| TABLE_NAME | PARTITION_NAME | TABLE_ROWS | AVG_ROW_LENGTH | DATA_LENGTH |
+------------+----------------+------------+----------------+-------------+
| th         | p0             |          0 |              0 |           0 |
| th         | p1             |          0 |              0 |           0 |
+------------+----------------+------------+----------------+-------------+
2 rows in set (0.00 sec)
```

Note that TABLE_ROWS for each partition is 0. Now insert two rows into th whose c1 column values are NULL and 0, and verify that these rows were inserted, as shown here:

```
mysql> INSERT INTO th VALUES (NULL, 'mothra'), (0, 'gigan');
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM th;
+------+---------+
| c1   | c2      |
+------+---------+
| NULL | mothra  |
+------+---------+
|    0 | gigan   |
+------+---------+
2 rows in set (0.01 sec)
```

Recall that for any integer *N*, the value of NULL MOD *N* is always NULL. For tables that are partitioned by HASH or KEY, this result is treated for determining the correct partition as 0. Checking the INFORMATION_SCHEMA.PARTITIONS table once again, we can see that both rows were inserted into partition p0:

```
mysql> SELECT TABLE_NAME, PARTITION_NAME, TABLE_ROWS, AVG_ROW_LENGTH, DATA_LENGTH
     >   FROM INFORMATION_SCHEMA.PARTITIONS
     >   WHERE TABLE_SCHEMA = 'p' AND TABLE_NAME ='th';
+------------+----------------+------------+----------------+-------------+
| TABLE_NAME | PARTITION_NAME | TABLE_ROWS | AVG_ROW_LENGTH | DATA_LENGTH |
+------------+----------------+------------+----------------+-------------+
| th         | p0             |          2 |             20 |          20 |
| th          | p1               |           0 |               0 |            0 |
+------------+----------------+------------+----------------+-------------+
2 rows in set (0.00 sec)
```

If you repeat this example using `PARTITION BY KEY` in place of `PARTITION BY HASH` in the definition of the table, you can verify easily that `NULL` is also treated like 0 for this type of partitioning.

17.3 Partition Management

MySQL 5.7 provides a number of ways to modify partitioned tables. It is possible to add, drop, redefine, merge, or split existing partitions. All of these actions can be carried out using the partitioning extensions to the `ALTER TABLE` statement. There are also ways to obtain information about partitioned tables and partitions. We discuss these topics in the sections that follow.

- For information about partition management in tables partitioned by `RANGE` or `LIST`, see Section 17.3.1, “Management of `RANGE` and `LIST` Partitions”.
- For a discussion of managing `HASH` and `KEY` partitions, see Section 17.3.2, “Management of `HASH` and `KEY` Partitions”.
- See Section 17.3.5, “Obtaining Information About Partitions”, for a discussion of mechanisms provided in MySQL 5.7 for obtaining information about partitioned tables and partitions.
- For a discussion of performing maintenance operations on partitions, see Section 17.3.4, “Maintenance of Partitions”.

Note

In MySQL 5.7, all partitions of a partitioned table must have the same number of subpartitions, and it is not possible to change the subpartitioning once the table has been created.

To change a table's partitioning scheme, it is necessary only to use the `ALTER TABLE` statement with a *`partition_options`* clause. This clause has the same syntax as that as used with `CREATE TABLE` for creating a partitioned table, and always begins with the keywords `PARTITION BY`. Suppose that you have a table partitioned by range using the following `CREATE TABLE` statement:

```
CREATE TABLE trb3 (id INT, name VARCHAR(50), purchased DATE)
    PARTITION BY RANGE( YEAR(purchased) ) (
        PARTITION p0 VALUES LESS THAN (1990),
        PARTITION p1 VALUES LESS THAN (1995),
        PARTITION p2 VALUES LESS THAN (2000),
        PARTITION p3 VALUES LESS THAN (2005)
    );
```

To repartition this table so that it is partitioned by key into two partitions using the `id` column value as the basis for the key, you can use this statement:

```
ALTER TABLE trb3 PARTITION BY KEY(id) PARTITIONS 2;
```

This has the same effect on the structure of the table as dropping the table and re-creating it using `CREATE TABLE trb3 PARTITION BY KEY(id) PARTITIONS 2;`.

`ALTER TABLE ... ENGINE = ...` changes only the storage engine used by the table, and leaves the table's partitioning scheme intact. Use `ALTER TABLE ... REMOVE PARTITIONING` to remove a table's partitioning. See Section 13.1.6, “`ALTER TABLE` Syntax”.

Important

Only a single PARTITION BY, ADD PARTITION, DROP PARTITION, REORGANIZE PARTITION, or COALESCE PARTITION clause can be used in a given ALTER TABLE statement. If you (for example) wish to drop a partition and reorganize a table's remaining partitions, you must do so in two separate ALTER TABLE statements (one using DROP PARTITION and then a second one using REORGANIZE PARTITIONS).

In MySQL 5.7, it is possible to delete all rows from one or more selected partitions using ALTER TABLE ... TRUNCATE PARTITION.

17.3.1 Management of RANGE and LIST Partitions

Range and list partitions are very similar with regard to how the adding and dropping of partitions are handled. For this reason we discuss the management of both sorts of partitioning in this section. For information about working with tables that are partitioned by hash or key, see Section 17.3.2, “Management of HASH and KEY Partitions”. Dropping a RANGE or LIST partition is more straightforward than adding one, so we discuss this first.

Dropping a partition from a table that is partitioned by either RANGE or by LIST can be accomplished using the ALTER TABLE statement with a DROP PARTITION clause. Here is a very basic example, which supposes that you have already created a table which is partitioned by range and then populated with 10 records using the following CREATE TABLE and INSERT statements:

```
mysql> CREATE TABLE tr (id INT, name VARCHAR(50), purchased DATE)
    ->     PARTITION BY RANGE( YEAR(purchased) ) (
    ->         PARTITION p0 VALUES LESS THAN (1990),
    ->         PARTITION p1 VALUES LESS THAN (1995),
    ->         PARTITION p2 VALUES LESS THAN (2000),
    ->         PARTITION p3 VALUES LESS THAN (2005)
    ->     );
Query OK, 0 rows affected (0.01 sec)

mysql> INSERT INTO tr VALUES
    ->     (1, 'desk organiser', '2003-10-15'),
    ->     (2, 'CD player', '1993-11-05'),
    ->     (3, 'TV set', '1996-03-10'),
    ->     (4, 'bookcase', '1982-01-10'),
    ->     (5, 'exercise bike', '2004-05-09'),
    ->     (6, 'sofa', '1987-06-05'),
    ->     (7, 'popcorn maker', '2001-11-22'),
    ->     (8, 'aquarium', '1992-08-04'),
    ->     (9, 'study desk', '1984-09-16'),
    ->     (10, 'lava lamp', '1998-12-25');
Query OK, 10 rows affected (0.01 sec)
```

You can see which items should have been inserted into partition p2 as shown here:

```
mysql> SELECT * FROM tr
    -> WHERE purchased BETWEEN '1995-01-01' AND '1999-12-31';
+------+-----------+------------+
| id   | name      | purchased  |
+------+-----------+------------+
|    3 | TV set    | 1996-03-10 |
|   10 | lava lamp | 1998-12-25 |
+------+-----------+------------+
2 rows in set (0.00 sec)
```

To drop the partition named p2, execute the following command:

```
mysql> ALTER TABLE tr DROP PARTITION p2;
Query OK, 0 rows affected (0.03 sec)
```

It is very important to remember that, *when you drop a partition, you also delete all the data that was stored in that partition*. You can see that this is the case by re-running the previous SELECT query:

```
mysql> SELECT * FROM tr WHERE purchased
    -> BETWEEN '1995-01-01' AND '1999-12-31';
Empty set (0.00 sec)
```

Because of this, you must have the DROP privilege for a table before you can execute ALTER TABLE ... DROP PARTITION on that table.

If you wish to drop all data from all partitions while preserving the table definition and its partitioning scheme, use the TRUNCATE TABLE statement. (See Section 13.1.27, “TRUNCATE TABLE Syntax”.)

If you intend to change the partitioning of a table *without* losing data, use ALTER TABLE ... REORGANIZE PARTITION instead. See below or in Section 13.1.6, “ALTER TABLE Syntax”, for information about REORGANIZE PARTITION.

If you now execute a SHOW CREATE TABLE statement, you can see how the partitioning makeup of the table has been changed:

```
mysql> SHOW CREATE TABLE tr\G
*************************** 1. row ***************************
       Table: tr
Create Table: CREATE TABLE `tr` (
  `id` int(11) default NULL,
  `name` varchar(50) default NULL,
  `purchased` date default NULL
) ENGINE=MyISAM DEFAULT CHARSET=latin1
PARTITION BY RANGE ( YEAR(purchased) ) (
  PARTITION p0 VALUES LESS THAN (1990) ENGINE = MyISAM,
  PARTITION p1 VALUES LESS THAN (1995) ENGINE = MyISAM,
  PARTITION p3 VALUES LESS THAN (2005) ENGINE = MyISAM
)
1 row in set (0.01 sec)
```

When you insert new rows into the changed table with purchased column values between '1995-01-01' and '2004-12-31' inclusive, those rows will be stored in partition p3. You can verify this as follows:

```
mysql> INSERT INTO tr VALUES (11, 'pencil holder', '1995-07-12');
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM tr WHERE purchased
    -> BETWEEN '1995-01-01' AND '2004-12-31';
+------+----------------+------------+
| id   | name           | purchased  |
+------+----------------+------------+
|   11 | pencil holder  | 1995-07-12 |
|    1 | desk organiser | 2003-10-15 |
|    5 | exercise bike  | 2004-05-09 |
|    7 | popcorn maker  | 2001-11-22 |
+------+----------------+------------+
4 rows in set (0.00 sec)

mysql> ALTER TABLE tr DROP PARTITION p3;
Query OK, 0 rows affected (0.03 sec)
```

```
mysql> SELECT * FROM tr WHERE purchased
    -> BETWEEN '1995-01-01' AND '2004-12-31';
Empty set (0.00 sec)
```

Note that the number of rows dropped from the table as a result of ALTER TABLE ... DROP PARTITION is not reported by the server as it would be by the equivalent DELETE query.

Dropping LIST partitions uses exactly the same ALTER TABLE ... DROP PARTITION syntax as used for dropping RANGE partitions. However, there is one important difference in the effect this has on your use of the table afterward: You can no longer insert into the table any rows having any of the values that were included in the value list defining the deleted partition. (See Section 17.2.2, "LIST Partitioning", for an example.)

To add a new range or list partition to a previously partitioned table, use the ALTER TABLE ... ADD PARTITION statement. For tables which are partitioned by RANGE, this can be used to add a new range to the end of the list of existing partitions. Suppose that you have a partitioned table containing membership data for your organization, which is defined as follows:

```
CREATE TABLE members (
    id INT,
    fname VARCHAR(25),
    lname VARCHAR(25),
    dob DATE
)
PARTITION BY RANGE( YEAR(dob) ) (
    PARTITION p0 VALUES LESS THAN (1970),
    PARTITION p1 VALUES LESS THAN (1980),
    PARTITION p2 VALUES LESS THAN (1990)
);
```

Suppose further that the minimum age for members is 16. As the calendar approaches the end of 2005, you realize that you will soon be admitting members who were born in 1990 (and later in years to come). You can modify the members table to accommodate new members born in the years 1990 to 1999 as shown here:

```
ALTER TABLE members ADD PARTITION (PARTITION p3 VALUES LESS THAN (2000));
```

With tables that are partitioned by range, you can use ADD PARTITION to add new partitions to the high end of the partitions list only. Trying to add a new partition in this manner between or before existing partitions results in an error as shown here:

```
mysql> ALTER TABLE members
     >     ADD PARTITION (
     >     PARTITION n VALUES LESS THAN (1960));
ERROR 1463 (HY000): VALUES LESS THAN value must be strictly »
   increasing for each partition
```

You can work around this problem by reorganizing the first partition into two new ones that split the range between them, like this:

```
ALTER TABLE members
    REORGANIZE PARTITION p0 INTO (
        PARTITION n0 VALUES LESS THAN (1960),
        PARTITION n1 VALUES LESS THAN (1970)
);
```

Using SHOW CREATE TABLE you can see that the ALTER TABLE statement has had the desired effect:

```
mysql> SHOW CREATE TABLE members\G
*************************** 1. row ***************************
       Table: members
Create Table: CREATE TABLE `members` (
  `id` int(11) DEFAULT NULL,
  `fname` varchar(25) DEFAULT NULL,
  `lname` varchar(25) DEFAULT NULL,
  `dob` date DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1
/*!50100 PARTITION BY RANGE ( YEAR(dob))
(PARTITION n0 VALUES LESS THAN (1960) ENGINE = InnoDB,
 PARTITION n1 VALUES LESS THAN (1970) ENGINE = InnoDB,
 PARTITION p1 VALUES LESS THAN (1980) ENGINE = InnoDB,
 PARTITION p2 VALUES LESS THAN (1990) ENGINE = InnoDB,
 PARTITION p3 VALUES LESS THAN (2000) ENGINE = InnoDB) */
1 row in set (0.00 sec)
```

See also Section 13.1.6.1, "ALTER TABLE Partition Operations".

You can also use ALTER TABLE ... ADD PARTITION to add new partitions to a table that is partitioned by LIST. Suppose a table tt is defined using the following CREATE TABLE statement:

```
CREATE TABLE tt (
    id INT,
    data INT
)
PARTITION BY LIST(data) (
    PARTITION p0 VALUES IN (5, 10, 15),
    PARTITION p1 VALUES IN (6, 12, 18)
);
```

You can add a new partition in which to store rows having the data column values 7, 14, and 21 as shown:

```
ALTER TABLE tt ADD PARTITION (PARTITION p2 VALUES IN (7, 14, 21));
```

Note that you *cannot* add a new LIST partition encompassing any values that are already included in the value list of an existing partition. If you attempt to do so, an error will result:

```
mysql> ALTER TABLE tt ADD PARTITION
     >     (PARTITION np VALUES IN (4, 8, 12));
ERROR 1465 (HY000): Multiple definition of same constant »
                    in list partitioning
```

Because any rows with the data column value 12 have already been assigned to partition p1, you cannot create a new partition on table tt that includes 12 in its value list. To accomplish this, you could drop p1, and add np and then a new p1 with a modified definition. However, as discussed earlier, this would result in the loss of all data stored in p1—and it is often the case that this is not what you really want to do. Another solution might appear to be to make a copy of the table with the new partitioning and to copy the data into it using CREATE TABLE ... SELECT ..., then drop the old table and rename the new one, but this could be very time-consuming when dealing with a large amounts of data. This also might not be feasible in situations where high availability is a requirement.

You can add multiple partitions in a single ALTER TABLE ... ADD PARTITION statement as shown here:

```
CREATE TABLE employees (
  id INT NOT NULL,
  fname VARCHAR(50) NOT NULL,
  lname VARCHAR(50) NOT NULL,
  hired DATE NOT NULL
)
PARTITION BY RANGE( YEAR(hired) ) (
  PARTITION p1 VALUES LESS THAN (1991),
  PARTITION p2 VALUES LESS THAN (1996),
  PARTITION p3 VALUES LESS THAN (2001),
  PARTITION p4 VALUES LESS THAN (2005)
);

ALTER TABLE employees ADD PARTITION (
    PARTITION p5 VALUES LESS THAN (2010),
    PARTITION p6 VALUES LESS THAN MAXVALUE
);
```

Fortunately, MySQL's partitioning implementation provides ways to redefine partitions without losing data. Let us look first at a couple of simple examples involving `RANGE` partitioning. Recall the `members` table which is now defined as shown here:

```
mysql> SHOW CREATE TABLE members\G
*************************** 1. row ***************************
       Table: members
Create Table: CREATE TABLE `members` (
  `id` int(11) default NULL,
  `fname` varchar(25) default NULL,
  `lname` varchar(25) default NULL,
  `dob` date default NULL
) ENGINE=MyISAM DEFAULT CHARSET=latin1
PARTITION BY RANGE ( YEAR(dob) ) (
  PARTITION p0 VALUES LESS THAN (1970) ENGINE = MyISAM,
  PARTITION p1 VALUES LESS THAN (1980) ENGINE = MyISAM,
  PARTITION p2 VALUES LESS THAN (1990) ENGINE = MyISAM.
  PARTITION p3 VALUES LESS THAN (2000) ENGINE = MyISAM
)
```

Suppose that you would like to move all rows representing members born before 1960 into a separate partition. As we have already seen, this cannot be done using `ALTER TABLE ... ADD PARTITION`. However, you can use another partition-related extension to `ALTER TABLE` to accomplish this:

```
ALTER TABLE members REORGANIZE PARTITION p0 INTO (
    PARTITION s0 VALUES LESS THAN (1960),
    PARTITION s1 VALUES LESS THAN (1970)
);
```

In effect, this command splits partition `p0` into two new partitions `s0` and `s1`. It also moves the data that was stored in `p0` into the new partitions according to the rules embodied in the two `PARTITION ... VALUES ...` clauses, so that `s0` contains only those records for which `YEAR(dob)` is less than 1960 and `s1` contains those rows in which `YEAR(dob)` is greater than or equal to 1960 but less than 1970.

A `REORGANIZE PARTITION` clause may also be used for merging adjacent partitions. You can return the `members` table to its previous partitioning as shown here:

```
ALTER TABLE members REORGANIZE PARTITION s0,s1 INTO (
    PARTITION p0 VALUES LESS THAN (1970)
);
```

No data is lost in splitting or merging partitions using `REORGANIZE PARTITION`. In executing the above statement, MySQL moves all of the records that were stored in partitions `s0` and `s1` into partition `p0`.

The general syntax for `REORGANIZE PARTITION` is shown here:

```
ALTER TABLE tbl_name
    REORGANIZE PARTITION partition_list
    INTO (partition_definitions);
```

Here, *tbl_name* is the name of the partitioned table, and *partition_list* is a comma-separated list of names of one or more existing partitions to be changed. *partition_definitions* is a comma-separated list of new partition definitions, which follow the same rules as for the *partition_definitions* list used in CREATE TABLE (see Section 13.1.14, “CREATE TABLE Syntax”). It should be noted that you are not limited to merging several partitions into one, or to splitting one partition into many, when using REORGANIZE PARTITION. For example, you can reorganize all four partitions of the members table into two, as follows:

```
ALTER TABLE members REORGANIZE PARTITION p0,p1,p2,p3 INTO (
    PARTITION m0 VALUES LESS THAN (1980),
    PARTITION m1 VALUES LESS THAN (2000)
);
```

You can also use REORGANIZE PARTITION with tables that are partitioned by LIST. Let us return to the problem of adding a new partition to the list-partitioned tt table and failing because the new partition had a value that was already present in the value-list of one of the existing partitions. We can handle this by adding a partition that contains only nonconflicting values, and then reorganizing the new partition and the existing one so that the value which was stored in the existing one is now moved to the new one:

```
ALTER TABLE tt ADD PARTITION (PARTITION np VALUES IN (4, 8));
ALTER TABLE tt REORGANIZE PARTITION p1,np INTO (
    PARTITION p1 VALUES IN (6, 18),
    PARTITION np VALUES in (4, 8, 12)
);
```

Here are some key points to keep in mind when using ALTER TABLE ... REORGANIZE PARTITION to repartition tables that are partitioned by RANGE or LIST:

- The PARTITION clauses used to determine the new partitioning scheme are subject to the same rules as those used with a CREATE TABLE statement.

  Most importantly, you should remember that the new partitioning scheme cannot have any overlapping ranges (applies to tables partitioned by RANGE) or sets of values (when reorganizing tables partitioned by LIST).

- The combination of partitions in the *partition_definitions* list should account for the same range or set of values overall as the combined partitions named in the *partition_list*.

  For instance, in the members table used as an example in this section, partitions p1 and p2 together cover the years 1980 through 1999. Therefore, any reorganization of these two partitions should cover the same range of years overall.

- For tables partitioned by RANGE, you can reorganize only adjacent partitions; you cannot skip over range partitions.

  For instance, you could not reorganize the members table used as an example in this section using a statement beginning with ALTER TABLE members REORGANIZE PARTITION p0,p2 INTO ... because p0 covers the years prior to 1970 and p2 the years from 1990 through 1999 inclusive, and thus the two are not adjacent partitions.

- You cannot use REORGANIZE PARTITION to change the table's partitioning type; that is, you cannot (for example) change RANGE partitions to HASH partitions or *vice versa*. You also cannot use this command to change the partitioning expression or column. To accomplish either of these tasks without dropping and re-creating the table, you can use `ALTER TABLE ... PARTITION BY ...`. For example:

```
ALTER TABLE members
    PARTITION BY HASH( YEAR(dob) )
    PARTITIONS 8;
```

17.3.2 Management of `HASH` and `KEY` Partitions

Tables which are partitioned by hash or by key are very similar to one another with regard to making changes in a partitioning setup, and both differ in a number of ways from tables which have been partitioned by range or list. For that reason, this section addresses the modification of tables partitioned by hash or by key only. For a discussion of adding and dropping of partitions of tables that are partitioned by range or list, see Section 17.3.1, “Management of `RANGE` and `LIST` Partitions”.

You cannot drop partitions from tables that are partitioned by `HASH` or `KEY` in the same way that you can from tables that are partitioned by `RANGE` or `LIST`. However, you can merge `HASH` or `KEY` partitions using the `ALTER TABLE ... COALESCE PARTITION` statement. Suppose that you have a table containing data about clients, which is divided into twelve partitions. The `clients` table is defined as shown here:

```
CREATE TABLE clients (
    id INT,
    fname VARCHAR(30),
    lname VARCHAR(30),
    signed DATE
)
PARTITION BY HASH( MONTH(signed) )
PARTITIONS 12;
```

To reduce the number of partitions from twelve to eight, execute the following `ALTER TABLE` command:

```
mysql> ALTER TABLE clients COALESCE PARTITION 4;
Query OK, 0 rows affected (0.02 sec)
```

`COALESCE` works equally well with tables that are partitioned by `HASH`, `KEY`, `LINEAR HASH`, or `LINEAR KEY`. Here is an example similar to the previous one, differing only in that the table is partitioned by `LINEAR KEY`:

```
mysql> CREATE TABLE clients_lk (
    ->     id INT,
    ->     fname VARCHAR(30),
    ->     lname VARCHAR(30),
    ->     signed DATE
    -> )
    -> PARTITION BY LINEAR KEY(signed)
    -> PARTITIONS 12;
Query OK, 0 rows affected (0.03 sec)

mysql> ALTER TABLE clients_lk COALESCE PARTITION 4;
Query OK, 0 rows affected (0.06 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

Note that the number following `COALESCE PARTITION` is the number of partitions to merge into the remainder—in other words, it is the number of partitions to remove from the table.

If you attempt to remove more partitions than the table has, the result is an error like the one shown:

```
mysql> ALTER TABLE clients COALESCE PARTITION 18;
ERROR 1478 (HY000): Cannot remove all partitions, use DROP TABLE instead
```

To increase the number of partitions for the `clients` table from 12 to 18. use `ALTER TABLE ... ADD PARTITION` as shown here:

```
ALTER TABLE clients ADD PARTITION PARTITIONS 6;
```

17.3.3 Exchanging Partitions and Subpartitions with Tables

In MySQL 5.7, it is possible to exchange a table partition or subpartition with a table using the `ALTER TABLE ... EXCHANGE PARTITION` statement—that is, to move any existing rows in the partition or subpartition to the nonpartitioned table, and any existing rows in the nonpartitioned table to the table partition or subpartition.

Such operations are subject to the following conditions:

- The table to be exchanged must not be partitioned, but must otherwise have the same table structure as the partitioned table.
- The table to be exchanged must not be a temporary table.
- Any rows existing in the nonpartitioned table prior to the exchange must lie within the range defined for the partition or subpartition.
- The table to be exchanged may not have any foreign keys, nor may any other tables have foreign keys which reference this table.
- In addition to the `ALTER`, `INSERT`, and `CREATE` privileges usually required for `ALTER TABLE` statements, you must have the `DROP` privilege to perform `ALTER TABLE ... EXCHANGE PARTITION`.

In addition, you should also be aware of the following effects of `ALTER TABLE ... EXCHANGE PARTITION`:

- Executing this statement does *not* invoke any triggers on either the partitioned table or the exchanged table.
- Any `AUTO_INCREMENT` columns in the exchanged table are reset.

The complete syntax of the the `ALTER TABLE ... EXCHANGE PARTITION` statement is shown here, where *`pt`* is the partitioned table, *`p`* is the partition or subpartition to be exchanged, and *`t`* is the nonpartitioned table to be exchanged with *`p`*:

```
ALTER TABLE pt
    EXCHANGE PARTITION p
    WITH TABLE t;
```

One and only one partition or subpartition may be exchanged with one and only one nonpartitioned table in a single `ALTER TABLE EXCHANGE PARTITION` statement. To exchange multiple partitions or subpartitions, use multiple `ALTER TABLE EXCHANGE PARTITION` statements. `EXCHANGE PARTITION` may not be combined with other `ALTER TABLE` options. The partitioning and (if applicable) subpartitioning used by the partitioned table may be of any type or types supported in MySQL 5.7.

Suppose that a partitioned table `e` has been created and populated using the following SQL statements:

```
CREATE TABLE e (
    id INT NOT NULL,
    fname VARCHAR(30),
    lname VARCHAR(30)
)
    PARTITION BY RANGE (id) (
        PARTITION p0 VALUES LESS THAN (50),
        PARTITION p1 VALUES LESS THAN (100),
        PARTITION p2 VALUES LESS THAN (150),
        PARTITION p3 VALUES LESS THAN (MAXVALUE)
);

INSERT INTO e VALUES
    (1669, "Jim", "Smith"),
    (337, "Mary", "Jones"),
    (16, "Frank", "White"),
    (2005, "Linda", "Black");
```

Now we create a nonpartitioned copy of e named e2. This can be done using the mysql client as shown here:

```
mysql> CREATE TABLE e2 LIKE e;
Query OK, 0 rows affected (1.34 sec)

mysql> ALTER TABLE e2 REMOVE PARTITIONING;
Query OK, 0 rows affected (0.90 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

You can see which partitions in table e contain rows by querying the INFORMATION_SCHEMA.PARTITIONS table, like this:

```
mysql> SELECT PARTITION_NAME, TABLE_ROWS
    ->     FROM INFORMATION_SCHEMA.PARTITIONS
    ->     WHERE TABLE_NAME = 'e';
+----------------+------------+
| PARTITION_NAME | TABLE_ROWS |
+----------------+------------+
| p0             |          1 |
| p1             |          0 |
| p2             |          0 |
| p3             |          3 |
+----------------+------------+
4 rows in set (0.00 sec)
```

To exchange partition p0 in table e with table e2, you can use the ALTER TABLE statement shown here:

```
mysql> ALTER TABLE e EXCHANGE PARTITION p0 WITH TABLE e2;
Query OK, 0 rows affected (0.28 sec)
```

More precisely, the statement just issued causes any rows found in the partition to be swapped with those found in the table. You can observe how this has happened by querying the INFORMATION_SCHEMA.PARTITIONS table, as before. The table row that was previously found in partition p0 is no longer present:

```
mysql> SELECT PARTITION_NAME, TABLE_ROWS
    ->     FROM INFORMATION_SCHEMA.PARTITIONS
    ->     WHERE TABLE_NAME = 'e';
+----------------+------------+
| PARTITION_NAME | TABLE_ROWS |
+----------------+------------+
| p0             |          0 |
| p1             |          0 |
| p2             |          0 |
| p3             |          3 |
+----------------+------------+
4 rows in set (0.00 sec)
```

If you query table e2, you can see that the “missing” row can now be found there:

```
mysql> SELECT * FROM e2;
+----+-------+-------+
| id | fname | lname |
+----+-------+-------+
| 16 | Frank | White |
+----+-------+-------+
1 row in set (0.00 sec)
```

The table to be exchanged with the partition does not necessarily have to be empty. To demonstrate this, we first insert a new row into table e, making sure that this row is stored in partition p0 by choosing an id column value that is less than 50, and verifying this afterwards by querying the PARTITIONS table:

```
mysql> INSERT INTO e VALUES (41, "Michael", "Green");
Query OK, 1 row affected (0.05 sec)

mysql> SELECT PARTITION_NAME, TABLE_ROWS
    ->     FROM INFORMATION_SCHEMA.PARTITIONS
    ->     WHERE TABLE_NAME = 'e';
+----------------+------------+
| PARTITION_NAME | TABLE_ROWS |
+----------------+------------+
| p0             |          1 |
| p1             |          0 |
| p2             |          0 |
| p3             |          3 |
+----------------+------------+
4 rows in set (0.00 sec)
```

Now we once again exchange partition p0 with table e2 using the same ALTER TABLE statement as previously:

```
mysql> ALTER TABLE e EXCHANGE PARTITION p0 WITH TABLE e2;
Query OK, 0 rows affected (0.28 sec)
```

The output of the following queries shows that the table row that was stored in partition p0 and the table row that was stored in table e2, prior to issuing the ALTER TABLE statement, have now switched places:

```
mysql> SELECT * FROM e;
+------+-------+-------+
| id   | fname | lname |
+------+-------+-------+
|   16 | Frank | White |
| 1669 | Jim   | Smith |
|  337 | Mary  | Jones |
| 2005 | Linda | Black |
+------+-------+-------+
4 rows in set (0.00 sec)

mysql> SELECT PARTITION_NAME, TABLE_ROWS
    ->     FROM INFORMATION_SCHEMA.PARTITIONS
    ->     WHERE TABLE_NAME = 'e';
+----------------+------------+
| PARTITION_NAME | TABLE_ROWS |
+----------------+------------+
| p0             |          1 |
| p1             |          0 |
| p2             |          0 |
| p3             |          3 |
+----------------+------------+
4 rows in set (0.00 sec)

mysql> SELECT * FROM e2;
+----+---------+-------+
| id | fname   | lname |
+----+---------+-------+
| 41 | Michael | Green |
+----+---------+-------+
1 row in set (0.00 sec)
```

You should keep in mind that any rows found in the nonpartitioned table prior to issuing the ALTER TABLE ... EXCHANGE PARTITION statement must meet any conditions required for them to be stored in the target partition; otherwise, the statement fails. To see how this occurs, first insert a row into e2 that cannot be stored in partition p0 of table e because its id column value is too large; then, try to exchange the table with the partition again:

```
mysql> INSERT INTO e2 VALUES (51, "Ellen", "McDonald");
Query OK, 1 row affected (0.08 sec)

mysql> ALTER TABLE e EXCHANGE PARTITION p0 WITH TABLE e2;
ERROR 1707 (HY000): Found row that does not match the partition
```

The IGNORE keyword is accepted, but has no effect when used with EXCHANGE PARTITION, as shown here:

```
mysql> ALTER IGNORE TABLE e EXCHANGE PARTITION p0 WITH TABLE e2;
ERROR 1707 (HY000): Found row that does not match the partition
```

You can also exchange a subpartition of a subpartitioned table (see Section 17.2.6, “Subpartitioning”) with a nonpartitioned table using an ALTER TABLE ... EXCHANGE PARTITION statement. In the following example, we first create a table es that is partitioned by RANGE and subpartitioned by KEY, populate this table as we did table e, and then create an empty, nonpartitioned copy es2 of the table, as shown here:

```
mysql> CREATE TABLE es (
    ->     id INT NOT NULL,
    ->     fname VARCHAR(30),
    ->     lname VARCHAR(30)
    -> )
    ->     PARTITION BY RANGE (id)
    ->     SUBPARTITION BY KEY (lname)
    ->     SUBPARTITIONS 2 (
    ->         PARTITION p0 VALUES LESS THAN (50),
    ->         PARTITION p1 VALUES LESS THAN (100),
    ->         PARTITION p2 VALUES LESS THAN (150),
    ->         PARTITION p3 VALUES LESS THAN (MAXVALUE)
    ->     );
Query OK, 0 rows affected (2.76 sec)

mysql> INSERT INTO es VALUES
    ->     (1669, "Jim", "Smith"),
    ->     (337, "Mary", "Jones"),
    ->     (16, "Frank", "White"),
    ->     (2005, "Linda", "Black");
Query OK, 4 rows affected (0.04 sec)
Records: 4  Duplicates: 0  Warnings: 0
```

```
mysql> CREATE TABLE es2 LIKE es;
Query OK, 0 rows affected (1.27 sec)

mysql> ALTER TABLE es2 REMOVE PARTITIONING;
Query OK, 0 rows affected (0.70 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

Although we did not explicitly name any of the subpartitions when creating table es, we can obtain generated names for these by including the SUBPARTITION_NAME of the PARTITIONS table from INFORMATION_SCHEMA when selecting from that table, as shown here:

```
mysql> SELECT PARTITION_NAME, SUBPARTITION_NAME, TABLE_ROWS
    ->     FROM INFORMATION_SCHEMA.PARTITIONS
    ->     WHERE TABLE_NAME = 'es';
+----------------+-------------------+------------+
| PARTITION_NAME | SUBPARTITION_NAME | TABLE_ROWS |
+----------------+-------------------+------------+
| p0             | p0sp0             |          1 |
| p0             | p0sp1             |          0 |
| p1             | p1sp0             |          0 |
| p1             | p1sp1             |          0 |
| p2             | p2sp0             |          0 |
| p2             | p2sp1             |          0 |
| p3             | p3sp0             |          3 |
| p3             | p3sp1             |          0 |
+----------------+-------------------+------------+
8 rows in set (0.00 sec)
```

The following ALTER TABLE statement exchanges subpartition p3sp0 table es with the nonpartitioned table es2:

```
mysql> ALTER TABLE es EXCHANGE PARTITION p3sp0 WITH TABLE es2;
Query OK, 0 rows affected (0.29 sec)
```

You can verify that the rows were exchanged by issuing the following queries:

```
mysql> SELECT PARTITION_NAME, SUBPARTITION_NAME, TABLE_ROWS
    ->     FROM INFORMATION_SCHEMA.PARTITIONS
    ->     WHERE TABLE_NAME = 'es';
+----------------+-------------------+------------+
| PARTITION_NAME | SUBPARTITION_NAME | TABLE_ROWS |
+----------------+-------------------+------------+
| p0             | p0sp0             |          1 |
| p0             | p0sp1             |          0 |
| p1             | p1sp0             |          0 |
| p1             | p1sp1             |          0 |
| p2             | p2sp0             |          0 |
| p2             | p2sp1             |          0 |
| p3             | p3sp0             |          0 |
| p3             | p3sp1             |          0 |
+----------------+-------------------+------------+
8 rows in set (0.00 sec)

mysql> SELECT * FROM es2;
+------+-------+-------+
| id   | fname | lname |
+------+-------+-------+
| 1669 | Jim   | Smith |
|  337 | Mary  | Jones |
| 2005 | Linda | Black |
+------+-------+-------+
3 rows in set (0.00 sec)
```

If a table is subpartitioned, you can exchange only a subpartition of the table—not an entire partition—with an unpartitioned table, as shown here:

```
mysql> ALTER TABLE es EXCHANGE PARTITION p3 WITH TABLE es2;
ERROR 1704 (HY000): Subpartitioned table, use subpartition instead of partition
```

The comparison of table structures used by MySQL is very strict. The number, order, names, and types of columns and indexes of the partitioned table and the nonpartitioned table must match exactly. In addition, both tables must use the same storage engine:

```
mysql> CREATE TABLE es3 LIKE e;
Query OK, 0 rows affected (1.31 sec)

mysql> ALTER TABLE es3 REMOVE PARTITIONING;
Query OK, 0 rows affected (0.53 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> SHOW CREATE TABLE es3\G
*************************** 1. row ***************************
       Table: es3
Create Table: CREATE TABLE `es3` (
  `id` int(11) NOT NULL,
  `fname` varchar(30) DEFAULT NULL,
  `lname` varchar(30) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1
1 row in set (0.00 sec)

mysql> ALTER TABLE es3 ENGINE = MyISAM;
Query OK, 0 rows affected (0.15 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> ALTER TABLE es EXCHANGE PARTITION p3sp0 WITH TABLE es3;
ERROR 1497 (HY000): The mix of handlers in the partitions is not allowed in this version of MySQL
```

17.3.4 Maintenance of Partitions

A number of table and partition maintenance tasks can be carried out using SQL statements intended for such purposes on partitioned tables in MySQL 5.7.

Table maintenance of partitioned tables can be accomplished using the statements CHECK TABLE, OPTIMIZE TABLE, ANALYZE TABLE, and REPAIR TABLE, which are supported for partitioned tables.

You can use a number of extensions to ALTER TABLE for performing operations of this type on one or more partitions directly, as described in the following list:

- **Rebuilding partitions.** Rebuilds the partition; this has the same effect as dropping all records stored in the partition, then reinserting them. This can be useful for purposes of defragmentation.

  Example:

  ```
  ALTER TABLE t1 REBUILD PARTITION p0, p1;
  ```

- **Optimizing partitions.** If you have deleted a large number of rows from a partition or if you have made many changes to a partitioned table with variable-length rows (that is, having VARCHAR, BLOB, or TEXT columns), you can use ALTER TABLE ... OPTIMIZE PARTITION to reclaim any unused space and to defragment the partition data file.

  Example:

```
ALTER TABLE t1 OPTIMIZE PARTITION p0, p1;
```

Using OPTIMIZE PARTITION on a given partition is equivalent to running CHECK PARTITION, ANALYZE PARTITION, and REPAIR PARTITION on that partition.

Some MySQL storage engines, including InnoDB, do not support per-partition optimization; in these cases, ALTER TABLE ... OPTIMIZE PARTITION analyzes and rebuilds the entire table, and causes an appropriate warning to be issued. (Bug #11751825, Bug #42822) Use ALTER TABLE ... REBUILD PARTITION and ALTER TABLE ... ANALYZE PARTITION instead, to avoid this issue.

- **Analyzing partitions.** This reads and stores the key distributions for partitions.

  Example:

  ```
  ALTER TABLE t1 ANALYZE PARTITION p3;
  ```

- **Repairing partitions.** This repairs corrupted partitions.

  Example:

  ```
  ALTER TABLE t1 REPAIR PARTITION p0,p1;
  ```

  Normally, REPAIR PARTITION fails when the partition contains duplicate key errors. In MySQL 5.7.2 and later, you can use ALTER IGNORE TABLE with this option, in which case all rows that cannot be moved due to the presence of duplicate keys are removed from the partition (Bug #16900947).

- **Checking partitions.** You can check partitions for errors in much the same way that you can use CHECK TABLE with nonpartitioned tables.

  Example:

  ```
  ALTER TABLE trb3 CHECK PARTITION p1;
  ```

  This command will tell you if the data or indexes in partition p1 of table t1 are corrupted. If this is the case, use ALTER TABLE ... REPAIR PARTITION to repair the partition.

  Normally, CHECK PARTITION fails when the partition contains duplicate key errors. In MySQL 5.7.2 and later, you can use ALTER IGNORE TABLE with this option, in which case the statement returns the contents of each row in the partition where a duplicate key violation is found. Note that only the values for the columns in the partitioning expression for the table are reported. (Bug #16900947)

Each of the statements in the list just shown also supports the keyword ALL in place of the list of partition names. Using ALL causes the statement to act on all partitions in the table.

The use of mysqlcheck and myisamchk is not supported with partitioned tables.

In MySQL 5.7, you can also truncate partitions using ALTER TABLE ... TRUNCATE PARTITION. This statement can be used to delete all rows from one or more partitions in much the same way that TRUNCATE TABLE deletes all rows from a table.

ALTER TABLE ... TRUNCATE PARTITION ALL truncates all partitions in the table.

Prior to MySQL 5.7.2, ANALYZE, CHECK, OPTIMIZE, REBUILD, REPAIR, and TRUNCATE operations were not permitted on subpartitions (Bug #14028340, Bug #65184).

17.3.5 Obtaining Information About Partitions

This section discusses obtaining information about existing partitions, which can be done in a number of ways. Methods of obtaining such information include the following:

- Using the SHOW CREATE TABLE statement to view the partitioning clauses used in creating a partitioned table.
- Using the SHOW TABLE STATUS statement to determine whether a table is partitioned.
- Querying the INFORMATION_SCHEMA.PARTITIONS table.
- Using the statement EXPLAIN PARTITIONS SELECT to see which partitions are used by a given SELECT.

As discussed elsewhere in this chapter, SHOW CREATE TABLE includes in its output the PARTITION BY clause used to create a partitioned table. For example:

```
mysql> SHOW CREATE TABLE trb3\G
*************************** 1. row ***************************
       Table: trb3
Create Table: CREATE TABLE `trb3` (
  `id` int(11) default NULL,
  `name` varchar(50) default NULL,
  `purchased` date default NULL
) ENGINE=MyISAM DEFAULT CHARSET=latin1
PARTITION BY RANGE (YEAR(purchased)) (
  PARTITION p0 VALUES LESS THAN (1990) ENGINE = MyISAM,
  PARTITION p1 VALUES LESS THAN (1995) ENGINE = MyISAM,
  PARTITION p2 VALUES LESS THAN (2000) ENGINE = MyISAM,
  PARTITION p3 VALUES LESS THAN (2005) ENGINE = MyISAM
)
1 row in set (0.00 sec)
```

The output from SHOW TABLE STATUS for partitioned tables is the same as that for nonpartitioned tables, except that the Create_options column contains the string partitioned. The Engine column contains the name of the storage engine used by all partitions of the table. (See Section 13.7.5.35, “SHOW TABLE STATUS Syntax”, for more information about this statement.)

You can also obtain information about partitions from INFORMATION_SCHEMA, which contains a PARTITIONS table. See Section 19.14, “The INFORMATION_SCHEMA PARTITIONS Table”.

It is possible to determine which partitions of a partitioned table are involved in a given SELECT query using EXPLAIN PARTITIONS. The PARTITIONS keyword adds a partitions column to the output of EXPLAIN listing the partitions from which records would be matched by the query.

Suppose that you have a table trb1 created and populated as follows:

```
CREATE TABLE trb1 (id INT, name VARCHAR(50), purchased DATE)
    PARTITION BY RANGE(id)
    (
        PARTITION p0 VALUES LESS THAN (3),
        PARTITION p1 VALUES LESS THAN (7),
        PARTITION p2 VALUES LESS THAN (9),
        PARTITION p3 VALUES LESS THAN (11)
    );

INSERT INTO trb1 VALUES
    (1, 'desk organiser', '2003-10-15'),
    (2, 'CD player', '1993-11-05'),
    (3, 'TV set', '1996-03-10'),
    (4, 'bookcase', '1982-01-10'),
    (5, 'exercise bike', '2004-05-09'),
    (6, 'sofa', '1987-06-05'),
    (7, 'popcorn maker', '2001-11-22'),
    (8, 'aquarium', '1992-08-04'),
    (9, 'study desk', '1984-09-16'),
    (10, 'lava lamp', '1998-12-25');
```

You can see which partitions are used in a query such as `SELECT * FROM trb1;`, as shown here:

```
mysql> EXPLAIN PARTITIONS SELECT * FROM trb1\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: trb1
   partitions: p0,p1,p2,p3
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 10
        Extra: Using filesort
```

In this case, all four partitions are searched. However, when a limiting condition making use of the partitioning key is added to the query, you can see that only those partitions containing matching values are scanned, as shown here:

```
mysql> EXPLAIN PARTITIONS SELECT * FROM trb1 WHERE id < 5\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: trb1
   partitions: p0,p1
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 10
        Extra: Using where
```

`EXPLAIN PARTITIONS` provides information about keys used and possible keys, just as with the standard `EXPLAIN SELECT` statement:

```
mysql> ALTER TABLE trb1 ADD PRIMARY KEY (id);
Query OK, 10 rows affected (0.03 sec)
Records: 10  Duplicates: 0  Warnings: 0

mysql> EXPLAIN PARTITIONS SELECT * FROM trb1 WHERE id < 5\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: trb1
   partitions: p0,p1
         type: range
possible_keys: PRIMARY
          key: PRIMARY
      key_len: 4
          ref: NULL
         rows: 7
        Extra: Using where
```

You should take note of the following restrictions and limitations on EXPLAIN PARTITIONS:

- You cannot use the PARTITIONS and EXTENDED keywords together in the same EXPLAIN ... SELECT statement. Attempting to do so produces a syntax error.
- If EXPLAIN PARTITIONS is used to examine a query against a nonpartitioned table, no error is produced, but the value of the partitions column is always NULL.

The rows column of EXPLAIN PARTITIONS output displays the total number of rows in the table.

See also Section 13.8.2, “EXPLAIN Syntax”.

17.4 Partition Pruning

This section discusses an optimization known as *partition pruning*. The core concept behind partition pruning is relatively simple, and can be described as “Do not scan partitions where there can be no matching values”. Suppose that you have a partitioned table t1 defined by this statement:

```
CREATE TABLE t1 (
    fname VARCHAR(50) NOT NULL,
    lname VARCHAR(50) NOT NULL,
    region_code TINYINT UNSIGNED NOT NULL,
    dob DATE NOT NULL
)
PARTITION BY RANGE( region_code ) (
    PARTITION p0 VALUES LESS THAN (64),
    PARTITION p1 VALUES LESS THAN (128),
    PARTITION p2 VALUES LESS THAN (192),
    PARTITION p3 VALUES LESS THAN MAXVALUE
);
```

Consider the case where you wish to obtain results from a SELECT statement such as this one:

```
SELECT fname, lname, region_code, dob
    FROM t1
    WHERE region_code > 125 AND region_code < 130;
```

It is easy to see that none of the rows which ought to be returned will be in either of the partitions p0 or p3; that is, we need to search only in partitions p1 and p2 to find matching rows. By doing so, it is possible to expend much less time and effort in finding matching rows than would be required to scan all partitions in the table. This “cutting away” of unneeded partitions is known as *pruning*. When the optimizer can make use of partition pruning in performing this query, execution of the query can be an order of magnitude faster than the same query against a nonpartitioned table containing the same column definitions and data.

The optimizer can perform pruning whenever a WHERE condition can be reduced to either one of the following two cases:

- *partition_column* = *constant*
- *partition_column* IN (*constant1*, *constant2*, ..., *constantN*)

In the first case, the optimizer simply evaluates the partitioning expression for the value given, determines which partition contains that value, and scans only this partition. In many cases, the equal sign can be replaced with another arithmetic comparison, including <, >, <=, >=, and <>. Some queries using BETWEEN in the WHERE clause can also take advantage of partition pruning. See the examples later in this section.

In the second case, the optimizer evaluates the partitioning expression for each value in the list, creates a list of matching partitions, and then scans only the partitions in this partition list.

MySQL can apply partition pruning to `SELECT`, `DELETE`, and `UPDATE` statements. `INSERT` statements currently cannot be pruned.

Pruning can also be applied to short ranges, which the optimizer can convert into equivalent lists of values. For instance, in the previous example, the `WHERE` clause can be converted to `WHERE region_code IN (126, 127, 128, 129)`. Then the optimizer can determine that the first three values in the list are found in partition `p1`, the remaining three values in partition `p2`, and that the other partitions contain no relevant values and so do not need to be searched for matching rows.

Yhe optimizer can also perform pruning for `WHERE` conditions that involve comparisons of the preceding types on multiple columns for tables that use `RANGE COLUMNS` or `LIST COLUMNS` partitioning.

This type of optimization can be applied whenever the partitioning expression consists of an equality or a range which can be reduced to a set of equalities, or when the partitioning expression represents an increasing or decreasing relationship. Pruning can also be applied for tables partitioned on a `DATE` or `DATETIME` column when the partitioning expression uses the `YEAR()` or `TO_DAYS()` function. In addition, in MySQL 5.7, pruning can be applied for such tables when the partitioning expression uses the `TO_SECONDS()` function.

Suppose that table `t2`, defined as shown here, is partitioned on a `DATE` column:

```
CREATE TABLE t2 (
    fname VARCHAR(50) NOT NULL,
    lname VARCHAR(50) NOT NULL,
    region_code TINYINT UNSIGNED NOT NULL,
    dob DATE NOT NULL
)
PARTITION BY RANGE( YEAR(dob) ) (
    PARTITION d0 VALUES LESS THAN (1970),
    PARTITION d1 VALUES LESS THAN (1975),
    PARTITION d2 VALUES LESS THAN (1980),
    PARTITION d3 VALUES LESS THAN (1985),
    PARTITION d4 VALUES LESS THAN (1990),
    PARTITION d5 VALUES LESS THAN (2000),
    PARTITION d6 VALUES LESS THAN (2005),
    PARTITION d7 VALUES LESS THAN MAXVALUE
);
```

The following statements using `t2` can make of use partition pruning:

```
SELECT * FROM t2 WHERE dob = '1982-06-23';

UPDATE t2 SET region_code = 8 WHERE dob BETWEEN '1991-02-15' AND '1997-04-25';

DELETE FROM t2 WHERE dob >= '1984-06-21' AND dob <= '1999-06-21'
```

In the case of the last statement, the optimizer can also act as follows:

1. *Find the partition containing the low end of the range*.

   `YEAR('1984-06-21')` yields the value `1984`, which is found in partition `d3`.

2. *Find the partition containing the high end of the range*.

   `YEAR('1999-06-21')` evaluates to `1999`, which is found in partition `d5`.

3. *Scan only these two partitions and any partitions that may lie between them*.

   In this case, this means that only partitions `d3`, `d4`, and `d5` are scanned. The remaining partitions may be safely ignored (and are ignored).

> **Important**
>
> Invalid `DATE` and `DATETIME` values referenced in the `WHERE` condition of a statement against a partitioned table are treated as `NULL`. This means that a query such as `SELECT * FROM partitioned_table WHERE date_column < '2008-12-00'` does not return any values (see Bug #40972).

So far, we have looked only at examples using `RANGE` partitioning, but pruning can be applied with other partitioning types as well.

Consider a table that is partitioned by `LIST`, where the partitioning expression is increasing or decreasing, such as the table `t3` shown here. (In this example, we assume for the sake of brevity that the `region_code` column is limited to values between 1 and 10 inclusive.)

```
CREATE TABLE t3 (
    fname VARCHAR(50) NOT NULL,
    lname VARCHAR(50) NOT NULL,
    region_code TINYINT UNSIGNED NOT NULL,
    dob DATE NOT NULL
)
PARTITION BY LIST(region_code) (
    PARTITION r0 VALUES IN (1, 3),
    PARTITION r1 VALUES IN (2, 5, 8),
    PARTITION r2 VALUES IN (4, 9),
    PARTITION r3 VALUES IN (6, 7, 10)
);
```

For a statement such as `SELECT * FROM t3 WHERE region_code BETWEEN 1 AND 3`, the optimizer determines in which partitions the values 1, 2, and 3 are found (`r0` and `r1`) and skips the remaining ones (`r2` and `r3`).

For tables that are partitioned by `HASH` or `KEY`, partition pruning is also possible in cases in which the `WHERE` clause uses a simple `=` relation against a column used in the partitioning expression. Consider a table created like this:

```
CREATE TABLE t4 (
    fname VARCHAR(50) NOT NULL,
    lname VARCHAR(50) NOT NULL,
    region_code TINYINT UNSIGNED NOT NULL,
    dob DATE NOT NULL
)
PARTITION BY KEY(region_code)
PARTITIONS 8;
```

A statement that compares a column value with a constant can be pruned:

```
UPDATE t4 WHERE region_code = 7;
```

Pruning can also be employed for short ranges, because the optimizer can turn such conditions into `IN` relations. For example, using the same table `t4` as defined previously, queries such as these can be pruned:

```
SELECT * FROM t4 WHERE region_code > 2 AND region_code < 6;

SELECT * FROM t4 WHERE region_code BETWEEN 3 AND 5;
```

In both these cases, the `WHERE` clause is transformed by the optimizer into `WHERE region_code IN (3, 4, 5)`.

> **Important**
>
> This optimization is used only if the range size is smaller than the number of partitions. Consider this statement:
>
> ```
> DELETE FROM t4 WHERE region_code BETWEEN 4 AND 12;
> ```
>
> The range in the `WHERE` clause covers 9 values (4, 5, 6, 7, 8, 9, 10, 11, 12), but `t4` has only 8 partitions. This means that the `DELETE` cannot be pruned.

When a table is partitioned by `HASH` or `KEY`, pruning can be used only on integer columns. For example, this statement cannot use pruning because `dob` is a `DATE` column:

```
SELECT * FROM t4 WHERE dob >= '2001-04-14' AND dob <= '2005-10-15';
```

However, if the table stores year values in an `INT` column, then a query having `WHERE year_col >= 2001 AND year_col <= 2005` can be pruned.

Prior to MySQL 5.7.1, partition pruning was disabled for all tables using a storage that provides automatic partitioning, such as the `NDB` storage engine used by MySQL Cluster (not currently supported in MySQL 5.7). (Bug #14672885) Beginning with MySQL 5.7.1, such tables can be pruned if they are explicitly partitioned. (Bug #14827952)

17.5 Partition Selection

MySQL 5.7 supports explicit selection of partitions and subpartitions that, when executing a statement, should be checked for rows matching a given `WHERE` condition. Partition selection is similar to partition pruning, in that only specific partitions are checked for matches, but differs in two key respects:

1. The partitions to be checked are specified by the issuer of the statement, unlike partition pruning, which is automatic.

2. Whereas partition pruning applies only to queries, explicit selection of partitions is supported for both queries and a number of DML statements.

SQL statements supporting explicit partition selection are listed here:

- `SELECT`
- `DELETE`
- `INSERT`
- `REPLACE`
- `UPDATE`
- `LOAD DATA`.

- LOAD XML.

The remainder of this section discusses explicit partition selection as it applies generally to the statements just listed, and provides some examples.

Explicit partition selection is implemented using a PARTITION option. For all supported statements, this option uses the syntax shown here:

```
      PARTITION (partition_names)

      partition_names:
          partition_name, ...
```

This option always follows the name of the table to which the partition or partitions belong. *partition_names* is a comma-separated list of partitions or subpartitions to be used. Each name in this list must be the name of an existing partition or subpartition of the specified table; if any of the partitions or subpartitions are not found, the statement fails with an error (partition '*partition_name*' doesn't exist). Partitions and subpartitions named in *partition_names* may be listed in any order, and may overlap.

When the PARTITION option is used, only the partitions and subpartitions listed are checked for matching rows. This option can be used in a SELECT statement to determine which rows belong to a given partition. Consider a partitioned table named employees, created and populated using the statements shown here:

```
SET @@SQL_MODE = '';

CREATE TABLE employees  (
    id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    fname VARCHAR(25) NOT NULL,
    lname VARCHAR(25) NOT NULL,
    store_id INT NOT NULL,
    department_id INT NOT NULL
)
    PARTITION BY RANGE(id)  (
        PARTITION p0 VALUES LESS THAN (5),
        PARTITION p1 VALUES LESS THAN (10),
        PARTITION p2 VALUES LESS THAN (15),
        PARTITION p3 VALUES LESS THAN MAXVALUE
);

INSERT INTO employees VALUES
    ('', 'Bob', 'Taylor', 3, 2), ('', 'Frank', 'Williams', 1, 2),
    ('', 'Ellen', 'Johnson', 3, 4), ('', 'Jim', 'Smith', 2, 4),
    ('', 'Mary', 'Jones', 1, 1), ('', 'Linda', 'Black', 2, 3),
    ('', 'Ed', 'Jones', 2, 1), ('', 'June', 'Wilson', 3, 1),
    ('', 'Andy', 'Smith', 1, 3), ('', 'Lou', 'Waters', 2, 4),
    ('', 'Jill', 'Stone', 1, 4), ('', 'Roger', 'White', 3, 2),
    ('', 'Howard', 'Andrews', 1, 2), ('', 'Fred', 'Goldberg', 3, 3),
    ('', 'Barbara', 'Brown', 2, 3), ('', 'Alice', 'Rogers', 2, 2),
    ('', 'Mark', 'Morgan', 3, 3), ('', 'Karen', 'Cole', 3, 2);
```

You can see which rows are stored in partition p1 like this:

```
mysql> SELECT * FROM employees PARTITION (p1);
+----+-------+--------+----------+---------------+
| id | fname | lname  | store_id | department_id |
+----+-------+--------+----------+---------------+
|  5 | Mary  | Jones  |        1 |             1 |
|  6 | Linda | Black  |        2 |             3 |
|  7 | Ed    | Jones  |        2 |             1 |
|  8 | June   | Wilson |        3 |               1 |
|  9 | Andy   | Smith  |        1 |               3 |
+----+-------+--------+----------+---------------+
5 rows in set (0.00 sec)
```

The result is the same as obtained by the query `SELECT * FROM employees WHERE id BETWEEN 5 AND 9`.

To obtain rows from multiple partitions, supply their names as a comma-delimited list. For example, `SELECT * FROM employees PARTITION (p1, p2)` returns all rows from partitions `p1` and `p2` while excluding rows from the remaining partitions.

Any valid query against a partitioned table can be rewritten with a `PARTITION` option to restrict the result to one or more desired partitions. You can use `WHERE` conditions, `ORDER BY` and `LIMIT` options, and so on. You can also use aggregate functions with `HAVING` and `GROUP BY` options. Each of the following queries produces a valid result when run on the `employees` table as previously defined:

```
mysql> SELECT * FROM employees PARTITION (p0, p2)
    ->     WHERE lname LIKE 'S%';
+----+-------+-------+----------+---------------+
| id | fname | lname | store_id | department_id |
+----+-------+-------+----------+---------------+
|  4 | Jim   | Smith |        2 |             4 |
| 11 | Jill  | Stone |        1 |             4 |
+----+-------+-------+----------+---------------+
2 rows in set (0.00 sec)

mysql> SELECT id, CONCAT(fname, ' ', lname) AS name
    ->     FROM employees PARTITION (p0) ORDER BY lname;
+----+----------------+
| id | name           |
+----+----------------+
|  3 | Ellen Johnson  |
|  4 | Jim Smith      |
|  1 | Bob Taylor     |
|  2 | Frank Williams |
+----+----------------+
4 rows in set (0.06 sec)

mysql> SELECT store_id, COUNT(department_id) AS c
    ->     FROM employees PARTITION (p1,p2,p3)
    ->     GROUP BY store_id HAVING c > 4;
+---+----------+
| c | store_id |
+---+----------+
| 5 |        2 |
| 5 |        3 |
+---+----------+
2 rows in set (0.00 sec)
```

Statements using partition selection can be employed with tables using any of the partitioning types supported in MySQL 5.7. When a table is created using `[LINEAR] HASH` or `[LINEAR] KEY` partitioning and the names of the partitions are not specified, MySQL automatically names the partitions `p0`, `p1`, `p2`, ..., `pN-1`, where `N` is the number of partitions. For subpartitions not explicitly named, MySQL assigns automatically to the subpartitions in each partition `pX` the names `pXsp0`, `pXsp1`, `pXsp2`, ..., `pXspM-1`, where `M` is the number of subpartitions. When executing against this table a `SELECT` (or other SQL statement for which explicit partition selection is allowed), you can use these generated names in a `PARTITION` option, as shown here:

```
mysql> CREATE TABLE employees_sub  (
    ->     id INT NOT NULL AUTO_INCREMENT,
    ->     fname VARCHAR(25) NOT NULL,
    ->     lname VARCHAR(25) NOT NULL,
    ->     store_id INT NOT NULL,
    ->     department_id INT NOT NULL,
    ->     PRIMARY KEY pk (id, lname)
    -> )
    ->     PARTITION BY RANGE(id)
    ->     SUBPARTITION BY KEY (lname)
    ->     SUBPARTITIONS 2 (
    ->         PARTITION p0 VALUES LESS THAN (5),
    ->         PARTITION p1 VALUES LESS THAN (10),
    ->         PARTITION p2 VALUES LESS THAN (15),
    ->         PARTITION p3 VALUES LESS THAN MAXVALUE
    -> );
Query OK, 0 rows affected (1.14 sec)

mysql> INSERT INTO employees_sub   # re-use data in employees table
    ->     SELECT * FROM employees;
Query OK, 18 rows affected (0.09 sec)
Records: 18  Duplicates: 0  Warnings: 0

mysql> SELECT id, CONCAT(fname, ' ', lname) AS name
    ->     FROM employees_sub PARTITION (p2sp1);
+----+---------------+
| id | name          |
+----+---------------+
| 10 | Lou Waters    |
| 14 | Fred Goldberg |
+----+---------------+
2 rows in set (0.00 sec)
```

You may also use a `PARTITION` option in the `SELECT` portion of an `INSERT ... SELECT` statement, as shown here:

```
mysql> CREATE TABLE employees_copy LIKE employees;
Query OK, 0 rows affected (0.28 sec)

mysql> INSERT INTO employees_copy
    ->     SELECT * FROM employees PARTITION (p2);
Query OK, 5 rows affected (0.04 sec)
Records: 5  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM employees_copy;
+----+--------+----------+----------+---------------+
| id | fname  | lname    | store_id | department_id |
+----+--------+----------+----------+---------------+
| 10 | Lou    | Waters   |        2 |             4 |
| 11 | Jill   | Stone    |        1 |             4 |
| 12 | Roger  | White    |        3 |             2 |
| 13 | Howard | Andrews  |        1 |             2 |
| 14 | Fred   | Goldberg |        3 |             3 |
+----+--------+----------+----------+---------------+
5 rows in set (0.00 sec)
```

Partition selection can also be used with joins. Suppose we create and populate two tables using the statements shown here:

```
CREATE TABLE stores (
    id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    city VARCHAR(30) NOT NULL
)
    PARTITION BY HASH(id)
    PARTITIONS 2;
```

```
INSERT INTO stores VALUES
    ('', 'Nambucca'), ('', 'Uranga'),
    ('', 'Bellingen'), ('', 'Grafton');

CREATE TABLE departments  (
    id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(30) NOT NULL
)
    PARTITION BY KEY(id)
    PARTITIONS 2;

INSERT INTO departments VALUES
    ('', 'Sales'), ('', 'Customer Service'),
    ('', 'Delivery'), ('', 'Accounting');
```

You can explicitly select partitions (or subpartitions, or both) from any or all of the tables in a join. (Note that the `PARTITION` option used to select partitions from a given table immediately follows the name of the table, before all other options, including any table alias.) For example, the following query gets the name, employee ID, department, and city of all employees who work in the Sales or Delivery department (partition `p1` of the `departments` table) at the stores in either of the cities of Nambucca and Bellingen (partition `p0` of the `stores` table):

```
mysql> SELECT
    ->     e.id AS 'Employee ID', CONCAT(e.fname, ' ', e.lname) AS Name,
    ->     s.city AS City, d.name AS department
    -> FROM employees AS e
    ->     JOIN stores PARTITION (p1) AS s ON e.store_id=s.id
    ->     JOIN departments PARTITION (p0) AS d ON e.department_id=d.id
    -> ORDER BY e.lname;
+-------------+---------------+-----------+------------+
| Employee ID | Name          | City      | department |
+-------------+---------------+-----------+------------+
|          14 | Fred Goldberg | Bellingen | Delivery   |
|           5 | Mary Jones    | Nambucca  | Sales      |
|          17 | Mark Morgan   | Bellingen | Delivery   |
|           9 | Andy Smith    | Nambucca  | Delivery   |
|           8 | June Wilson   | Bellingen | Sales      |
+-------------+---------------+-----------+------------+
5 rows in set (0.00 sec)
```

For general information about joins in MySQL, see Section 13.2.9.2, “`JOIN` Syntax”.

When the `PARTITION` option is used with `DELETE` statements, only those partitions (and subpartitions, if any) listed with the option are checked for rows to be deleted. Any other partitions are ignored, as shown here:

```
mysql> SELECT * FROM employees WHERE fname LIKE 'j%';
+----+-------+--------+----------+---------------+
| id | fname | lname  | store_id | department_id |
+----+-------+--------+----------+---------------+
|  4 | Jim   | Smith  |        2 |             4 |
|  8 | June  | Wilson |        3 |             1 |
| 11 | Jill  | Stone  |        1 |             4 |
+----+-------+--------+----------+---------------+
3 rows in set (0.00 sec)

mysql> DELETE FROM employees PARTITION (p0, p1)
    ->     WHERE fname LIKE 'j%';
Query OK, 2 rows affected (0.09 sec)

mysql> SELECT * FROM employees WHERE fname LIKE 'j%';
+----+-------+-------+----------+---------------+
| id | fname | lname | store_id | department_id |
+----+-------+-------+----------+---------------+
| 11 | Jill  | Stone |        1 |             4 |
+----+-------+-------+----------+---------------+
1 row in set (0.00 sec)
```

Only the two rows in partitions p0 and p1 matching the WHERE condition were deleted. As you can see from the result when the SELECT is run a second time, there remains a row in the table matching the WHERE condition, but residing in a different partition (p2).

UPDATE statements using explicit partition selection behave in the same way; only rows in the partitions referenced by the PARTITION option are considered when determining the rows to be updated, as can be seen by executing the following statements:

```
mysql> UPDATE employees PARTITION (p0)
    ->     SET store_id = 2 WHERE fname = 'Jill';
Query OK, 0 rows affected (0.00 sec)
Rows matched: 0  Changed: 0  Warnings: 0

mysql> SELECT * FROM employees WHERE fname = 'Jill';
+----+-------+-------+----------+---------------+
| id | fname | lname | store_id | department_id |
+----+-------+-------+----------+---------------+
| 11 | Jill  | Stone |        1 |             4 |
+----+-------+-------+----------+---------------+
1 row in set (0.00 sec)

mysql> UPDATE employees PARTITION (p2)
    ->     SET store_id = 2 WHERE fname = 'Jill';
Query OK, 1 row affected (0.09 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT * FROM employees WHERE fname = 'Jill';
+----+-------+-------+----------+---------------+
| id | fname | lname | store_id | department_id |
+----+-------+-------+----------+---------------+
| 11 | Jill  | Stone |        2 |             4 |
+----+-------+-------+----------+---------------+
1 row in set (0.00 sec)
```

In the same way, when PARTITION is used with DELETE, only rows in the partition or partitions named in the partition list are checked for deletion.

For statements that insert rows, the behavior differs in that failure to find a suitable partition causes the statement to fail. This is true for both INSERT and REPLACE statements, as shown here:

```
mysql> INSERT INTO employees PARTITION (p2) VALUES (20, 'Jan', 'Jones', 1, 3);
ERROR 1729 (HY000): Found a row not matching the given partition set
mysql> INSERT INTO employees PARTITION (p3) VALUES (20, 'Jan', 'Jones', 1, 3);
Query OK, 1 row affected (0.07 sec)

mysql> REPLACE INTO employees PARTITION (p0) VALUES (20, 'Jan', 'Jones', 3, 2);
ERROR 1729 (HY000): Found a row not matching the given partition set

mysql> REPLACE INTO employees PARTITION (p3) VALUES (20, 'Jan', 'Jones', 3, 2);
Query OK, 2 rows affected (0.09 sec)
```

For statements that write multiple rows to a partitioned table that uses the InnoDB storage engine: If any row in the list following VALUES cannot be written to one of the partitions specified in the *partition_names* list, the entire statement fails and no rows are written. This is shown for INSERT statements in the following example, reusing the employees table created previously:

```
mysql> ALTER TABLE employees
    ->     REORGANIZE PARTITION p3 INTO (
    ->         PARTITION p3 VALUES LESS THAN (20),
    ->         PARTITION p4 VALUES LESS THAN (25),
    ->         PARTITION p5 VALUES LESS THAN MAXVALUE
    ->     );
Query OK, 6 rows affected (2.09 sec)
Records: 6  Duplicates: 0  Warnings: 0

mysql> SHOW CREATE TABLE employees\G
*************************** 1. row ***************************
       Table: employees
Create Table: CREATE TABLE `employees` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `fname` varchar(25) NOT NULL,
  `lname` varchar(25) NOT NULL,
  `store_id` int(11) NOT NULL,
  `department_id` int(11) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=27 DEFAULT CHARSET=latin1
/*!50100 PARTITION BY RANGE (id)
(PARTITION p0 VALUES LESS THAN (5) ENGINE = InnoDB,
 PARTITION p1 VALUES LESS THAN (10) ENGINE = InnoDB,
 PARTITION p2 VALUES LESS THAN (15) ENGINE = InnoDB,
 PARTITION p3 VALUES LESS THAN (20) ENGINE = InnoDB,
 PARTITION p4 VALUES LESS THAN (25) ENGINE = InnoDB,
 PARTITION p5 VALUES LESS THAN MAXVALUE ENGINE = InnoDB) */
1 row in set (0.00 sec)

mysql> INSERT INTO employees PARTITION (p3, p4) VALUES
    ->     (24, 'Tim', 'Greene', 3, 1),  (26, 'Linda', 'Mills', 2, 1);
ERROR 1729 (HY000): Found a row not matching the given partition set

mysql> INSERT INTO employees PARTITION (p3, p4. p5) VALUES
    ->     (24, 'Tim', 'Greene', 3, 1),  (26, 'Linda', 'Mills', 2, 1);
Query OK, 2 rows affected (0.06 sec)
Records: 2  Duplicates: 0  Warnings: 0
```

The preceding is true for both `INSERT` statements and `REPLACE` statements that write multiple rows.

In MySQL 5.7.1 and later, partition selection is disabled for tables employing a storage engine that supplies automatic partitioning, such as `NDB`. (Bug #14827952)

17.6 Restrictions and Limitations on Partitioning

This section discusses current restrictions and limitations on MySQL partitioning support.

Prohibited constructs. The following constructs are not permitted in partitioning expressions:

- Stored procedures, stored functions, UDFs, or plugins.
- Declared variables or user variables.

For a list of SQL functions which are permitted in partitioning expressions, see Section 17.6.3, “Partitioning Limitations Relating to Functions”.

Arithmetic and logical operators. Use of the arithmetic operators `+`, `-`, and `*` is permitted in partitioning expressions. However, the result must be an integer value or `NULL` (except in the case of `[LINEAR] KEY` partitioning, as discussed elsewhere in this chapter; see Section 17.2, “Partitioning Types”, for more information).

The `DIV` operator is also supported, and the `/` operator is not permitted. (Bug #30188, Bug #33182)

The bit operators |, &, ^, <<, >>, and ~ are not permitted in partitioning expressions.

HANDLER statements. Previously, the HANDLER statement was not supported with partitioned tables. This limitation is removed beginning with MySQL 5.7.1.

Server SQL mode. Tables employing user-defined partitioning do not preserve the SQL mode in effect at the time that they were created. As discussed in Section 5.1.7, “Server SQL Modes”, the results of many MySQL functions and operators may change according to the server SQL mode. Therefore, a change in the SQL mode at any time after the creation of partitioned tables may lead to major changes in the behavior of such tables, and could easily lead to corruption or loss of data. For these reasons, *it is strongly recommended that you never change the server SQL mode after creating partitioned tables*.

Examples. The following examples illustrate some changes in behavior of partitioned tables due to a change in the server SQL mode:

1. **Error handling.** Suppose that you create a partitioned table whose partitioning expression is one such as *column* DIV 0 or *column* MOD 0, as shown here:

```
mysql> CREATE TABLE tn (c1 INT)
    ->     PARTITION BY LIST(1 DIV c1) (
    ->       PARTITION p0 VALUES IN (NULL),
    ->       PARTITION p1 VALUES IN (1)
    -> );
Query OK, 0 rows affected (0.05 sec)
```

   The default behavior for MySQL is to return NULL for the result of a division by zero, without producing any errors:

```
mysql> SELECT @@sql_mode;
+------------+
| @@sql_mode |
+------------+
|            |
+------------+
1 row in set (0.00 sec)


mysql> INSERT INTO tn VALUES (NULL), (0), (1);
Query OK, 3 rows affected (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 0
```

   However, changing the server SQL mode to treat division by zero as an error and to enforce strict error handling causes the same INSERT statement to fail, as shown here:

```
mysql> SET sql_mode='STRICT_ALL_TABLES,ERROR_FOR_DIVISION_BY_ZERO';
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO tn VALUES (NULL), (0), (1);
ERROR 1365 (22012): Division by 0
```

   As of MySQL 5.7.4, strict mode includes the effect of ERROR_FOR_DIVISION_BY_ZERO, so that mode need not be named explicitly when setting sql_mode.

2. **Table accessibility.** Sometimes a change in the server SQL mode can make partitioned tables unusable. The following CREATE TABLE statement can be executed successfully only if the NO_UNSIGNED_SUBTRACTION mode is in effect:

```
mysql> SELECT @@sql_mode;
```

```
+------------+
| @@sql_mode |
+------------+
|            |
+------------+
1 row in set (0.00 sec)

mysql> CREATE TABLE tu (c1 BIGINT UNSIGNED)
    ->   PARTITION BY RANGE(c1 - 10) (
    ->     PARTITION p0 VALUES LESS THAN (-5),
    ->     PARTITION p1 VALUES LESS THAN (0),
    ->     PARTITION p2 VALUES LESS THAN (5),
    ->     PARTITION p3 VALUES LESS THAN (10),
    ->     PARTITION p4 VALUES LESS THAN (MAXVALUE)
    -> );
ERROR 1563 (HY000): Partition constant is out of partition function domain

mysql> SET sql_mode='NO_UNSIGNED_SUBTRACTION';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @@sql_mode;
+-------------------------+
| @@sql_mode              |
+-------------------------+
| NO_UNSIGNED_SUBTRACTION |
+-------------------------+
1 row in set (0.00 sec)

mysql> CREATE TABLE tu (c1 BIGINT UNSIGNED)
    ->   PARTITION BY RANGE(c1 - 10) (
    ->     PARTITION p0 VALUES LESS THAN (-5),
    ->     PARTITION p1 VALUES LESS THAN (0),
    ->     PARTITION p2 VALUES LESS THAN (5),
    ->     PARTITION p3 VALUES LESS THAN (10),
    ->     PARTITION p4 VALUES LESS THAN (MAXVALUE)
    -> );
Query OK, 0 rows affected (0.05 sec)
```

If you remove the NO_UNSIGNED_SUBTRACTION server SQL mode after creating tu, you may no longer be able to access this table:

```
mysql> SET sql_mode='';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM tu;
ERROR 1563 (HY000): Partition constant is out of partition function domain
mysql> INSERT INTO tu VALUES (20);
ERROR 1563 (HY000): Partition constant is out of partition function domain
```

Server SQL modes also impact replication of partitioned tables. Differing SQL modes on master and slave can lead to partitioning expressions being evaluated differently; this can cause the distribution of data among partitions to be different in the master's and slave's copies of a given table, and may even cause inserts into partitioned tables that succeed on the master to fail on the slave. For best results, you should always use the same server SQL mode on the master and on the slave.

Performance considerations. Some affects of partitioning operations on performance are given in the following list:

- **File system operations.** Partitioning and repartitioning operations (such as ALTER TABLE with PARTITION BY ..., REORGANIZE PARTITIONS, or REMOVE PARTITIONING) depend on file system operations for their implementation. This means that the speed of these operations is affected by such factors as file system type and characteristics, disk speed, swap space, file handling efficiency of the operating system, and MySQL server options and variables that relate to file handling. In particular, you should make sure that `large_files_support` is enabled and that `open_files_limit` is set properly. For partitioned tables using the `MyISAM` storage engine, increasing `myisam_max_sort_file_size` may improve performance; partitioning and repartitioning operations involving `InnoDB` tables may be made more efficient by enabling `innodb_file_per_table`.

See also Maximum number of partitions.

- **Table locks.** The process executing a partitioning operation on a table takes a write lock on the table. Reads from such tables are relatively unaffected; pending `INSERT` and `UPDATE` operations are performed as soon as the partitioning operation has completed.

- **Storage engine.** Partitioning operations, queries, and update operations generally tend to be faster with `MyISAM` tables than with `InnoDB` tables.

- **Indexes; partition pruning.** As with nonpartitioned tables, proper use of indexes can speed up queries on partitioned tables significantly. In addition, designing partitioned tables and queries on these tables to take advantage of *partition pruning* can improve performance dramatically. See Section 17.4, “Partition Pruning”, for more information.

  Previously, index condition pushdown was not supported for partitioned tables. This limitation was removed in MySQL 5.7.3. See Section 8.2.1.6, “Index Condition Pushdown Optimization”.

- **Performance with `LOAD DATA`.** In MySQL 5.7, `LOAD DATA` uses buffering to improve performance. You should be aware that the buffer uses 130 KB memory per partition to achieve this.

Maximum number of partitions.
In MySQL 5.7, the maximum possible number of partitions for a given table is 8192. This number includes subpartitions.

If, when creating tables with a large number of partitions (but less than the maximum), you encounter an error message such as `Got error ... from storage engine: Out of resources when opening file`, you may be able to address the issue by increasing the value of the `open_files_limit` system variable. However, this is dependent on the operating system, and may not be possible or advisable on all platforms; see Section C.5.2.18, “`'`*`File`*`' Not Found` and Similar Errors”, for more information. In some cases, using large numbers (hundreds) of partitions may also not be advisable due to other concerns, so using more partitions does not automatically lead to better results.

See also File system operations.

Query cache not supported.
The query cache is not supported for partitioned tables, and is automatically disabled for queries involving partitioned tables. The query cache cannot be enabled for such queries.

Per-partition key caches.
In MySQL 5.7, key caches are supported for partitioned `MyISAM` tables, using the `CACHE INDEX` and `LOAD INDEX INTO CACHE` statements. Key caches may be defined for one, several, or all partitions, and indexes for one, several, or all partitions may be preloaded into key caches.

Foreign keys not supported for partitioned `InnoDB` tables.
Partitioned tables using the `InnoDB` storage engine do not support foreign keys. More specifically, this means that the following two statements are true:

1. No definition of an `InnoDB` table employing user-defined partitioning may contain foreign key references; no `InnoDB` table whose definition contains foreign key references may be partitioned.

2. No `InnoDB` table definition may contain a foreign key reference to a user-partitioned table; no `InnoDB` table with user-defined partitioning may contain columns referenced by foreign keys.

The scope of the restrictions just listed includes all tables that use the InnoDB storage engine. CREATE TABLE and ALTER TABLE statements that would result in tables violating these restrictions are not allowed.

ALTER TABLE ... ORDER BY. An ALTER TABLE ... ORDER BY *column* statement run against a partitioned table causes ordering of rows only within each partition.

FULLTEXT indexes.
Partitioned tables do not support FULLTEXT indexes or searches, even for partitioned tables employing the InnoDB or MyISAM storage engine.

Spatial columns. Columns with spatial data types such as POINT or GEOMETRY cannot be used in partitioned tables.

Temporary tables.
Temporary tables cannot be partitioned. (Bug #17497)

Log tables. It is not possible to partition the log tables; an ALTER TABLE ... PARTITION BY ... statement on such a table fails with an error.

Data type of partitioning key.
A partitioning key must be either an integer column or an expression that resolves to an integer. The column or expression value may also be NULL. (See Section 17.2.7, “How MySQL Partitioning Handles NULL”.)

There are two exceptions to this restriction:

1. When partitioning by [LINEAR] KEY, it is possible to use columns of other types as partitioning keys, because MySQL's internal key-hashing functions produce the correct data type from these types. For example, the following CREATE TABLE statement is valid:

   ```
   CREATE TABLE tkc (c1 CHAR)
   PARTITION BY KEY(c1)
   PARTITIONS 4;
   ```

2. When partitioning by RANGE COLUMNS or LIST COLUMNS, it is possible to use string, DATE, and DATETIME columns. For example, each of the following CREATE TABLE statements is valid:

   ```
   CREATE TABLE rc (c1 INT, c2 DATE)
   PARTITION BY RANGE COLUMNS(c2) (
       PARTITION p0 VALUES LESS THAN('1990-01-01'),
       PARTITION p1 VALUES LESS THAN('1995-01-01'),
       PARTITION p2 VALUES LESS THAN('2000-01-01'),
       PARTITION p3 VALUES LESS THAN('2005-01-01'),
       PARTITION p4 VALUES LESS THAN(MAXVALUE)
   );

   CREATE TABLE lc (c1 INT, c2 CHAR(1))
   PARTITION BY LIST COLUMNS(c2) (
       PARTITION p0 VALUES IN('a', 'd', 'g', 'j', 'm', 'p', 's', 'v', 'y'),
       PARTITION p1 VALUES IN('b', 'e', 'h', 'k', 'n', 'q', 't', 'w', 'z'),
       PARTITION p2 VALUES IN('c', 'f', 'i', 'l', 'o', 'r', 'u', 'x', NULL)
   );
   ```

Neither of the preceding exceptions applies to BLOB or TEXT column types.

Subqueries.
A partitioning key may not be a subquery, even if that subquery resolves to an integer value or NULL.

Issues with subpartitions.
Subpartitions must use HASH or KEY partitioning. Only RANGE and LIST partitions may be subpartitioned; HASH and KEY partitions cannot be subpartitioned.

Currently, SUBPARTITION BY KEY requires that the subpartitioning column or columns be specified explicitly, unlike the case with PARTITION BY KEY, where it can be omitted (in which case the table's primary key column is used by default). Consider the table created by this statement:

```
CREATE TABLE ts (
    id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(30)
);
```

You can create a table having the same columns, partitioned by KEY, using a statement such as this one:

```
CREATE TABLE ts (
    id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(30)
)
PARTITION BY KEY()
PARTITIONS 4;
```

The previous statement is treated as though it had been written like this, with the table's primary key column used as the partitioning column:

```
CREATE TABLE ts (
    id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(30)
)
PARTITION BY KEY(id)
PARTITIONS 4;
```

However, the following statement that attempts to create a subpartitioned table using the default column as the subpartitioning column fails, and the column must be specified for the statement to succeed, as shown here:

```
mysql> CREATE TABLE ts (
    ->     id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    ->     name VARCHAR(30)
    -> )
    -> PARTITION BY RANGE(id)
    -> SUBPARTITION BY KEY()
    -> SUBPARTITIONS 4
    -> (
    ->     PARTITION p0 VALUES LESS THAN (100),
    ->     PARTITION p1 VALUES LESS THAN (MAXVALUE)
    -> );
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that
corresponds to your MySQL server version for the right syntax to use near ')

mysql> CREATE TABLE ts (
    ->     id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    ->     name VARCHAR(30)
    -> )
    -> PARTITION BY RANGE(id)
    -> SUBPARTITION BY KEY(id)
    -> SUBPARTITIONS 4
    -> (
    ->     PARTITION p0 VALUES LESS THAN (100),
    ->     PARTITION p1 VALUES LESS THAN (MAXVALUE)
    -> );
Query OK, 0 rows affected (0.07 sec)
```

This is a known issue (see Bug #51470).

DATA DIRECTORY and INDEX DIRECTORY options. DATA DIRECTORY and INDEX DIRECTORY are subject to the following restrictions when used with partitioned tables:

- Table-level DATA DIRECTORY and INDEX DIRECTORY options are ignored (see Bug #32091).

- On Windows, the DATA DIRECTORY and INDEX DIRECTORY options are not supported for individual partitions or subpartitions of MyISAM tables. However, you can use DATA DIRECTORY for individual partitions or subpartitions of InnoDB tables.

Repairing and rebuilding partitioned tables. The statements CHECK TABLE, OPTIMIZE TABLE, ANALYZE TABLE, and REPAIR TABLE are supported for partitioned tables.

In addition, you can use ALTER TABLE ... REBUILD PARTITION to rebuild one or more partitions of a partitioned table; ALTER TABLE ... REORGANIZE PARTITION also causes partitions to be rebuilt. See Section 13.1.6, “ALTER TABLE Syntax”, for more information about these two statements.

mysqlcheck, myisamchk, and myisampack are not supported with partitioned tables.

FOR EXPORT option (FLUSH TABLES). The FLUSH TABLES statement's FOR EXPORT option is not supported for for partitioned InnoDB tables in MySQL 5.7.4 and earlier. (Bug #16943907)

17.6.1 Partitioning Keys, Primary Keys, and Unique Keys

This section discusses the relationship of partitioning keys with primary keys and unique keys. The rule governing this relationship can be expressed as follows: All columns used in the partitioning expression for a partitioned table must be part of every unique key that the table may have.

In other words, *every unique key on the table must use every column in the table's partitioning expression*. (This also includes the table's primary key, since it is by definition a unique key. This particular case is discussed later in this section.) For example, each of the following table creation statements is invalid:

```
CREATE TABLE t1 (
    col1 INT NOT NULL,
    col2 DATE NOT NULL,
    col3 INT NOT NULL,
    col4 INT NOT NULL,
    UNIQUE KEY (col1, col2)
)
PARTITION BY HASH(col3)
PARTITIONS 4;

CREATE TABLE t2 (
    col1 INT NOT NULL,
    col2 DATE NOT NULL,
    col3 INT NOT NULL,
    col4 INT NOT NULL,
    UNIQUE KEY (col1),
    UNIQUE KEY (col3)
)
PARTITION BY HASH(col1 + col3)
PARTITIONS 4;
```

In each case, the proposed table would have at least one unique key that does not include all columns used in the partitioning expression.

Each of the following statements is valid, and represents one way in which the corresponding invalid table creation statement could be made to work:

```
CREATE TABLE t1 (
    col1 INT NOT NULL,
    col2 DATE NOT NULL,
    col3 INT NOT NULL,
    col4 INT NOT NULL,
    UNIQUE KEY (col1, col2, col3)
)
PARTITION BY HASH(col3)
PARTITIONS 4;

CREATE TABLE t2 (
    col1 INT NOT NULL,
    col2 DATE NOT NULL,
    col3 INT NOT NULL,
    col4 INT NOT NULL,
    UNIQUE KEY (col1, col3)
)
PARTITION BY HASH(col1 + col3)
PARTITIONS 4;
```

This example shows the error produced in such cases:

```
mysql> CREATE TABLE t3 (
    ->     col1 INT NOT NULL,
    ->     col2 DATE NOT NULL,
    ->     col3 INT NOT NULL,
    ->     col4 INT NOT NULL,
    ->     UNIQUE KEY (col1, col2),
    ->     UNIQUE KEY (col3)
    -> )
    -> PARTITION BY HASH(col1 + col3)
    -> PARTITIONS 4;
ERROR 1491 (HY000): A PRIMARY KEY must include all columns in the table's partitioning function
```

The `CREATE TABLE` statement fails because both `col1` and `col3` are included in the proposed partitioning key, but neither of these columns is part of both of unique keys on the table. This shows one possible fix for the invalid table definition:

```
mysql> CREATE TABLE t3 (
    ->     col1 INT NOT NULL,
    ->     col2 DATE NOT NULL,
    ->     col3 INT NOT NULL,
    ->     col4 INT NOT NULL,
    ->     UNIQUE KEY (col1, col2, col3),
    ->     UNIQUE KEY (col3)
    -> )
    -> PARTITION BY HASH(col3)
    -> PARTITIONS 4;
Query OK, 0 rows affected (0.05 sec)
```

In this case, the proposed partitioning key `col3` is part of both unique keys, and the table creation statement succeeds.

The following table cannot be partitioned at all, because there is no way to include in a partitioning key any columns that belong to both unique keys:

```
CREATE TABLE t4 (
    col1 INT NOT NULL,
    col2 INT NOT NULL,
    col3 INT NOT NULL,
    col4 INT NOT NULL,
    UNIQUE KEY (col1, col3),
    UNIQUE KEY (col2, col4)
);
```

Since every primary key is by definition a unique key, this restriction also includes the table's primary key, if it has one. For example, the next two statements are invalid:

```
CREATE TABLE t5 (
    col1 INT NOT NULL,
    col2 DATE NOT NULL,
    col3 INT NOT NULL,
    col4 INT NOT NULL,
    PRIMARY KEY(col1, col2)
)
PARTITION BY HASH(col3)
PARTITIONS 4;

CREATE TABLE t6 (
    col1 INT NOT NULL,
    col2 DATE NOT NULL,
    col3 INT NOT NULL,
    col4 INT NOT NULL,
    PRIMARY KEY(col1, col3),
    UNIQUE KEY(col2)
)
PARTITION BY HASH( YEAR(col2) )
PARTITIONS 4;
```

In both cases, the primary key does not include all columns referenced in the partitioning expression. However, both of the next two statements are valid:

```
CREATE TABLE t7 (
    col1 INT NOT NULL,
    col2 DATE NOT NULL,
    col3 INT NOT NULL,
    col4 INT NOT NULL,
    PRIMARY KEY(col1, col2)
)
PARTITION BY HASH(col1 + YEAR(col2))
PARTITIONS 4;

CREATE TABLE t8 (
    col1 INT NOT NULL,
    col2 DATE NOT NULL,
    col3 INT NOT NULL,
    col4 INT NOT NULL,
    PRIMARY KEY(col1, col2, col4),
    UNIQUE KEY(col2, col1)
)
PARTITION BY HASH(col1 + YEAR(col2))
PARTITIONS 4;
```

If a table has no unique keys—this includes having no primary key—then this restriction does not apply, and you may use any column or columns in the partitioning expression as long as the column type is compatible with the partitioning type.

For the same reason, you cannot later add a unique key to a partitioned table unless the key includes all columns used by the table's partitioning expression. Consider the partitioned table created as shown here:

```
mysql> CREATE TABLE t_no_pk (c1 INT, c2 INT)
    ->     PARTITION BY RANGE(c1) (
    ->         PARTITION p0 VALUES LESS THAN (10),
    ->         PARTITION p1 VALUES LESS THAN (20),
    ->         PARTITION p2 VALUES LESS THAN (30),
    ->         PARTITION p3 VALUES LESS THAN (40)
    ->     );
Query OK, 0 rows affected (0.12 sec)
```

It is possible to add a primary key to `t_no_pk` using either of these `ALTER TABLE` statements:

```
#  possible PK
mysql> ALTER TABLE t_no_pk ADD PRIMARY KEY(c1);
Query OK, 0 rows affected (0.13 sec)
Records: 0  Duplicates: 0  Warnings: 0

# drop this PK
mysql> ALTER TABLE t_no_pk DROP PRIMARY KEY;
Query OK, 0 rows affected (0.10 sec)
Records: 0  Duplicates: 0  Warnings: 0

#  use another possible PK
mysql> ALTER TABLE t_no_pk ADD PRIMARY KEY(c1, c2);
Query OK, 0 rows affected (0.12 sec)
Records: 0  Duplicates: 0  Warnings: 0

# drop this PK
mysql> ALTER TABLE t_no_pk DROP PRIMARY KEY;
Query OK, 0 rows affected (0.09 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

However, the next statement fails, because `c1` is part of the partitioning key, but is not part of the proposed primary key:

```
#  fails with error 1503
mysql> ALTER TABLE t_no_pk ADD PRIMARY KEY(c2);
ERROR 1503 (HY000): A PRIMARY KEY must include all columns in the table's partitioning function
```

Since `t_no_pk` has only `c1` in its partitioning expression, attempting to adding a unique key on `c2` alone fails. However, you can add a unique key that uses both `c1` and `c2`.

These rules also apply to existing nonpartitioned tables that you wish to partition using `ALTER TABLE ... PARTITION BY`. Consider a table `np_pk` created as shown here:

```
mysql> CREATE TABLE np_pk (
    ->     id INT NOT NULL AUTO_INCREMENT,
    ->     name VARCHAR(50),
    ->     added DATE,
    ->     PRIMARY KEY (id)
    -> );
Query OK, 0 rows affected (0.08 sec)
```

The following `ALTER TABLE` statement fails with an error, because the `added` column is not part of any unique key in the table:

```
mysql> ALTER TABLE np_pk
    ->     PARTITION BY HASH( TO_DAYS(added) )
    ->     PARTITIONS 4;
ERROR 1503 (HY000): A PRIMARY KEY must include all columns in the table's partitioning function
```

However, this statement using the `id` column for the partitioning column is valid, as shown here:

```
mysql> ALTER TABLE np_pk
    ->     PARTITION BY HASH(id)
    ->     PARTITIONS 4;
Query OK, 0 rows affected (0.11 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

In the case of `np_pk`, the only column that may be used as part of a partitioning expression is `id`; if you wish to partition this table using any other column or columns in the partitioning expression, you must first modify the table, either by adding the desired column or columns to the primary key, or by dropping the primary key altogether.

17.6.2 Partitioning Limitations Relating to Storage Engines

The following limitations apply to the use of storage engines with user-defined partitioning of tables.

`MERGE` storage engine. User-defined partitioning and the `MERGE` storage engine are not compatible. Tables using the `MERGE` storage engine cannot be partitioned. Partitioned tables cannot be merged.

`FEDERATED` storage engine. Partitioning of `FEDERATED` tables is not supported; it is not possible to create partitioned `FEDERATED` tables.

`CSV` storage engine. Partitioned tables using the `CSV` storage engine are not supported; it is not possible to create partitioned `CSV` tables.

`InnoDB` storage engine. `InnoDB` foreign keys and MySQL partitioning are not compatible. Partitioned `InnoDB` tables cannot have foreign key references, nor can they have columns referenced by foreign keys. `InnoDB` tables which have or which are referenced by foreign keys cannot be partitioned.

In addition, `ALTER TABLE ... OPTIMIZE PARTITION` does not work correctly with partitioned tables that use the `InnoDB` storage engine. Use `ALTER TABLE ... REBUILD PARTITION` and `ALTER TABLE ... ANALYZE PARTITION`, instead, for such tables. For more information, see Section 13.1.6.1, "`ALTER TABLE` Partition Operations".

Upgrading partitioned tables. When performing an upgrade, tables which are partitioned by `KEY` must be dumped and reloaded.

Same storage engine for all partitions. All partitions of a partitioned table must use the same storage engine and it must be the same storage engine used by the table as a whole. In addition, if one does not specify an engine on the table level, then one must do either of the following when creating or altering a partitioned table:

- Do *not* specify any engine for *any* partition or subpartition
- Specify the engine for *all* partitions or subpartitions

17.6.3 Partitioning Limitations Relating to Functions

This section discusses limitations in MySQL Partitioning relating specifically to functions used in partitioning expressions.

Only the MySQL functions shown in the following table are allowed in partitioning expressions.

`ABS()`	`CEILING()` (see `CEILING()` and `FLOOR()`)	`DAY()`
`DAYOFMONTH()`	`DAYOFWEEK()`	`DAYOFYEAR()`

DATEDIFF()	EXTRACT() (see EXTRACT() function with WEEK specifier)	FLOOR() (see CEILING() and FLOOR())
HOUR()	MICROSECOND()	MINUTE()
MOD()	MONTH()	QUARTER()
SECOND()	TIME_TO_SEC()	TO_DAYS()
TO_SECONDS()	UNIX_TIMESTAMP() (with TIMESTAMP columns)	WEEKDAY()
YEAR()		YEARWEEK()

In MySQL 5.7, partition pruning is supported for the TO_DAYS(), TO_SECONDS(), YEAR(), and UNIX_TIMESTAMP() functions. See Section 17.4, "Partition Pruning", for more information.

CEILING() and FLOOR(). Each of these functions returns an integer only if it is passed an argument of an exact numeric type, such as one of the INT types or DECIMAL. This means, for example, that the following CREATE TABLE statement fails with an error, as shown here:

```
mysql> CREATE TABLE t (c FLOAT) PARTITION BY LIST( FLOOR(c) )(
    ->     PARTITION p0 VALUES IN (1,3,5),
    ->     PARTITION p1 VALUES IN (2,4,6)
    -> );
ERROR 1490 (HY000): The PARTITION function returns the wrong type
```

EXTRACT() function with WEEK specifier. The value returned by the EXTRACT() function, when used as EXTRACT(WEEK FROM *col*), depends on the value of the default_week_format system variable. For this reason, EXTRACT() is not permitted as a partitioning function when it specifies the unit as WEEK. (Bug #54483)

See Section 12.6.2, "Mathematical Functions", for more information about the return types of these functions, as well as Section 11.2, "Numeric Types".

17.6.4 Partitioning and Locking

For storage engines such as MyISAM that actually execute table-level locks when executing DML or DDL statements, such a statement in older versions of MySQL (5.6.5 and earlier)that affected a partitioned table imposed a lock on the table as a whole; that is, all partitions were locked until the statement was finished. In MySQL 5.7, *partition lock pruning* eliminates unneeded locks in many cases, and most statements reading from or updating a partitioned MyISAM table cause only the effected partitions to be locked. For example, a SELECT from a partitioned MyISAM table locks only those partitions actually containing rows that satisfy the SELECT statement's WHERE condition are locked.

For statements effecting partitioned tables using storage engines such as InnoDB, that employ row-level locking and do not actually perform (or need to perform) the locks prior to partition pruning, this is not an issue.

The next few paragraphs discuss the effects of partition lock pruning for various MySQL statements on tables using storage engines that employ table-level locks.

Affects on DML statements

SELECT statements (including those containing unions or joins) lock only those partitions that actually need to be read. This also applies to SELECT ... PARTITION.

An UPDATE prunes locks only for tables on which no partitioning columns are updated.

REPLACE and INSERT lock only those partitions having rows to be inserted or replaced. However, if an AUTO_INCREMENT value is generated for any partitioning column then all partitions are locked.

INSERT ... ON DUPLICATE KEY UPDATE is pruned as long as no partitioning column is updated.

INSERT ... SELECT locks only those partitions in the source table that need to be read, although all partitions in the target table are locked.

Locks imposed by LOAD DATA statements on partitioned tables cannot be pruned.

The presence of BEFORE INSERT or BEFORE UPDATE triggers using any partitioning column of a partitioned table means that locks on INSERT and UPDATE statements updating this table cannot be pruned, since the trigger can alter its values: A BEFORE INSERT trigger on any of the table's partitioning columns means that locks set by INSERT or REPLACE cannot be pruned, since the BEFORE INSERT trigger may change a row's partitioning columns before the row is inserted, forcing the row into a different partition than it would be otherwise. A BEFORE UPDATE trigger on a partitioning column means that locks imposed by UPDATE or INSERT ... ON DUPLICATE KEY UPDATE cannot be pruned.

Affected DDL statements

CREATE VIEW does not cause any locks.

ALTER TABLE ... EXCHANGE PARTITION prunes locks; only the exchanged table and the exchanged partition are locked.

ALTER TABLE ... TRUNCATE PARTITION prunes locks; only the partitions to be emptied are locked.

In addition, ALTER TABLE statements take metadata locks on the table level.

Other statements

LOCK TABLES cannot prune partition locks.

CALL stored_procedure(*expr*) supports lock pruning, but evaluating *expr* does not.

DO and SET statements do not support partitioning lock pruning.

Chapter 18 Stored Programs and Views

Table of Contents

This chapter discusses stored programs and views, which are database objects defined in terms of SQL code that is stored on the server for later execution.

Stored programs include these objects:

- Stored routines, that is, stored procedures and functions. A stored procedure is invoked using the CALL statement. A procedure does not have a return value but can modify its parameters for later inspection by the caller. It can also generate result sets to be returned to the client program. A stored function is used much like a built-in function. you invoke it in an expression and it returns a value during expression evaluation.

- Triggers. A trigger is a named database object that is associated with a table and that is activated when a particular event occurs for the table, such as an insert or update.

- Events. An event is a task that the server runs according to schedule.

Views are stored queries that when referenced produce a result set. A view acts as a virtual table.

This chapter describes how to use stored programs and views. The following sections provide additional information about SQL syntax for statements related to these objects:

- For each object type, there are CREATE, ALTER, and DROP statements that control which objects exist and how they are defined. See Section 13.1, “Data Definition Statements”.

- The CALL statement is used to invoke stored procedures. See Section 13.2.1, “CALL Syntax”.

- Stored program definitions include a body that may use compound statements, loops, conditionals, and declared variables. See Section 13.6, “MySQL Compound-Statement Syntax”.

In MySQL 5.7, metadata changes to objects referred to by stored programs are detected and cause automatic reparsing of the affected statements when the program is next executed. For more information, see Section 8.9.4, “Caching of Prepared Statements and Stored Programs”.

18.1 Defining Stored Programs

Each stored program contains a body that consists of an SQL statement. This statement may be a compound statement made up of several statements separated by semicolon (`;`) characters. For example, the following stored procedure has a body made up of a `BEGIN ... END` block that contains a `SET` statement and a `REPEAT` loop that itself contains another `SET` statement:

```
CREATE PROCEDURE dorepeat(p1 INT)
BEGIN
  SET @x = 0;
  REPEAT SET @x = @x + 1; UNTIL @x > p1 END REPEAT;
END;
```

If you use the `mysql` client program to define a stored program containing semicolon characters, a problem arises. By default, `mysql` itself recognizes the semicolon as a statement delimiter, so you must redefine the delimiter temporarily to cause `mysql` to pass the entire stored program definition to the server.

To redefine the `mysql` delimiter, use the `delimiter` command. The following example shows how to do this for the `dorepeat()` procedure just shown. The delimiter is changed to `//` to enable the entire definition to be passed to the server as a single statement, and then restored to `;` before invoking the procedure. This enables the `;` delimiter used in the procedure body to be passed through to the server rather than being interpreted by `mysql` itself.

```
mysql> delimiter //

mysql> CREATE PROCEDURE dorepeat(p1 INT)
    -> BEGIN
    ->   SET @x = 0;
    ->   REPEAT SET @x = @x + 1; UNTIL @x > p1 END REPEAT;
    -> END
    -> //
Query OK, 0 rows affected (0.00 sec)

mysql> delimiter ;

mysql> CALL dorepeat(1000);
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @x;
+------+
| @x   |
+------+
| 1001 |
+------+
1 row in set (0.00 sec)
```

You can redefine the delimiter to a string other than `//`, and the delimiter can consist of a single character or multiple characters. You should avoid the use of the backslash (“`\`”) character because that is the escape character for MySQL.

The following is an example of a function that takes a parameter, performs an operation using an SQL function, and returns the result. In this case, it is unnecessary to use `delimiter` because the function definition contains no internal `;` statement delimiters:

```
mysql> CREATE FUNCTION hello (s CHAR(20))
mysql> RETURNS CHAR(50) DETERMINISTIC
    -> RETURN CONCAT('Hello, ',s,'!');
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT hello('world');
+----------------+
| hello('world') |
+----------------+
| Hello, world!  |
+----------------+
1 row in set (0.00 sec)
```

18.2 Using Stored Routines (Procedures and Functions)

Stored routines (procedures and functions) are supported in MySQL 5.7. A stored routine is a set of SQL statements that can be stored in the server. Once this has been done, clients don't need to keep reissuing the individual statements but can refer to the stored routine instead.

Stored routines require the `proc` table in the `mysql` database. This table is created during the MySQL 5.7 installation procedure. If you are upgrading to MySQL 5.7 from an earlier version, be sure to update your grant tables to make sure that the `proc` table exists. See Section 4.4.7, “`mysql_upgrade` — Check and Upgrade MySQL Tables”.

Stored routines can be particularly useful in certain situations:

- When multiple client applications are written in different languages or work on different platforms, but need to perform the same database operations.

- When security is paramount. Banks, for example, use stored procedures and functions for all common operations. This provides a consistent and secure environment, and routines can ensure that each operation is properly logged. In such a setup, applications and users would have no access to the database tables directly, but can only execute specific stored routines.

Stored routines can provide improved performance because less information needs to be sent between the server and the client. The tradeoff is that this does increase the load on the database server because more of the work is done on the server side and less is done on the client (application) side. Consider this if many client machines (such as Web servers) are serviced by only one or a few database servers.

Stored routines also enable you to have libraries of functions in the database server. This is a feature shared by modern application languages that enable such design internally (for example, by using classes). Using these client application language features is beneficial for the programmer even outside the scope of database use.

MySQL follows the SQL:2003 syntax for stored routines, which is also used by IBM's DB2. All syntax described here is supported and any limitations and extensions are documented where appropriate.

Additional Resources

- You may find the Stored Procedures User Forum of use when working with stored procedures and functions.

- For answers to some commonly asked questions regarding stored routines in MySQL, see Section B.4, “MySQL 5.7 FAQ: Stored Procedures and Functions”.

- There are some restrictions on the use of stored routines. See Section E.1, “Restrictions on Stored Programs”.

- Binary logging for stored routines takes place as described in Section 18.7, “Binary Logging of Stored Programs”.

18.2.1 Stored Routine Syntax

A stored routine is either a procedure or a function. Stored routines are created with the CREATE PROCEDURE and CREATE FUNCTION statements (see Section 13.1.12, “CREATE PROCEDURE and CREATE FUNCTION Syntax”). A procedure is invoked using a CALL statement (see Section 13.2.1, “CALL Syntax”), and can only pass back values using output variables. A function can be called from inside a statement just like any other function (that is, by invoking the function's name), and can return a scalar value. The body of a stored routine can use compound statements (see Section 13.6, “MySQL Compound-Statement Syntax”).

Stored routines can be dropped with the DROP PROCEDURE and DROP FUNCTION statements (see Section 13.1.21, “DROP PROCEDURE and DROP FUNCTION Syntax”), and altered with the ALTER PROCEDURE and ALTER FUNCTION statements (see Section 13.1.4, “ALTER PROCEDURE Syntax”).

A stored procedure or function is associated with a particular database. This has several implications:

- When the routine is invoked, an implicit USE *db_name* is performed (and undone when the routine terminates). USE statements within stored routines are not permitted.

- You can qualify routine names with the database name. This can be used to refer to a routine that is not in the current database. For example, to invoke a stored procedure p or function f that is associated with the test database, you can say CALL test.p() or test.f().

- When a database is dropped, all stored routines associated with it are dropped as well.

Stored functions cannot be recursive.

Recursion in stored procedures is permitted but disabled by default. To enable recursion, set the max_sp_recursion_depth server system variable to a value greater than zero. Stored procedure recursion increases the demand on thread stack space. If you increase the value of max_sp_recursion_depth, it may be necessary to increase thread stack size by increasing the value of thread_stack at server startup. See Section 5.1.4, “Server System Variables”, for more information.

MySQL supports a very useful extension that enables the use of regular SELECT statements (that is, without using cursors or local variables) inside a stored procedure. The result set of such a query is simply sent directly to the client. Multiple SELECT statements generate multiple result sets, so the client must use a MySQL client library that supports multiple result sets. This means the client must use a client library from a version of MySQL at least as recent as 4.1. The client should also specify the CLIENT_MULTI_RESULTS option when it connects. For C programs, this can be done with the mysql_real_connect() C API function. See Section 21.8.7.54, “mysql_real_connect()”, and Section 21.8.17, “C API Support for Multiple Statement Execution”.

18.2.2 Stored Routines and MySQL Privileges

The MySQL grant system takes stored routines into account as follows:

- The CREATE ROUTINE privilege is needed to create stored routines.

- The ALTER ROUTINE privilege is needed to alter or drop stored routines. This privilege is granted automatically to the creator of a routine if necessary, and dropped from the creator when the routine is dropped.

- The EXECUTE privilege is required to execute stored routines. However, this privilege is granted automatically to the creator of a routine if necessary (and dropped from the creator when the routine is dropped). Also, the default SQL SECURITY characteristic for a routine is DEFINER, which enables users who have access to the database with which the routine is associated to execute the routine.

- If the automatic_sp_privileges system variable is 0, the EXECUTE and ALTER ROUTINE privileges are not automatically granted to and dropped from the routine creator.
- The creator of a routine is the account used to execute the CREATE statement for it. This might not be the same as the account named as the DEFINER in the routine definition.

The server manipulates the mysql.proc table in response to statements that create, alter, or drop stored routines. It is not supported that the server will notice manual manipulation of this table.

18.2.3 Stored Routine Metadata

Metadata about stored routines can be obtained as follows:

- Query the ROUTINES table of the INFORMATION_SCHEMA database. See Section 19.19, “The INFORMATION_SCHEMA ROUTINES Table”.
- Use the SHOW CREATE PROCEDURE and SHOW CREATE FUNCTION statements to see routine definitions. See Section 13.7.5.9, “SHOW CREATE PROCEDURE Syntax”.
- Use the SHOW PROCEDURE STATUS and SHOW FUNCTION STATUS statements to see routine characteristics. See Section 13.7.5.27, “SHOW PROCEDURE STATUS Syntax”.

18.2.4 Stored Procedures, Functions, Triggers, and LAST_INSERT_ID()

Within the body of a stored routine (procedure or function) or a trigger, the value of LAST_INSERT_ID() changes the same way as for statements executed outside the body of these kinds of objects (see Section 12.14, “Information Functions”). The effect of a stored routine or trigger upon the value of LAST_INSERT_ID() that is seen by following statements depends on the kind of routine:

- If a stored procedure executes statements that change the value of LAST_INSERT_ID(), the changed value is seen by statements that follow the procedure call.
- For stored functions and triggers that change the value, the value is restored when the function or trigger ends, so following statements do not see a changed value.

18.3 Using Triggers

A trigger is a named database object that is associated with a table, and that activates when a particular event occurs for the table. Some uses for triggers are to perform checks of values to be inserted into a table or to perform calculations on values involved in an update.

A trigger is defined to activate when a statement inserts, updates, or deletes rows in the associated table. These row operations are trigger events. For example, rows can be inserted by INSERT or LOAD DATA statements, and an insert trigger activates for each inserted row. A trigger can be set to activate either before or after the trigger event. For example, you can have a trigger activate before each row that is inserted into a table or after each row that is updated.

> ⚠ **Important**
>
> MySQL triggers activate only for changes made to tables by SQL statements. They do not activate for changes in views, nor by changes to tables made by APIs that do not transmit SQL statements to the MySQL Server. This means that triggers are not activated by changes in INFORMATION_SCHEMA or performance_schema tables, because these tables are actually views.

The following sections describe the syntax for creating and dropping triggers, show some examples of how to use them, and indicate how to obtain trigger metadata.

Additional Resources

- You may find the Triggers User Forum of use when working with triggers.
- For answers to commonly asked questions regarding triggers in MySQL, see Section B.5, "MySQL 5.7 FAQ: Triggers".
- There are some restrictions on the use of triggers; see Section E.1, "Restrictions on Stored Programs".
- Binary logging for triggers takes place as described in Section 18.7, "Binary Logging of Stored Programs".

18.3.1 Trigger Syntax and Examples

To create a trigger or drop a trigger, use the `CREATE TRIGGER` or `DROP TRIGGER` statement, described in Section 13.1.15, "`CREATE TRIGGER` Syntax", and Section 13.1.24, "`DROP TRIGGER` Syntax".

Here is a simple example that associates a trigger with a table, to activate for `INSERT` operations. The trigger acts as an accumulator, summing the values inserted into one of the columns of the table.

```
mysql> CREATE TABLE account (acct_num INT, amount DECIMAL(10,2));
Query OK, 0 rows affected (0.03 sec)

mysql> CREATE TRIGGER ins_sum BEFORE INSERT ON account
    -> FOR EACH ROW SET @sum = @sum + NEW.amount;
Query OK, 0 rows affected (0.06 sec)
```

The `CREATE TRIGGER` statement creates a trigger named `ins_sum` that is associated with the `account` table. It also includes clauses that specify the trigger action time, the triggering event, and what to do when the trigger activates:

- The keyword `BEFORE` indicates the trigger action time. In this case, the trigger activates before each row inserted into the table. The other permitted keyword here is `AFTER`.
- The keyword `INSERT` indicates the trigger event; that is, the type of operation that activates the trigger. In the example, `INSERT` operations cause trigger activation. You can also create triggers for `DELETE` and `UPDATE` operations.
- The statement following `FOR EACH ROW` defines the trigger body; that is, the statement to execute each time the trigger activates, which occurs once for each row affected by the triggering event. In the example, the trigger body is a simple `SET` that accumulates into a user variable the values inserted into the `amount` column. The statement refers to the column as `NEW.amount` which means "the value of the `amount` column to be inserted into the new row."

To use the trigger, set the accumulator variable to zero, execute an `INSERT` statement, and then see what value the variable has afterward:

```
mysql> SET @sum = 0;
mysql> INSERT INTO account VALUES(137,14.98),(141,1937.50),(97,-100.00);
mysql> SELECT @sum AS 'Total amount inserted';
+-----------------------+
| Total amount inserted |
+-----------------------+
| 1852.48               |
+-----------------------+
```

In this case, the value of @sum after the INSERT statement has executed is 14.98 + 1937.50 - 100, or 1852.48.

To destroy the trigger, use a DROP TRIGGER statement. You must specify the schema name if the trigger is not in the default schema:

```
mysql> DROP TRIGGER test.ins_sum;
```

If you drop a table, any triggers for the table are also dropped.

Trigger names exist in the schema namespace, meaning that all triggers must have unique names within a schema. Triggers in different schemas can have the same name.

As of MySQL 5.7.2, it is possible to define multiple triggers for a given table that have the same trigger event and action time. For example, you can have two BEFORE UPDATE triggers for a table. By default, triggers that have the same trigger event and action time activate in the order they were created. To affect trigger order, specify a clause after FOR EACH ROW that indicates FOLLOWS or PRECEDES and the name of an existing trigger that also has the same trigger event and action time. With FOLLOWS, the new trigger activates after the existing trigger. With PRECEDES, the new trigger activates before the existing trigger.

For example, the following trigger definition defines another BEFORE INSERT trigger for the account table:

```
mysql> CREATE TRIGGER ins_transaction BEFORE INSERT ON account
    -> FOR EACH ROW PRECEDES ins_sum
    -> SET
    -> @deposits = @deposits + IF(NEW.amount>0,NEW.amount,0),
    -> @withdrawals = @withdrawals + IF(NEW.amount<0,-NEW.amount,0);
Query OK, 0 rows affected (0.02 sec)
```

This trigger, ins_transaction, is similar to ins_sum but accumulates deposits and withdrawals separately. It has a PRECEDES clause that causes it to activate before ins_sum; without that clause, it would activate after ins_sum because it is created after ins_sum.

Before MySQL 5.7.2, there cannot be multiple triggers for a given table that have the same trigger event and action time. For example, you cannot have two BEFORE UPDATE triggers for a table. To work around this, you can define a trigger that executes multiple statements by using the BEGIN ... END compound statement construct after FOR EACH ROW. (An example appears later in this section.)

Within the trigger body, the OLD and NEW keywords enable you to access columns in the rows affected by a trigger. OLD and NEW are MySQL extensions to triggers; they are not case sensitive.

In an INSERT trigger, only NEW.*col_name* can be used; there is no old row. In a DELETE trigger, only OLD.*col_name* can be used; there is no new row. In an UPDATE trigger, you can use OLD.*col_name* to refer to the columns of a row before it is updated and NEW.*col_name* to refer to the columns of the row after it is updated.

A column named with OLD is read only. You can refer to it (if you have the SELECT privilege), but not modify it. You can refer to a column named with NEW if you have the SELECT privilege for it. In a BEFORE trigger, you can also change its value with SET NEW.*col_name* = *value* if you have the UPDATE privilege for it. This means you can use a trigger to modify the values to be inserted into a new row or used to update a row. (Such a SET statement has no effect in an AFTER trigger because the row change will have already occurred.)

In a BEFORE trigger, the NEW value for an AUTO_INCREMENT column is 0, not the sequence number that is generated automatically when the new row actually is inserted.

By using the BEGIN ... END construct, you can define a trigger that executes multiple statements. Within the BEGIN block, you also can use other syntax that is permitted within stored routines such as conditionals and loops. However, just as for stored routines, if you use the mysql program to define a trigger that executes multiple statements, it is necessary to redefine the mysql statement delimiter so that you can use the ; statement delimiter within the trigger definition. The following example illustrates these points. It defines an UPDATE trigger that checks the new value to be used for updating each row, and modifies the value to be within the range from 0 to 100. This must be a BEFORE trigger because the value must be checked before it is used to update the row:

```
mysql> delimiter //
mysql> CREATE TRIGGER upd_check BEFORE UPDATE ON account
    -> FOR EACH ROW
    -> BEGIN
    ->     IF NEW.amount < 0 THEN
    ->         SET NEW.amount = 0;
    ->     ELSEIF NEW.amount > 100 THEN
    ->         SET NEW.amount = 100;
    ->     END IF;
    -> END;//
mysql> delimiter ;
```

It can be easier to define a stored procedure separately and then invoke it from the trigger using a simple CALL statement. This is also advantageous if you want to execute the same code from within several triggers.

There are limitations on what can appear in statements that a trigger executes when activated:

- The trigger cannot use the CALL statement to invoke stored procedures that return data to the client or that use dynamic SQL. (Stored procedures are permitted to return data to the trigger through OUT or INOUT parameters.)

- The trigger cannot use statements that explicitly or implicitly begin or end a transaction, such as START TRANSACTION, COMMIT, or ROLLBACK.

See also Section E.1, “Restrictions on Stored Programs”.

MySQL handles errors during trigger execution as follows:

- If a BEFORE trigger fails, the operation on the corresponding row is not performed.

- A BEFORE trigger is activated by the *attempt* to insert or modify the row, regardless of whether the attempt subsequently succeeds.

- An AFTER trigger is executed only if any BEFORE triggers and the row operation execute successfully.

- An error during either a BEFORE or AFTER trigger results in failure of the entire statement that caused trigger invocation.

- For transactional tables, failure of a statement should cause rollback of all changes performed by the statement. Failure of a trigger causes the statement to fail, so trigger failure also causes rollback. For nontransactional tables, such rollback cannot be done, so although the statement fails, any changes performed prior to the point of the error remain in effect.

Triggers can contain direct references to tables by name, such as the trigger named testref shown in this example:

```
CREATE TABLE test1(a1 INT);
CREATE TABLE test2(a2 INT);
CREATE TABLE test3(a3 INT NOT NULL AUTO_INCREMENT PRIMARY KEY);
CREATE TABLE test4(
  a4 INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
  b4 INT DEFAULT 0
);

delimiter |

CREATE TRIGGER testref BEFORE INSERT ON test1
  FOR EACH ROW
  BEGIN
    INSERT INTO test2 SET a2 = NEW.a1;
    DELETE FROM test3 WHERE a3 = NEW.a1;
    UPDATE test4 SET b4 = b4 + 1 WHERE a4 = NEW.a1;
  END;
|

delimiter ;

INSERT INTO test3 (a3) VALUES
  (NULL), (NULL), (NULL), (NULL), (NULL),
  (NULL), (NULL), (NULL), (NULL), (NULL);

INSERT INTO test4 (a4) VALUES
  (0), (0), (0), (0), (0), (0), (0), (0), (0), (0);
```

Suppose that you insert the following values into table `test1` as shown here:

```
mysql> INSERT INTO test1 VALUES
    -> (1), (3), (1), (7), (1), (8), (4), (4);
Query OK, 8 rows affected (0.01 sec)
Records: 8  Duplicates: 0  Warnings: 0
```

As a result, the four tables contain the following data:

```
mysql> SELECT * FROM test1;
+------+
| a1   |
+------+
|    1 |
|    3 |
|    1 |
|    7 |
|    1 |
|    8 |
|    4 |
|    4 |
+------+
8 rows in set (0.00 sec)

mysql> SELECT * FROM test2;
+------+
| a2   |
+------+
|    1 |
|    3 |
|    1 |
|    7 |
|    1 |
|    8 |
|    4 |
|    4 |
+------+
8 rows in set (0.00 sec)

mysql> SELECT * FROM test3;
```

```
+----+
| a3 |
+----+
|  2 |
|  5 |
|  6 |
|  9 |
| 10 |
+----+
5 rows in set (0.00 sec)

mysql> SELECT * FROM test4;
+----+------+
| a4 | b4   |
+----+------+
|  1 |    3 |
|  2 |    0 |
|  3 |    1 |
|  4 |    2 |
|  5 |    0 |
|  6 |    0 |
|  7 |    1 |
|  8 |    1 |
|  9 |    0 |
| 10 |    0 |
+----+------+
10 rows in set (0.00 sec)
```

18.3.2 Trigger Metadata

Metadata about triggers can be obtained as follows:

- Query the TRIGGERS table of the INFORMATION_SCHEMA database. See Section 19.27, "The INFORMATION_SCHEMA TRIGGERS Table".

- Use the SHOW CREATE TRIGGER statement. See Section 13.7.5.11, "SHOW CREATE TRIGGER Syntax".

- Use the SHOW TRIGGERS statement. See Section 13.7.5.37, "SHOW TRIGGERS Syntax".

18.4 Using the Event Scheduler

The *MySQL Event Scheduler* manages the scheduling and execution of events, that is, tasks that run according to a schedule. The following discussion covers the Event Scheduler and is divided into the following sections:

- Section 18.4.1, "Event Scheduler Overview", provides an introduction to and conceptual overview of MySQL Events.

- Section 18.4.3, "Event Syntax", discusses the SQL statements for creating, altering, and dropping MySQL Events.

- Section 18.4.4, "Event Metadata", shows how to obtain information about events and how this information is stored by the MySQL Server.

- Section 18.4.6, "The Event Scheduler and MySQL Privileges", discusses the privileges required to work with events and the ramifications that events have with regard to privileges when executing.

Stored routines require the event table in the mysql database. This table is created during the MySQL 5.7 installation procedure. If you are upgrading to MySQL 5.7 from an earlier version, be sure to update your grant tables to make sure that the event table exists. See Section 4.4.7, "mysql_upgrade — Check and Upgrade MySQL Tables".

Additional Resources

- You may find the MySQL Event Scheduler User Forum of use when working with scheduled events.
- There are some restrictions on the use of events; see Section E.1, "Restrictions on Stored Programs".
- Binary logging for events takes place as described in Section 18.7, "Binary Logging of Stored Programs".

18.4.1 Event Scheduler Overview

MySQL Events are tasks that run according to a schedule. Therefore, we sometimes refer to them as *scheduled* events. When you create an event, you are creating a named database object containing one or more SQL statements to be executed at one or more regular intervals, beginning and ending at a specific date and time. Conceptually, this is similar to the idea of the Unix `crontab` (also known as a "cron job") or the Windows Task Scheduler.

Scheduled tasks of this type are also sometimes known as "temporal triggers", implying that these are objects that are triggered by the passage of time. While this is essentially correct, we prefer to use the term *events* to avoid confusion with triggers of the type discussed in Section 18.3, "Using Triggers". Events should more specifically not be confused with "temporary triggers". Whereas a trigger is a database object whose statements are executed in response to a specific type of event that occurs on a given table, a (scheduled) event is an object whose statements are executed in response to the passage of a specified time interval.

While there is no provision in the SQL Standard for event scheduling, there are precedents in other database systems, and you may notice some similarities between these implementations and that found in the MySQL Server.

MySQL Events have the following major features and properties:

- In MySQL 5.7, an event is uniquely identified by its name and the schema to which it is assigned.
- An event performs a specific action according to a schedule. This action consists of an SQL statement, which can be a compound statement in a `BEGIN ... END` block if desired (see Section 13.6, "MySQL Compound-Statement Syntax"). An event's timing can be either *one-time* or *recurrent*. A one-time event executes one time only. A recurrent event repeats its action at a regular interval, and the schedule for a recurring event can be assigned a specific start day and time, end day and time, both, or neither. (By default, a recurring event's schedule begins as soon as it is created, and continues indefinitely, until it is disabled or dropped.)

  If a repeating event does not terminate within its scheduling interval, the result may be multiple instances of the event executing simultaneously. If this is undesirable, you should institute a mechanism to prevent simultaneous instances. For example, you could use the `GET_LOCK()` function, or row or table locking.
- Users can create, modify, and drop scheduled events using SQL statements intended for these purposes. Syntactically invalid event creation and modification statements fail with an appropriate error message. *A user may include statements in an event's action which require privileges that the user does not actually have*. The event creation or modification statement succeeds but the event's action fails. See Section 18.4.6, "The Event Scheduler and MySQL Privileges" for details.
- Many of the properties of an event can be set or modified using SQL statements. These properties include the event's name, timing, persistence (that is, whether it is preserved following the expiration of its schedule), status (enabled or disabled), action to be performed, and the schema to which it is assigned. See Section 13.1.2, "`ALTER EVENT` Syntax".

  The default definer of an event is the user who created the event, unless the event has been altered, in which case the definer is the user who issued the last `ALTER EVENT` statement affecting that event. An event can be modified by any user having the `EVENT` privilege on the database for which the event is defined. See Section 18.4.6, “The Event Scheduler and MySQL Privileges”.

- An event's action statement may include most SQL statements permitted within stored routines. For restrictions, see Section E.1, “Restrictions on Stored Programs”.

18.4.2 Event Scheduler Configuration

Events are executed by a special *event scheduler thread*; when we refer to the Event Scheduler, we actually refer to this thread. When running, the event scheduler thread and its current state can be seen by users having the `PROCESS` privilege in the output of `SHOW PROCESSLIST`, as shown in the discussion that follows.

The global `event_scheduler` system variable determines whether the Event Scheduler is enabled and running on the server. It has one of these 3 values, which affect event scheduling as described here:

- `OFF`: The Event Scheduler is stopped. The event scheduler thread does not run, is not shown in the output of `SHOW PROCESSLIST`, and no scheduled events are executed. `OFF` is the default value for `event_scheduler`.

  When the Event Scheduler is stopped (`event_scheduler` is `OFF`), it can be started by setting the value of `event_scheduler` to `ON`. (See next item.)

- `ON`: The Event Scheduler is started; the event scheduler thread runs and executes all scheduled events.

  When the Event Scheduler is `ON`, the event scheduler thread is listed in the output of `SHOW PROCESSLIST` as a daemon process, and its state is represented as shown here:

```
mysql> SHOW PROCESSLIST\G
*************************** 1. row ***************************
     Id: 1
   User: root
   Host: localhost
     db: NULL
Command: Query
   Time: 0
  State: NULL
   Info: show processlist
*************************** 2. row ***************************
     Id: 2
   User: event_scheduler
   Host: localhost
     db: NULL
Command: Daemon
   Time: 3
  State: Waiting for next activation
   Info: NULL
2 rows in set (0.00 sec)
```

  Event scheduling can be stopped by setting the value of `event_scheduler` to `OFF`.

- `DISABLED`: This value renders the Event Scheduler nonoperational. When the Event Scheduler is `DISABLED`, the event scheduler thread does not run (and so does not appear in the output of `SHOW PROCESSLIST`). In addition, the Event Scheduler state cannot be changed at runtime.

If the Event Scheduler status has not been set to `DISABLED`, `event_scheduler` can be toggled between `ON` and `OFF` (using `SET`). It is also possible to use `0` for `OFF`, and `1` for `ON` when setting this variable. Thus, any of the following 4 statements can be used in the `mysql` client to turn on the Event Scheduler:

```
SET GLOBAL event_scheduler = ON;
SET @@global.event_scheduler = ON;
SET GLOBAL event_scheduler = 1;
SET @@global.event_scheduler = 1;
```

Similarly, any of these 4 statements can be used to turn off the Event Scheduler:

```
SET GLOBAL event_scheduler = OFF;
SET @@global.event_scheduler = OFF;
SET GLOBAL event_scheduler = 0;
SET @@global.event_scheduler = 0;
```

Although `ON` and `OFF` have numeric equivalents, the value displayed for `event_scheduler` by `SELECT` or `SHOW VARIABLES` is always one of `OFF`, `ON`, or `DISABLED`. *`DISABLED` has no numeric equivalent*. For this reason, `ON` and `OFF` are usually preferred over `1` and `0` when setting this variable.

Note that attempting to set `event_scheduler` without specifying it as a global variable causes an error:

```
mysql< SET @@event_scheduler = OFF;
ERROR 1229 (HY000): Variable 'event_scheduler' is a GLOBAL
variable and should be set with SET GLOBAL
```

Important

It is possible to set the Event Scheduler to `DISABLED` only at server startup. If `event_scheduler` is `ON` or `OFF`, you cannot set it to `DISABLED` at runtime. Also, if the Event Scheduler is set to `DISABLED` at startup, you cannot change the value of `event_scheduler` at runtime.

To disable the event scheduler, use one of the following two methods:

- As a command-line option when starting the server:

  ```
  --event-scheduler=DISABLED
  ```

- In the server configuration file (`my.cnf`, or `my.ini` on Windows systems), include the line where it will be read by the server (for example, in a `[mysqld]` section):

  ```
  event_scheduler=DISABLED
  ```

To enable the Event Scheduler, restart the server without the `--event-scheduler=DISABLED` command-line option, or after removing or commenting out the line containing `event-scheduler=DISABLED` in the server configuration file, as appropriate. Alternatively, you can use `ON` (or `1`) or `OFF` (or `0`) in place of the `DISABLED` value when starting the server.

Note

You can issue event-manipulation statements when `event_scheduler` is set to `DISABLED`. No warnings or errors are generated in such cases (provided that the statements are themselves valid). However, scheduled events cannot execute until this variable is set to `ON` (or `1`). Once this has been done, the event scheduler thread executes all events whose scheduling conditions are satisfied.

Starting the MySQL server with the `--skip-grant-tables` option causes `event_scheduler` to be set to `DISABLED`, overriding any other value set either on the command line or in the `my.cnf` or `my.ini` file (Bug #26807).

For SQL statements used to create, alter, and drop events, see Section 18.4.3, "Event Syntax".

MySQL 5.7 provides an `EVENTS` table in the `INFORMATION_SCHEMA` database. This table can be queried to obtain information about scheduled events which have been defined on the server. See Section 18.4.4, "Event Metadata", and Section 19.7, "The `INFORMATION_SCHEMA EVENTS` Table", for more information.

For information regarding event scheduling and the MySQL privilege system, see Section 18.4.6, "The Event Scheduler and MySQL Privileges".

18.4.3 Event Syntax

MySQL 5.7 provides several SQL statements for working with scheduled events:

- New events are defined using the `CREATE EVENT` statement. See Section 13.1.9, "`CREATE EVENT` Syntax".

- The definition of an existing event can be changed by means of the `ALTER EVENT` statement. See Section 13.1.2, "`ALTER EVENT` Syntax".

- When a scheduled event is no longer wanted or needed, it can be deleted from the server by its definer using the `DROP EVENT` statement. See Section 13.1.18, "`DROP EVENT` Syntax". Whether an event persists past the end of its schedule also depends on its `ON COMPLETION` clause, if it has one. See Section 13.1.9, "`CREATE EVENT` Syntax".

  An event can be dropped by any user having the `EVENT` privilege for the database on which the event is defined. See Section 18.4.6, "The Event Scheduler and MySQL Privileges".

18.4.4 Event Metadata

Metadata about events can be obtained as follows:

- Query the `event` table of the `mysql` database.

- Query the `EVENTS` table of the `INFORMATION_SCHEMA` database. See Section 19.7, "The `INFORMATION_SCHEMA EVENTS` Table".

- Use the `SHOW CREATE EVENT` statement. See Section 13.7.5.7, "`SHOW CREATE EVENT` Syntax".

- Use the `SHOW EVENTS` statement. See Section 13.7.5.17, "`SHOW EVENTS` Syntax".

Event Scheduler Time Representation

Each session in MySQL has a session time zone (STZ). This is the session `time_zone` value that is initialized from the server's global `time_zone` value when the session begins but may be changed during the session.

The session time zone that is current when a `CREATE EVENT` or `ALTER EVENT` statement executes is used to interpret times specified in the event definition. This becomes the event time zone (ETZ); that is, the time zone that is used for event scheduling and is in effect within the event as it executes.

For representation of event information in the `mysql.event` table, the `execute_at`, `starts`, and `ends` times are converted to UTC and stored along with the event time zone. This enables event execution to proceed as defined regardless of any subsequent changes to the server time zone or daylight saving time effects. The `last_executed` time is also stored in UTC.

If you select information from `mysql.event`, the times just mentioned are retrieved as UTC values. These times can also be obtained by selecting from the `INFORMATION_SCHEMA.EVENTS` table or from `SHOW EVENTS`, but they are reported as ETZ values. Other times available from these sources indicate when an event was created or last altered; these are displayed as STZ values. The following table summarizes representation of event times.

Value	mysql.event	INFORMATION_SCHEMA.EVENTS	SHOW EVENTS
Execute at	UTC	ETZ	ETZ
Starts	UTC	ETZ	ETZ
Ends	UTC	ETZ	ETZ
Last executed	UTC	ETZ	n/a
Created	STZ	STZ	n/a
Last altered	STZ	STZ	n/a

18.4.5 Event Scheduler Status

The Event Scheduler writes information about event execution that terminates with an error or warning to the MySQL Server's error log. See Section 18.4.6, “The Event Scheduler and MySQL Privileges” for an example.

To obtain information about the state of the Event Scheduler for debugging and troubleshooting purposes, run `mysqladmin debug` (see Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”); after running this command, the server's error log contains output relating to the Event Scheduler, similar to what is shown here:

```
Events status:
LLA = Last Locked At  LUA = Last Unlocked At
WOC = Waiting On Condition  DL = Data Locked

Event scheduler status:
State      : INITIALIZED
Thread id  : 0
LLA        : init_scheduler:313
LUA        : init_scheduler:318
WOC        : NO
Workers    : 0
Executed   : 0
Data locked: NO

Event queue status:
Element count   : 1
Data locked     : NO
Attempting lock : NO
LLA             : init_queue:148
LUA             : init_queue:168
WOC             : NO
Next activation : 0000-00-00 00:00:00
```

In statements that occur as part of events executed by the Event Scheduler, diagnostics messages (not only errors, but also warnings) are written to the error log, and, on Windows, to the application event log. For frequently executed events, it is possible for this to result in many logged messages. For example, for `SELECT ... INTO` *`var_list`* statements, if the query returns no rows, a warning with error code 1329 occurs (`No data`), and the variable values remain unchanged. If the query returns multiple rows, error 1172 occurs (`Result consisted of more than one row`). For either condition, you can avoid having the warnings be logged by declaring a condition handler; see Section 13.6.7.2, “`DECLARE ... HANDLER` Syntax”. For statements that may retrieve multiple rows, another strategy is to use `LIMIT 1` to limit the result set to a single row.

18.4.6 The Event Scheduler and MySQL Privileges

To enable or disable the execution of scheduled events, it is necessary to set the value of the global `event_scheduler` system variable. This requires the `SUPER` privilege.

The `EVENT` privilege governs the creation, modification, and deletion of events. This privilege can be bestowed using `GRANT`. For example, this `GRANT` statement confers the `EVENT` privilege for the schema named `myschema` on the user `jon@ghidora`:

```
GRANT EVENT ON myschema.* TO jon@ghidora;
```

(We assume that this user account already exists, and that we wish for it to remain unchanged otherwise.)

To grant this same user the `EVENT` privilege on all schemas, use the following statement:

```
GRANT EVENT ON *.* TO jon@ghidora;
```

The `EVENT` privilege has global or schema-level scope. Therefore, trying to grant it on a single table results in an error as shown:

```
mysql> GRANT EVENT ON myschema.mytable TO jon@ghidora;
ERROR 1144 (42000): Illegal GRANT/REVOKE command; please
consult the manual to see which privileges can be used
```

It is important to understand that an event is executed with the privileges of its definer, and that it cannot perform any actions for which its definer does not have the requisite privileges. For example, suppose that `jon@ghidora` has the `EVENT` privilege for `myschema`. Suppose also that this user has the `SELECT` privilege for `myschema`, but no other privileges for this schema. It is possible for `jon@ghidora` to create a new event such as this one:

```
CREATE EVENT e_store_ts
    ON SCHEDULE
      EVERY 10 SECOND
    DO
      INSERT INTO myschema.mytable VALUES (UNIX_TIMESTAMP());
```

The user waits for a minute or so, and then performs a `SELECT * FROM mytable;` query, expecting to see several new rows in the table. Instead, the table is empty. Since the user does not have the `INSERT` privilege for the table in question, the event has no effect.

If you inspect the MySQL error log (*`hostname`*`.err`), you can see that the event is executing, but the action it is attempting to perform fails:

```
2013-09-24T12:41:31.261992Z 25 [ERROR] Event Scheduler:
[jon@ghidora][cookbook.e_store_ts] INSERT command denied to user
'jon'@'ghidora' for table 'mytable'
2013-09-24T12:41:31.262022Z 25 [Note] Event Scheduler:
[jon@ghidora].[myschema.e_store_ts] event execution failed.
2013-09-24T12:41:41.271796Z 26 [ERROR] Event Scheduler:
[jon@ghidora][cookbook.e_store_ts] INSERT command denied to user
'jon'@'ghidora' for table 'mytable'
2013-09-24T12:41:41.272761Z 26 [Note] Event Scheduler:
[jon@ghidora].[myschema.e_store_ts] event execution failed.
```

Since this user very likely does not have access to the error log, it is possible to verify whether the event's action statement is valid by executing it directly:

```
mysql> INSERT INTO myschema.mytable VALUES (UNIX_TIMESTAMP());
```

```
ERROR 1142 (42000): INSERT command denied to user
'jon'@'ghidora' for table 'mytable'
```

Inspection of the `INFORMATION_SCHEMA.EVENTS` table shows that `e_store_ts` exists and is enabled, but its `LAST_EXECUTED` column is `NULL`:

```
mysql> SELECT * FROM INFORMATION_SCHEMA.EVENTS
     >     WHERE EVENT_NAME='e_store_ts'
     >     AND EVENT_SCHEMA='myschema'\G
*************************** 1. row ***************************
   EVENT_CATALOG: NULL
    EVENT_SCHEMA: myschema
      EVENT_NAME: e_store_ts
         DEFINER: jon@ghidora
      EVENT_BODY: SQL
EVENT_DEFINITION: INSERT INTO myschema.mytable VALUES (UNIX_TIMESTAMP())
      EVENT_TYPE: RECURRING
      EXECUTE_AT: NULL
  INTERVAL_VALUE: 5
  INTERVAL_FIELD: SECOND
        SQL_MODE: NULL
          STARTS: 0000-00-00 00:00:00
            ENDS: 0000-00-00 00:00:00
          STATUS: ENABLED
   ON_COMPLETION: NOT PRESERVE
         CREATED: 2006-02-09 22:36:06
    LAST_ALTERED: 2006-02-09 22:36:06
   LAST_EXECUTED: NULL
   EVENT_COMMENT:
1 row in set (0.00 sec)
```

To rescind the `EVENT` privilege, use the `REVOKE` statement. In this example, the `EVENT` privilege on the schema `myschema` is removed from the `jon@ghidora` user account:

```
REVOKE EVENT ON myschema.* FROM jon@ghidora;
```

Important

Revoking the `EVENT` privilege from a user does not delete or disable any events that may have been created by that user.

An event is not migrated or dropped as a result of renaming or dropping the user who created it.

Suppose that the user `jon@ghidora` has been granted the `EVENT` and `INSERT` privileges on the `myschema` schema. This user then creates the following event:

```
CREATE EVENT e_insert
    ON SCHEDULE
      EVERY 7 SECOND
    DO
      INSERT INTO myschema.mytable;
```

After this event has been created, `root` revokes the `EVENT` privilege for `jon@ghidora`. However, `e_insert` continues to execute, inserting a new row into `mytable` each seven seconds. The same would be true if `root` had issued either of these statements:

- `DROP USER jon@ghidora;`
- `RENAME USER jon@ghidora TO someotherguy@ghidora;`

You can verify that this is true by examining the mysql.event table (discussed later in this section) or the INFORMATION_SCHEMA.EVENTS table (see Section 19.7, “The INFORMATION_SCHEMA EVENTS Table”) before and after issuing a DROP USER or RENAME USER statement.

Event definitions are stored in the mysql.event table. To drop an event created by another user account, the MySQL root user (or another user with the necessary privileges) can delete rows from this table. For example, to remove the event e_insert shown previously, root can use the following statement:

```
DELETE FROM mysql.event
    WHERE db = 'myschema'
      AND definer = 'jon@ghidora'
      AND name = 'e_insert';
```

It is very important to match the event name, database schema name, and user account when deleting rows from the mysql.event table. This is because the same user can create different events of the same name in different schemas.

Users' EVENT privileges are stored in the Event_priv columns of the mysql.user and mysql.db tables. In both cases, this column holds one of the values 'Y' or 'N'. 'N' is the default. mysql.user.Event_priv is set to 'Y' for a given user only if that user has the global EVENT privilege (that is, if the privilege was bestowed using GRANT EVENT ON *.*). For a schema-level EVENT privilege, GRANT creates a row in mysql.db and sets that row's Db column to the name of the schema, the User column to the name of the user, and the Event_priv column to 'Y'. There should never be any need to manipulate these tables directly, since the GRANT EVENT and REVOKE EVENT statements perform the required operations on them.

Five status variables provide counts of event-related operations (but *not* of statements executed by events; see Section E.1, “Restrictions on Stored Programs”). These are:

- Com_create_event: The number of CREATE EVENT statements executed since the last server restart.
- Com_alter_event: The number of ALTER EVENT statements executed since the last server restart.
- Com_drop_event: The number of DROP EVENT statements executed since the last server restart.
- Com_show_create_event: The number of SHOW CREATE EVENT statements executed since the last server restart.
- Com_show_events: The number of SHOW EVENTS statements executed since the last server restart.

You can view current values for all of these at one time by running the statement SHOW STATUS LIKE '%event%';.

18.5 Using Views

Views (including updatable views) are available in MySQL Server 5.7. Views are stored queries that when invoked produce a result set. A view acts as a virtual table.

To use views if you have upgraded to MySQL 5.7 from an older release that did not support views, you should upgrade your grant tables so that they contain the view-related privileges. See Section 4.4.7, “mysql_upgrade — Check and Upgrade MySQL Tables”.

The following discussion describes the syntax for creating and dropping views, and shows some examples of how to use them.

Additional Resources

- You may find the Views User Forum of use when working with views.

- For answers to some commonly asked questions regarding views in MySQL, see Section B.6, “MySQL 5.7 FAQ: Views”.
- There are some restrictions on the use of views; see Section E.5, “Restrictions on Views”.

18.5.1 View Syntax

The CREATE VIEW statement creates a new view (see Section 13.1.16, “CREATE VIEW Syntax”). To alter the definition of a view or drop a view, use ALTER VIEW (see Section 13.1.7, “ALTER VIEW Syntax”), or DROP VIEW (see Section 13.1.25, “DROP VIEW Syntax”).

A view can be created from many kinds of SELECT statements. It can refer to base tables or other views. It can use joins, UNION, and subqueries. The SELECT need not even refer to any tables. The following example defines a view that selects two columns from another table, as well as an expression calculated from those columns:

```
mysql> CREATE TABLE t (qty INT, price INT);
mysql> INSERT INTO t VALUES(3, 50), (5, 60);
mysql> CREATE VIEW v AS SELECT qty, price, qty*price AS value FROM t;
mysql> SELECT * FROM v;
+------+-------+-------+
| qty  | price | value |
+------+-------+-------+
|    3 |    50 |   150 |
|    5 |    60 |   300 |
+------+-------+-------+
mysql> SELECT * FROM v WHERE qty = 5;
+------+-------+-------+
| qty  | price | value |
+------+-------+-------+
|    5 |    60 |   300 |
+------+-------+-------+
```

18.5.2 View Processing Algorithms

The optional ALGORITHM clause for CREATE VIEW or ALTER VIEW is a MySQL extension to standard SQL. It affects how MySQL processes the view. ALGORITHM takes three values: MERGE, TEMPTABLE, or UNDEFINED. The default algorithm is UNDEFINED if no ALGORITHM clause is present.

For MERGE, the text of a statement that refers to the view and the view definition are merged such that parts of the view definition replace corresponding parts of the statement.

For TEMPTABLE, the results from the view are retrieved into a temporary table, which then is used to execute the statement.

For UNDEFINED, MySQL chooses which algorithm to use. It prefers MERGE over TEMPTABLE if possible, because MERGE is usually more efficient and because a view cannot be updatable if a temporary table is used.

A reason to choose TEMPTABLE explicitly is that locks can be released on underlying tables after the temporary table has been created and before it is used to finish processing the statement. This might result in quicker lock release than the MERGE algorithm so that other clients that use the view are not blocked as long.

A view algorithm can be UNDEFINED for three reasons:

- No ALGORITHM clause is present in the CREATE VIEW statement.
- The CREATE VIEW statement has an explicit ALGORITHM = UNDEFINED clause.

- ALGORITHM = MERGE is specified for a view that can be processed only with a temporary table. In this case, MySQL generates a warning and sets the algorithm to UNDEFINED.

As mentioned earlier, MERGE is handled by merging corresponding parts of a view definition into the statement that refers to the view. The following examples briefly illustrate how the MERGE algorithm works. The examples assume that there is a view v_merge that has this definition:

```
CREATE ALGORITHM = MERGE VIEW v_merge (vc1, vc2) AS
SELECT c1, c2 FROM t WHERE c3 > 100;
```

Example 1: Suppose that we issue this statement:

```
SELECT * FROM v_merge;
```

MySQL handles the statement as follows:

- v_merge becomes t
- * becomes vc1, vc2, which corresponds to c1, c2
- The view WHERE clause is added

The resulting statement to be executed becomes:

```
SELECT c1, c2 FROM t WHERE c3 > 100;
```

Example 2: Suppose that we issue this statement:

```
SELECT * FROM v_merge WHERE vc1 < 100;
```

This statement is handled similarly to the previous one, except that vc1 < 100 becomes c1 < 100 and the view WHERE clause is added to the statement WHERE clause using an AND connective (and parentheses are added to make sure the parts of the clause are executed with correct precedence). The resulting statement to be executed becomes:

```
SELECT c1, c2 FROM t WHERE (c3 > 100) AND (c1 < 100);
```

Effectively, the statement to be executed has a WHERE clause of this form:

```
WHERE (select WHERE) AND (view WHERE)
```

If the MERGE algorithm cannot be used, a temporary table must be used instead. MERGE cannot be used if the view contains any of the following constructs:

- Aggregate functions (SUM(), MIN(), MAX(), COUNT(), and so forth)
- DISTINCT
- GROUP BY
- HAVING
- LIMIT
- UNION or UNION ALL
- Subquery in the select list

- Refers only to literal values (in this case, there is no underlying table)

18.5.3 Updatable and Insertable Views

Some views are updatable. That is, you can use them in statements such as `UPDATE`, `DELETE`, or `INSERT` to update the contents of the underlying table. For a view to be updatable, there must be a one-to-one relationship between the rows in the view and the rows in the underlying table. There are also certain other constructs that make a view nonupdatable. To be more specific, a view is not updatable if it contains any of the following:

- Aggregate functions (`SUM()`, `MIN()`, `MAX()`, `COUNT()`, and so forth)
- `DISTINCT`
- `GROUP BY`
- `HAVING`
- `UNION` or `UNION ALL`
- Subquery in the select list
- Certain joins (see additional join discussion later in this section)
- Nonupdatable view in the `FROM` clause
- A subquery in the `WHERE` clause that refers to a table in the `FROM` clause
- Refers only to literal values (in this case, there is no underlying table to update)
- Uses `ALGORITHM = TEMPTABLE` (use of a temporary table always makes a view nonupdatable)
- Multiple references to any column of a base table.

With respect to insertability (being updatable with `INSERT` statements), an updatable view is insertable if it also satisfies these additional requirements for the view columns:

- There must be no duplicate view column names.
- The view must contain all columns in the base table that do not have a default value.
- The view columns must be simple column references and not derived columns. A derived column is one that is not a simple column reference but is derived from an expression. These are examples of derived columns:

  ```
  3.14159
  col1 + 3
  UPPER(col2)
  col3 / col4
  (subquery)
  ```

A view that has a mix of simple column references and derived columns is not insertable, but it can be updatable if you update only those columns that are not derived. Consider this view:

```
CREATE VIEW v AS SELECT col1, 1 AS col2 FROM t;
```

This view is not insertable because `col2` is derived from an expression. But it is updatable if the update does not try to update `col2`. This update is permissible:

```
UPDATE v SET col1 = 0;
```

This update is not permissible because it attempts to update a derived column:

```
UPDATE v SET col2 = 0;
```

It is sometimes possible for a multiple-table view to be updatable, assuming that it can be processed with the `MERGE` algorithm. For this to work, the view must use an inner join (not an outer join or a `UNION`). Also, only a single table in the view definition can be updated, so the `SET` clause must name only columns from one of the tables in the view. Views that use `UNION ALL` are not permitted even though they might be theoretically updatable, because the implementation uses temporary tables to process them.

For a multiple-table updatable view, `INSERT` can work if it inserts into a single table. `DELETE` is not supported.

If a table contains an `AUTO_INCREMENT` column, inserting into an insertable view on the table that does not include the `AUTO_INCREMENT` column does not change the value of `LAST_INSERT_ID()`, because the side effects of inserting default values into columns not part of the view should not be visible.

The `WITH CHECK OPTION` clause can be given for an updatable view to prevent inserts or updates to rows except those for which the `WHERE` clause in the *`select_statement`* is true.

In a `WITH CHECK OPTION` clause for an updatable view, the `LOCAL` and `CASCADED` keywords determine the scope of check testing when the view is defined in terms of another view. The `LOCAL` keyword restricts the `CHECK OPTION` only to the view being defined. `CASCADED` causes the checks for underlying views to be evaluated as well. When neither keyword is given, the default is `CASCADED`. Consider the definitions for the following table and set of views:

```
mysql> CREATE TABLE t1 (a INT);
mysql> CREATE VIEW v1 AS SELECT * FROM t1 WHERE a < 2
    -> WITH CHECK OPTION;
mysql> CREATE VIEW v2 AS SELECT * FROM v1 WHERE a > 0
    -> WITH LOCAL CHECK OPTION;
mysql> CREATE VIEW v3 AS SELECT * FROM v1 WHERE a > 0
    -> WITH CASCADED CHECK OPTION;
```

Here the `v2` and `v3` views are defined in terms of another view, `v1`. `v2` has a `LOCAL` check option, so inserts are tested only against the `v2` check. `v3` has a `CASCADED` check option, so inserts are tested not only against its own check, but against those of underlying views. The following statements illustrate these differences:

```
mysql> INSERT INTO v2 VALUES (2);
Query OK, 1 row affected (0.00 sec)
mysql> INSERT INTO v3 VALUES (2);
ERROR 1369 (HY000): CHECK OPTION failed 'test.v3'
```

MySQL sets a flag, called the view updatability flag, at `CREATE VIEW` time. The flag is set to `YES` (true) if `UPDATE` and `DELETE` (and similar operations) are legal for the view. Otherwise, the flag is set to `NO` (false). The `IS_UPDATABLE` column in the `INFORMATION_SCHEMA.VIEWS` table displays the status of this flag. It means that the server always knows whether a view is updatable. If the view is not updatable, statements such `UPDATE`, `DELETE`, and `INSERT` are illegal and will be rejected. (Note that even if a view is updatable, it might not be possible to insert into it, as described elsewhere in this section.)

The updatability of views may be affected by the value of the `updatable_views_with_limit` system variable. See Section 5.1.4, “Server System Variables”.

18.5.4 View Metadata

Metadata about views can be obtained as follows:

- Query the `VIEWS` table of the `INFORMATION_SCHEMA` database. See Section 19.29, “The `INFORMATION_SCHEMA VIEWS` Table”.
- Use the `SHOW CREATE VIEW` statement. See Section 13.7.5.12, “`SHOW CREATE VIEW` Syntax”.

18.6 Access Control for Stored Programs and Views

Stored programs and views are defined prior to use and, when referenced, execute within a security context that determines their privileges. These privileges are controlled by their `DEFINER` attribute, and, if there is one, their `SQL SECURITY` characteristic.

All stored programs (procedures, functions, triggers, and events) and views can have a `DEFINER` attribute that names a MySQL account. If the `DEFINER` attribute is omitted from a stored program or view definition, the default account is the user who creates the object.

In addition, stored routines (procedures and functions) and views can have a `SQL SECURITY` characteristic with a value of `DEFINER` or `INVOKER` to specify whether the object executes in definer or invoker context. If the `SQL SECURITY` characteristic is omitted, the default is definer context.

Triggers and events have no `SQL SECURITY` characteristic and always execute in definer context. The server invokes these objects automatically as necessary, so there is no invoking user.

Definer and invoker security contexts differ as follows:

- A stored program or view that executes in definer security context executes with the privileges of the account named by its `DEFINER` attribute. These privileges may be entirely different from those of the invoking user. The invoker must have appropriate privileges to reference the object (for example, `EXECUTE` to call a stored procedure or `SELECT` to select from a view), but when the object executes, the invoker's privileges are ignored and only the `DEFINER` account privileges matter. If this account has few privileges, the object is correspondingly limited in the operations it can perform. If the `DEFINER` account is highly privileged (such as a `root` account), the object can perform powerful operations *no matter who invokes it.*
- A stored routine or view that executes in invoker security context can perform only operations for which the invoker has privileges. The `DEFINER` attribute can be specified but has no effect for objects that execute in invoker context.

Consider the following stored procedure:

```
CREATE DEFINER = 'admin'@'localhost' PROCEDURE p1()
SQL SECURITY DEFINER
BEGIN
  UPDATE t1 SET counter = counter + 1;
END;
```

Any user who has the `EXECUTE` privilege for `p1` can invoke it with a `CALL` statement. However, when `p1` executes, it does so in `DEFINER` security context and thus executes with the privileges of `'admin'@'localhost'`, the account named in the `DEFINER` attribute. This account must have the `EXECUTE` privilege for `p1` as well as the `UPDATE` privilege for the table `t1`. Otherwise, the procedure fails.

Now consider this stored procedure, which is identical to `p1` except that its `SQL SECURITY` characteristic is `INVOKER`:

```
CREATE DEFINER = 'admin'@'localhost' PROCEDURE p2()
SQL SECURITY INVOKER
BEGIN
  UPDATE t1 SET counter = counter + 1;
END;
```

p2, unlike p1, executes in INVOKER security context. The DEFINER attribute is irrelevant and p2 executes with the privileges of the invoking user. p2 fails if the invoker lacks the EXECUTE privilege for p2 or the UPDATE privilege for the table t1.

MySQL uses the following rules to control which accounts a user can specify in an object DEFINER attribute:

- You can specify a DEFINER value other than your own account only if you have the SUPER privilege.
- If you do not have the SUPER privilege, the only legal user value is your own account, either specified literally or by using CURRENT_USER. You cannot set the definer to some other account.

To minimize the risk potential for stored program and view creation and use, follow these guidelines:

- For a stored routine or view, use SQL SECURITY INVOKER in the object definition when possible so that it can be used only by users with permissions appropriate for the operations performed by the object.
- If you create definer-context stored programs or views while using an account that has the SUPER privilege, specify an explicit DEFINER attribute that names an account possessing only the privileges required for the operations performed by the object. Specify a highly privileged DEFINER account only when absolutely necessary.
- Administrators can prevent users from specifying highly privileged DEFINER accounts by not granting them the SUPER privilege.
- Definer-context objects should be written keeping in mind that they may be able to access data for which the invoking user has no privileges. In some cases, you can prevent reference to these objects by not granting unauthorized users particular privileges:

  - A stored procedure or function cannot be referenced by a user who does not have the EXECUTE privilege for it.
  - A view cannot be referenced by a user who does not have the appropriate privilege for it (SELECT to select from it, INSERT to insert into it, and so forth).

  However, no such control exists for triggers because users do not reference them directly. A trigger always executes in DEFINER context and is activated by access to the table with which it is associated, even ordinary table accesses by users with no special privileges. If the DEFINER account is highly privileged, the trigger can perform sensitive or dangerous operations. This remains true if the SUPER and TRIGGER privileges needed to create the trigger are revoked from the account of the user who created it. Administrators should be especially careful about granting users that combination of privileges.

18.7 Binary Logging of Stored Programs

The binary log contains information about SQL statements that modify database contents. This information is stored in the form of “events” that describe the modifications. The binary log has two important purposes:

- For replication, the binary log is used on master replication servers as a record of the statements to be sent to slave servers. The master server sends the events contained in its binary log to its slaves, which execute those events to make the same data changes that were made on the master. See Section 16.2, “Replication Implementation”.

- Certain data recovery operations require use of the binary log. After a backup file has been restored, the events in the binary log that were recorded after the backup was made are re-executed. These events bring databases up to date from the point of the backup. See Section 7.3.2, “Using Backups for Recovery”.

However, there are certain binary logging issues that apply with respect to stored programs (stored procedures and functions, triggers, and events), if logging occurs at the statement level:

- In some cases, it is possible that a statement will affect different sets of rows on a master and a slave.
- Replicated statements executed on a slave are processed by the slave SQL thread, which has full privileges. It is possible for a procedure to follow different execution paths on master and slave servers, so a user can write a routine containing a dangerous statement that will execute only on the slave where it is processed by a thread that has full privileges.
- If a stored program that modifies data is nondeterministic, it is not repeatable. This can result in different data on a master and slave, or cause restored data to differ from the original data.

This section describes how MySQL 5.7 handles binary logging for stored programs. It states the current conditions that the implementation places on the use of stored programs, and what you can do to avoid problems. It also provides additional information about the reasons for these conditions.

In general, the issues described here result when binary logging occurs at the SQL statement level. If you use row-based binary logging, the log contains changes made to individual rows as a result of executing SQL statements. When routines or triggers execute, row changes are logged, not the statements that make the changes. For stored procedures, this means that the `CALL` statement is not logged. For stored functions, row changes made within the function are logged, not the function invocation. For triggers, row changes made by the trigger are logged. On the slave side, only the row changes are seen, not the stored program invocation. For general information about row-based logging, see Section 16.1.2, “Replication Formats”.

Unless noted otherwise, the remarks here assume that you have enabled binary logging by starting the server with the `--log-bin` option. (See Section 5.2.4, “The Binary Log”.) If the binary log is not enabled, replication is not possible, nor is the binary log available for data recovery.

The current conditions on the use of stored functions in MySQL 5.7 can be summarized as follows. These conditions do not apply to stored procedures or Event Scheduler events and they do not apply unless binary logging is enabled.

- To create or alter a stored function, you must have the `SUPER` privilege, in addition to the `CREATE ROUTINE` or `ALTER ROUTINE` privilege that is normally required. (Depending on the `DEFINER` value in the function definition, `SUPER` might be required regardless of whether binary logging is enabled. See Section 13.1.12, “`CREATE PROCEDURE` and `CREATE FUNCTION` Syntax”.)
- When you create a stored function, you must declare either that it is deterministic or that it does not modify data. Otherwise, it may be unsafe for data recovery or replication.

  By default, for a `CREATE FUNCTION` statement to be accepted, at least one of `DETERMINISTIC`, `NO SQL`, or `READS SQL DATA` must be specified explicitly. Otherwise an error occurs:

  ```
  ERROR 1418 (HY000): This function has none of DETERMINISTIC, NO SQL,
  or READS SQL DATA in its declaration and binary logging is enabled
  (you *might* want to use the less safe log_bin_trust_function_creators
  variable)
  ```

  This function is deterministic (and does not modify data), so it is safe:

```
CREATE FUNCTION f1(i INT)
RETURNS INT
DETERMINISTIC
READS SQL DATA
BEGIN
  RETURN i;
END;
```

This function uses `UUID()`, which is not deterministic, so the function also is not deterministic and is not safe:

```
CREATE FUNCTION f2()
RETURNS CHAR(36) CHARACTER SET utf8
BEGIN
  RETURN UUID();
END;
```

This function modifies data, so it may not be safe:

```
CREATE FUNCTION f3(p_id INT)
RETURNS INT
BEGIN
  UPDATE t SET modtime = NOW() WHERE id = p_id;
  RETURN ROW_COUNT();
END;
```

Assessment of the nature of a function is based on the “honesty” of the creator: MySQL does not check that a function declared `DETERMINISTIC` is free of statements that produce nondeterministic results.

- Although it is possible to create a deterministic stored function without specifying `DETERMINISTIC`, you cannot execute this function using statement-based binary logging. To execute such a function, you must use row-based or mixed binary logging. Alternatively, if you explicitly specify `DETERMINISTIC` in the function definition, you can use any kind of logging, including statement-based logging.

- To relax the preceding conditions on function creation (that you must have the `SUPER` privilege and that a function must be declared deterministic or to not modify data), set the global `log_bin_trust_function_creators` system variable to 1. By default, this variable has a value of 0, but you can change it like this:

```
mysql> SET GLOBAL log_bin_trust_function_creators = 1;
```

  You can also set this variable by using the `--log-bin-trust-function-creators=1` option when starting the server.

  If binary logging is not enabled, `log_bin_trust_function_creators` does not apply. `SUPER` is not required for function creation unless, as described previously, the `DEFINER` value in the function definition requires it.

- For information about built-in functions that may be unsafe for replication (and thus cause stored functions that use them to be unsafe as well), see Section 16.4.1, “Replication Features and Issues”.

Triggers are similar to stored functions, so the preceding remarks regarding functions also apply to triggers with the following exception: `CREATE TRIGGER` does not have an optional `DETERMINISTIC` characteristic, so triggers are assumed to be always deterministic. However, this assumption might in some cases be invalid. For example, the `UUID()` function is nondeterministic (and does not replicate). You should be careful about using such functions in triggers.

Triggers can update tables, so error messages similar to those for stored functions occur with `CREATE TRIGGER` if you do not have the required privileges. On the slave side, the slave uses the trigger `DEFINER` attribute to determine which user is considered to be the creator of the trigger.

The rest of this section provides additional detail about the logging implementation and its implications. You need not read it unless you are interested in the background on the rationale for the current logging-related conditions on stored routine use. This discussion applies only for statement-based logging, and not for row-based logging, with the exception of the first item: `CREATE` and `DROP` statements are logged as statements regardless of the logging mode.

- The server writes `CREATE EVENT`, `CREATE PROCEDURE`, `CREATE FUNCTION`, `ALTER EVENT`, `ALTER PROCEDURE`, `ALTER FUNCTION`, `DROP EVENT`, `DROP PROCEDURE`, and `DROP FUNCTION` statements to the binary log.

- A stored function invocation is logged as a `SELECT` statement if the function changes data and occurs within a statement that would not otherwise be logged. This prevents nonreplication of data changes that result from use of stored functions in nonlogged statements. For example, `SELECT` statements are not written to the binary log, but a `SELECT` might invoke a stored function that makes changes. To handle this, a `SELECT func_name()` statement is written to the binary log when the given function makes a change. Suppose that the following statements are executed on the master:

  ```
  CREATE FUNCTION f1(a INT) RETURNS INT
  BEGIN
    IF (a < 3) THEN
      INSERT INTO t2 VALUES (a);
    END IF;
    RETURN 0;
  END;

  CREATE TABLE t1 (a INT);
  INSERT INTO t1 VALUES (1),(2),(3);

  SELECT f1(a) FROM t1;
  ```

  When the `SELECT` statement executes, the function `f1()` is invoked three times. Two of those invocations insert a row, and MySQL logs a `SELECT` statement for each of them. That is, MySQL writes the following statements to the binary log:

  ```
  SELECT f1(1);
  SELECT f1(2);
  ```

  The server also logs a `SELECT` statement for a stored function invocation when the function invokes a stored procedure that causes an error. In this case, the server writes the `SELECT` statement to the log along with the expected error code. On the slave, if the same error occurs, that is the expected result and replication continues. Otherwise, replication stops.

- Logging stored function invocations rather than the statements executed by a function has a security implication for replication, which arises from two factors:

  - It is possible for a function to follow different execution paths on master and slave servers.

  - Statements executed on a slave are processed by the slave SQL thread which has full privileges.

  The implication is that although a user must have the `CREATE ROUTINE` privilege to create a function, the user can write a function containing a dangerous statement that will execute only on the slave where it is processed by a thread that has full privileges. For example, if the master and slave servers have server ID values of 1 and 2, respectively, a user on the master server could create and invoke an unsafe function `unsafe_func()` as follows:

```
mysql> delimiter //
mysql> CREATE FUNCTION unsafe_func () RETURNS INT
    -> BEGIN
    ->   IF @@server_id=2 THEN dangerous_statement; END IF;
    ->   RETURN 1;
    -> END;
    -> //
mysql> delimiter ;
mysql> INSERT INTO t VALUES(unsafe_func());
```

The `CREATE FUNCTION` and `INSERT` statements are written to the binary log, so the slave will execute them. Because the slave SQL thread has full privileges, it will execute the dangerous statement. Thus, the function invocation has different effects on the master and slave and is not replication-safe.

To guard against this danger for servers that have binary logging enabled, stored function creators must have the `SUPER` privilege, in addition to the usual `CREATE ROUTINE` privilege that is required. Similarly, to use `ALTER FUNCTION`, you must have the `SUPER` privilege in addition to the `ALTER ROUTINE` privilege. Without the `SUPER` privilege, an error will occur:

```
ERROR 1419 (HY000): You do not have the SUPER privilege and
binary logging is enabled (you *might* want to use the less safe
log_bin_trust_function_creators variable)
```

If you do not want to require function creators to have the `SUPER` privilege (for example, if all users with the `CREATE ROUTINE` privilege on your system are experienced application developers), set the global `log_bin_trust_function_creators` system variable to 1. You can also set this variable by using the `--log-bin-trust-function-creators=1` option when starting the server. If binary logging is not enabled, `log_bin_trust_function_creators` does not apply. `SUPER` is not required for function creation unless, as described previously, the `DEFINER` value in the function definition requires it.

- If a function that performs updates is nondeterministic, it is not repeatable. This can have two undesirable effects:

  - It will make a slave different from the master.

  - Restored data will be different from the original data.

  To deal with these problems, MySQL enforces the following requirement: On a master server, creation and alteration of a function is refused unless you declare the function to be deterministic or to not modify data. Two sets of function characteristics apply here:

  - The `DETERMINISTIC` and `NOT DETERMINISTIC` characteristics indicate whether a function always produces the same result for given inputs. The default is `NOT DETERMINISTIC` if neither characteristic is given. To declare that a function is deterministic, you must specify `DETERMINISTIC` explicitly.

  - The `CONTAINS SQL`, `NO SQL`, `READS SQL DATA`, and `MODIFIES SQL DATA` characteristics provide information about whether the function reads or writes data. Either `NO SQL` or `READS SQL DATA` indicates that a function does not change data, but you must specify one of these explicitly because the default is `CONTAINS SQL` if no characteristic is given.

  By default, for a `CREATE FUNCTION` statement to be accepted, at least one of `DETERMINISTIC`, `NO SQL`, or `READS SQL DATA` must be specified explicitly. Otherwise an error occurs:

```
ERROR 1418 (HY000): This function has none of DETERMINISTIC, NO SQL,
or READS SQL DATA in its declaration and binary logging is enabled
(you *might* want to use the less safe log_bin_trust_function_creators
variable)
```

If you set `log_bin_trust_function_creators` to 1, the requirement that functions be deterministic or not modify data is dropped.

- Stored procedure calls are logged at the statement level rather than at the `CALL` level. That is, the server does not log the `CALL` statement, it logs those statements within the procedure that actually execute. As a result, the same changes that occur on the master will be observed on slave servers. This prevents problems that could result from a procedure having different execution paths on different machines.

  In general, statements executed within a stored procedure are written to the binary log using the same rules that would apply were the statements to be executed in standalone fashion. Some special care is taken when logging procedure statements because statement execution within procedures is not quite the same as in nonprocedure context:

  - A statement to be logged might contain references to local procedure variables. These variables do not exist outside of stored procedure context, so a statement that refers to such a variable cannot be logged literally. Instead, each reference to a local variable is replaced by this construct for logging purposes:

    ```
    NAME_CONST(var_name, var_value)
    ```

    `var_name` is the local variable name, and *`var_value`* is a constant indicating the value that the variable has at the time the statement is logged. `NAME_CONST()` has a value of *`var_value`*, and a “name” of *`var_name`*. Thus, if you invoke this function directly, you get a result like this:

    ```
    mysql> SELECT NAME_CONST('myname', 14);
    +--------+
    | myname |
    +--------+
    |     14 |
    +--------+
    ```

    `NAME_CONST()` enables a logged standalone statement to be executed on a slave with the same effect as the original statement that was executed on the master within a stored procedure.

    The use of `NAME_CONST()` can result in a problem for `CREATE TABLE ... SELECT` statements when the source column expressions refer to local variables. Converting these references to `NAME_CONST()` expressions can result in column names that are different on the master and slave servers, or names that are too long to be legal column identifiers. A workaround is to supply aliases for columns that refer to local variables. Consider this statement when `myvar` has a value of 1:

    ```
    CREATE TABLE t1 SELECT myvar;
    ```

    That will be rewritten as follows:

    ```
    CREATE TABLE t1 SELECT NAME_CONST(myvar, 1);
    ```

    To ensure that the master and slave tables have the same column names, write the statement like this:

```
CREATE TABLE t1 SELECT myvar AS myvar;
```

The rewritten statement becomes:

```
CREATE TABLE t1 SELECT NAME_CONST(myvar, 1) AS myvar;
```

- A statement to be logged might contain references to user-defined variables. To handle this, MySQL writes a SET statement to the binary log to make sure that the variable exists on the slave with the same value as on the master. For example, if a statement refers to a variable @my_var, that statement will be preceded in the binary log by the following statement, where *value* is the value of @my_var on the master:

  ```
  SET @my_var = value;
  ```

- Procedure calls can occur within a committed or rolled-back transaction. Transactional context is accounted for so that the transactional aspects of procedure execution are replicated correctly. That is, the server logs those statements within the procedure that actually execute and modify data, and also logs BEGIN, COMMIT, and ROLLBACK statements as necessary. For example, if a procedure updates only transactional tables and is executed within a transaction that is rolled back, those updates are not logged. If the procedure occurs within a committed transaction, BEGIN and COMMIT statements are logged with the updates. For a procedure that executes within a rolled-back transaction, its statements are logged using the same rules that would apply if the statements were executed in standalone fashion:

  - Updates to transactional tables are not logged.
  - Updates to nontransactional tables are logged because rollback does not cancel them.
  - Updates to a mix of transactional and nontransactional tables are logged surrounded by BEGIN and ROLLBACK so that slaves will make the same changes and rollbacks as on the master.

- A stored procedure call is *not* written to the binary log at the statement level if the procedure is invoked from within a stored function. In that case, the only thing logged is the statement that invokes the function (if it occurs within a statement that is logged) or a DO statement (if it occurs within a statement that is not logged). For this reason, care should be exercised in the use of stored functions that invoke a procedure, even if the procedure is otherwise safe in itself.

Chapter 19 INFORMATION_SCHEMA Tables

Table of Contents

INFORMATION_SCHEMA provides access to database *metadata*, information about the MySQL server such as the name of a database or table, the data type of a column, or access privileges. Other terms that are sometimes used for this information are *data dictionary* and *system catalog*.

Usage Notes for the INFORMATION_SCHEMA Database

INFORMATION_SCHEMA is a database within each MySQL instance, the place that stores information about all the other databases that the MySQL server maintains. The INFORMATION_SCHEMA database contains several read-only tables. They are actually views, not base tables, so there are no files associated with them, and you cannot set triggers on them. Also, there is no database directory with that name.

Although you can select INFORMATION_SCHEMA as the default database with a USE statement, you can only read the contents of tables, not perform INSERT, UPDATE, or DELETE operations on them.

Example

Here is an example of a statement that retrieves information from INFORMATION_SCHEMA:

```
mysql> SELECT table_name, table_type, engine
    -> FROM information_schema.tables
    -> WHERE table_schema = 'db5'
    -> ORDER BY table_name;
+------------+------------+--------+
| table_name | table_type | engine |
+------------+------------+--------+
| fk         | BASE TABLE | InnoDB |
| fk2        | BASE TABLE | InnoDB |
| goto       | BASE TABLE | MyISAM |
| into       | BASE TABLE | MyISAM |
| k          | BASE TABLE | MyISAM |
| kurs       | BASE TABLE | MyISAM |
| loop       | BASE TABLE | MyISAM |
| pk         | BASE TABLE | InnoDB |
| t          | BASE TABLE | MyISAM |
| t2         | BASE TABLE | MyISAM |
| t3         | BASE TABLE | MyISAM |
| t7         | BASE TABLE | MyISAM |
| tables     | BASE TABLE | MyISAM |
| v          | VIEW       | NULL   |
| v2         | VIEW       | NULL   |
| v3         | VIEW       | NULL   |
| v56        | VIEW       | NULL   |
+------------+------------+--------+
17 rows in set (0.01 sec)
```

Explanation: The statement requests a list of all the tables in database db5, showing just three pieces of information: the name of the table, its type, and its storage engine.

Character Set Considerations

The definition for character columns (for example, TABLES.TABLE_NAME) is generally VARCHAR(*N*) CHARACTER SET utf8 where *N* is at least 64. MySQL uses the default collation for this character set (utf8_general_ci) for all searches, sorts, comparisons, and other string operations on such columns.

Because some MySQL objects are represented as files, searches in INFORMATION_SCHEMA string columns can be affected by file system case sensitivity. For more information, see Section 10.1.7.9, “Collation and INFORMATION_SCHEMA Searches”.

INFORMATION_SCHEMA as Alternative to SHOW Statements

The SELECT ... FROM INFORMATION_SCHEMA statement is intended as a more consistent way to provide access to the information provided by the various SHOW statements that MySQL supports (SHOW DATABASES, SHOW TABLES, and so forth). Using SELECT has these advantages, compared to SHOW:

- It conforms to Codd's rules, because all access is done on tables.
- You can use the familiar syntax of the SELECT statement, and only need to learn some table and column names.
- The implementor need not worry about adding keywords.
- You can filter, sort, concatenate, and transform the results from INFORMATION_SCHEMA queries into whatever format your application needs, such as a data structure or a text representation to parse.
- This technique is more interoperable with other database systems. For example, Oracle Database users are familiar with querying tables in the Oracle data dictionary.

Because SHOW is familiar and widely used, the SHOW statements remain as an alternative. In fact, along with the implementation of INFORMATION_SCHEMA, there are enhancements to SHOW as described in Section 19.31, “Extensions to SHOW Statements”.

Privileges

Each MySQL user has the right to access these tables, but can see only the rows in the tables that correspond to objects for which the user has the proper access privileges. In some cases (for example, the ROUTINE_DEFINITION column in the INFORMATION_SCHEMA.ROUTINES table), users who have insufficient privileges see NULL. These restrictions do not apply for InnoDB tables; you can see them with only the PROCESS privilege.

The same privileges apply to selecting information from INFORMATION_SCHEMA and viewing the same information through SHOW statements. In either case, you must have some privilege on an object to see information about it.

Performance Considerations

INFORMATION_SCHEMA queries that search for information from more than one database might take a long time and impact performance. To check the efficiency of a query, you can use EXPLAIN. For information about using EXPLAIN output to tune INFORMATION_SCHEMA queries, see Section 8.2.4, “Optimizing INFORMATION_SCHEMA Queries”.

Standards Considerations

The implementation for the INFORMATION_SCHEMA table structures in MySQL follows the ANSI/ISO SQL:2003 standard Part 11 *Schemata*. Our intent is approximate compliance with SQL:2003 core feature F021 *Basic information schema*.

Users of SQL Server 2000 (which also follows the standard) may notice a strong similarity. However, MySQL has omitted many columns that are not relevant for our implementation, and added columns that are MySQL-specific. One such column is the ENGINE column in the INFORMATION_SCHEMA.TABLES table.

Although other DBMSs use a variety of names, like syscat or system, the standard name is INFORMATION_SCHEMA.

To avoid using any name that is reserved in the standard or in DB2, SQL Server, or Oracle, we changed the names of some columns marked “MySQL extension”. (For example, we changed COLLATION to TABLE_COLLATION in the TABLES table.) See the list of reserved words near the end of this article: http://web.archive.org/web/20070409075643rn_1/www.dbazine.com/db2/db2-disarticles/gulutzan5.

Conventions in the INFORMATION_SCHEMA Reference Sections

The following sections describe each of the tables and columns in INFORMATION_SCHEMA. For each column, there are three pieces of information:

- “INFORMATION_SCHEMA Name” indicates the name for the column in the INFORMATION_SCHEMA table. This corresponds to the standard SQL name unless the “Remarks” field says “MySQL extension.”
- “SHOW Name” indicates the equivalent field name in the closest SHOW statement, if there is one.
- “Remarks” provides additional information where applicable. If this field is NULL, it means that the value of the column is always NULL. If this field says “MySQL extension,” the column is a MySQL extension to standard SQL.

Many sections indicate what SHOW statement is equivalent to a SELECT that retrieves information from INFORMATION_SCHEMA. For SHOW statements that display information for the default database if you omit a FROM *db_name* clause, you can often select information for the default database by adding an AND TABLE_SCHEMA = SCHEMA() condition to the WHERE clause of a query that retrieves information from an INFORMATION_SCHEMA table.

For information about INFORMATION_SCHEMA tables specific to the InnoDB storage engine, see Section 19.30, “INFORMATION_SCHEMA Tables for InnoDB”.

For answers to questions that are often asked concerning the INFORMATION_SCHEMA database, see Section B.7, “MySQL 5.7 FAQ: INFORMATION_SCHEMA”.

19.1 The INFORMATION_SCHEMA CHARACTER_SETS Table

The CHARACTER_SETS table provides information about available character sets.

INFORMATION_SCHEMA Name	SHOW Name	Remarks
CHARACTER_SET_NAME	Charset	
DEFAULT_COLLATE_NAME	Default collation	
DESCRIPTION	Description	MySQL extension
MAXLEN	Maxlen	MySQL extension

The following statements are equivalent:

```
SELECT * FROM INFORMATION_SCHEMA.CHARACTER_SETS
```

```
  [WHERE CHARACTER_SET_NAME LIKE 'wild']

SHOW CHARACTER SET
  [LIKE 'wild']
```

19.2 The INFORMATION_SCHEMA COLLATIONS Table

The COLLATIONS table provides information about collations for each character set.

INFORMATION_SCHEMA Name	**SHOW Name**	**Remarks**
COLLATION_NAME	Collation	
CHARACTER_SET_NAME	Charset	MySQL extension
ID	Id	MySQL extension
IS_DEFAULT	Default	MySQL extension
IS_COMPILED	Compiled	MySQL extension
SORTLEN	Sortlen	MySQL extension

- COLLATION_NAME is the collation name.
- CHARACTER_SET_NAME is the name of the character set with which the collation is associated.
- ID is the collation ID.
- IS_DEFAULT indicates whether the collation is the default for its character set.
- IS_COMPILED indicates whether the character set is compiled into the server.
- SORTLEN is related to the amount of memory required to sort strings expressed in the character set.

Collation information is also available from the SHOW COLLATION statement. The following statements are equivalent:

```
SELECT COLLATION_NAME FROM INFORMATION_SCHEMA.COLLATIONS
  [WHERE COLLATION_NAME LIKE 'wild']

SHOW COLLATION
  [LIKE 'wild']
```

19.3 The INFORMATION_SCHEMA COLLATION_CHARACTER_SET_APPLICABILITY Table

The COLLATION_CHARACTER_SET_APPLICABILITY table indicates what character set is applicable for what collation. The columns are equivalent to the first two display fields that we get from SHOW COLLATION.

INFORMATION_SCHEMA Name	**SHOW Name**	**Remarks**
COLLATION_NAME	Collation	
CHARACTER_SET_NAME	Charset	

19.4 The INFORMATION_SCHEMA COLUMNS Table

The COLUMNS table provides information about columns in tables.

INFORMATION_SCHEMA Name	**SHOW Name**	**Remarks**
TABLE_CATALOG		def
TABLE_SCHEMA		
TABLE_NAME		
COLUMN_NAME	Field	
ORDINAL_POSITION		see notes
COLUMN_DEFAULT	Default	
IS_NULLABLE	Null	
DATA_TYPE	Type	
CHARACTER_MAXIMUM_LENGTH	Type	
CHARACTER_OCTET_LENGTH		
NUMERIC_PRECISION	Type	
NUMERIC_SCALE	Type	
DATETIME_PRECISION	Type	
CHARACTER_SET_NAME		
COLLATION_NAME	Collation	
COLUMN_TYPE	Type	MySQL extension
COLUMN_KEY	Key	MySQL extension
EXTRA	Extra	MySQL extension
PRIVILEGES	Privileges	MySQL extension
COLUMN_COMMENT	Comment	MySQL extension

Notes:

- In SHOW, the Type display includes values from several different COLUMNS columns.
- ORDINAL_POSITION is necessary because you might want to say ORDER BY ORDINAL_POSITION. Unlike SHOW, SELECT does not have automatic ordering.
- CHARACTER_OCTET_LENGTH should be the same as CHARACTER_MAXIMUM_LENGTH, except for multi-byte character sets.
- CHARACTER_SET_NAME can be derived from Collation. For example, if you say SHOW FULL COLUMNS FROM t, and you see in the Collation column a value of latin1_swedish_ci, the character set is what is before the first underscore: latin1.

The following statements are nearly equivalent:

```
SELECT COLUMN_NAME, DATA_TYPE, IS_NULLABLE, COLUMN_DEFAULT
  FROM INFORMATION_SCHEMA.COLUMNS
  WHERE table_name = 'tbl_name'
  [AND table_schema = 'db_name']
  [AND column_name LIKE 'wild']

SHOW COLUMNS
  FROM tbl_name
  [FROM db_name]
  [LIKE 'wild']
```

19.5 The INFORMATION_SCHEMA COLUMN_PRIVILEGES Table

The COLUMN_PRIVILEGES table provides information about column privileges. This information comes from the mysql.columns_priv grant table.

INFORMATION_SCHEMA Name	SHOW Name	Remarks
GRANTEE		'user_name'@'host_name' value
TABLE_CATALOG		def
TABLE_SCHEMA		
TABLE_NAME		
COLUMN_NAME		
PRIVILEGE_TYPE		
IS_GRANTABLE		

Notes:

- In the output from SHOW FULL COLUMNS, the privileges are all in one field and in lowercase, for example, select,insert,update,references. In COLUMN_PRIVILEGES, there is one privilege per row, in uppercase.
- PRIVILEGE_TYPE can contain one (and only one) of these values: SELECT, INSERT, UPDATE, REFERENCES.
- If the user has GRANT OPTION privilege, IS_GRANTABLE should be YES. Otherwise, IS_GRANTABLE should be NO. The output does not list GRANT OPTION as a separate privilege.

The following statements are *not* equivalent:

```
SELECT ... FROM INFORMATION_SCHEMA.COLUMN_PRIVILEGES

SHOW GRANTS ...
```

19.6 The INFORMATION_SCHEMA ENGINES Table

The ENGINES table provides information about storage engines.

INFORMATION_SCHEMA Name	SHOW Name	Remarks
ENGINE	Engine	MySQL extension
SUPPORT	Support	MySQL extension
COMMENT	Comment	MySQL extension
TRANSACTIONS	Transactions	MySQL extension
XA	XA	MySQL extension
SAVEPOINTS	Savepoints	MySQL extension

Notes:

- The ENGINES table is a nonstandard table. Its contents correspond to the columns of the SHOW ENGINES statement. For descriptions of its columns, see Section 13.7.5.15, “SHOW ENGINES Syntax”.

See also Section 13.7.5.15, “SHOW ENGINES Syntax”.

19.7 The INFORMATION_SCHEMA EVENTS Table

The EVENTS table provides information about scheduled events, which are discussed in Section 18.4, “Using the Event Scheduler”. The SHOW Name values correspond to column names of the SHOW EVENTS statement.

INFORMATION_SCHEMA Name	SHOW Name	Remarks
EVENT_CATALOG		def, MySQL extension
EVENT_SCHEMA	Db	MySQL extension
EVENT_NAME	Name	MySQL extension
DEFINER	Definer	MySQL extension
TIME_ZONE	Time zone	MySQL extension
EVENT_BODY		MySQL extension
EVENT_DEFINITION		MySQL extension
EVENT_TYPE	Type	MySQL extension
EXECUTE_AT	Execute at	MySQL extension
INTERVAL_VALUE	Interval value	MySQL extension
INTERVAL_FIELD	Interval field	MySQL extension
SQL_MODE		MySQL extension
STARTS	Starts	MySQL extension
ENDS	Ends	MySQL extension
STATUS	Status	MySQL extension
ON_COMPLETION		MySQL extension
CREATED		MySQL extension
LAST_ALTERED		MySQL extension
LAST_EXECUTED		MySQL extension
EVENT_COMMENT		MySQL extension
ORIGINATOR	Originator	MySQL extension
CHARACTER_SET_CLIENT	character_set_client	MySQL extension
COLLATION_CONNECTION	collation_connection	MySQL extension
DATABASE_COLLATION	Database Collation	MySQL extension

Notes:

- The EVENTS table is a nonstandard table.
- EVENT_CATALOG: The value of this column is always def.
- EVENT_SCHEMA: The name of the schema (database) to which this event belongs.

- EVENT_NAME: The name of the event.
- DEFINER: The account of the user who created the event, in '*user_name*'@'*host_name*' format.
- TIME_ZONE: The event time zone, which is the time zone used for scheduling the event and that is in effect within the event as it executes. The default value is SYSTEM.
- EVENT_BODY: The language used for the statements in the event's DO clause; in MySQL 5.7, this is always SQL.

  This column is not to be confused with the column of the same name (now named EVENT_DEFINITION) that existed in earlier MySQL versions.
- EVENT_DEFINITION: The text of the SQL statement making up the event's DO clause; in other words, the statement executed by this event.
- EVENT_TYPE: The event repetition type, either ONE TIME (transient) or RECURRING (repeating).
- EXECUTE_AT: For a one-time event, this is the DATETIME value specified in the AT clause of the CREATE EVENT statement used to create the event, or of the last ALTER EVENT statement that modified the event. The value shown in this column reflects the addition or subtraction of any INTERVAL value included in the event's AT clause. For example, if an event is created using ON SCHEDULE AT CURRENT_TIMESTAMP + '1:6' DAY_HOUR, and the event was created at 2006-02-09 14:05:30, the value shown in this column would be '2006-02-10 20:05:30'.

  If the event's timing is determined by an EVERY clause instead of an AT clause (that is, if the event is recurring), the value of this column is NULL.
- INTERVAL_VALUE: For recurring events, this column contains the numeric portion of the event's EVERY clause.

  For a one-time event (that is, an event whose timing is determined by an AT clause), this column is NULL.
- INTERVAL_FIELD: For recurring events, this column contains the units portion of the EVERY clause governing the timing of the event. Thus, this column contains a value such as 'YEAR', 'QUARTER', 'DAY', and so on.

  For a one-time event (that is, an event whose timing is determined by an AT clause), this column is NULL.
- SQL_MODE: The SQL mode in effect when the event was created or altered, and under which the event executes. For the permitted values, see Section 5.1.7, “Server SQL Modes”.
- STARTS: For a recurring event whose definition includes a STARTS clause, this column contains the corresponding DATETIME value. As with the EXECUTE_AT column, this value resolves any expressions used.

  If there is no STARTS clause affecting the timing of the event, this column is NULL
- ENDS: For a recurring event whose definition includes a ENDS clause, this column contains the corresponding DATETIME value. As with the EXECUTE_AT column, this value resolves any expressions used.

  If there is no ENDS clause affecting the timing of the event, this column is NULL.
- STATUS: One of the three values ENABLED, DISABLED, or SLAVESIDE_DISABLED.

SLAVESIDE_DISABLED indicates that the creation of the event occurred on another MySQL server acting as a replication master and was replicated to the current MySQL server which is acting as a slave, but the event is not presently being executed on the slave. See Section 16.4.1.11, “Replication of Invoked Features”, for more information.

- ON_COMPLETION: One of the two values PRESERVE or NOT PRESERVE.

- CREATED: The date and time when the event was created. This is a TIMESTAMP value.

- LAST_ALTERED: The date and time when the event was last modified. This is a TIMESTAMP value. If the event has not been modified since its creation, this column holds the same value as the CREATED column.

- LAST_EXECUTED: The date and time when the event last executed. A DATETIME value. If the event has never executed, this column is NULL.

  LAST_EXECUTED indicates when the event started. As a result, the ENDS column is never less than LAST_EXECUTED.

- EVENT_COMMENT: The text of a comment, if the event has one. If not, the value of this column is an empty string.

- ORIGINATOR: The server ID of the MySQL server on which the event was created; used in replication. The default value is 0.

- CHARACTER_SET_CLIENT: The session value of the character_set_client system variable when the event was created.

- COLLATION_CONNECTION: The session value of the collation_connection system variable when the event was created.

- DATABASE_COLLATION: The collation of the database with which the event is associated.

Example: Suppose that the user jon@ghidora creates an event named e_daily, and then modifies it a few minutes later using an ALTER EVENT statement, as shown here:

```
DELIMITER |

CREATE EVENT e_daily
    ON SCHEDULE
      EVERY 1 DAY
    COMMENT 'Saves total number of sessions then clears the table each day'
    DO
      BEGIN
        INSERT INTO site_activity.totals (time, total)
          SELECT CURRENT_TIMESTAMP, COUNT(*)
            FROM site_activity.sessions;
        DELETE FROM site_activity.sessions;
      END |

DELIMITER ;

ALTER EVENT e_daily
    ENABLED;
```

(Note that comments can span multiple lines.)

This user can then run the following SELECT statement, and obtain the output shown:

```
mysql> SELECT * FROM INFORMATION_SCHEMA.EVENTS
     > WHERE EVENT_NAME = 'e_daily'
     > AND EVENT_SCHEMA = 'myschema'\G
*************************** 1. row ***************************
       EVENT_CATALOG: def
        EVENT_SCHEMA: test
          EVENT_NAME: e_daily
             DEFINER: me@localhost
           TIME_ZONE: SYSTEM
          EVENT_BODY: SQL
    EVENT_DEFINITION: BEGIN
        INSERT INTO site_activity.totals (time, total)
          SELECT CURRENT_TIMESTAMP, COUNT(*)
            FROM site_activity.sessions;
        DELETE FROM site_activity.sessions;
      END
          EVENT_TYPE: RECURRING
          EXECUTE_AT: NULL
      INTERVAL_VALUE: 1
      INTERVAL_FIELD: DAY
            SQL_MODE:
              STARTS: 2008-09-03 12:13:39
                ENDS: NULL
              STATUS: ENABLED
       ON_COMPLETION: NOT PRESERVE
             CREATED: 2008-09-03 12:13:39
        LAST_ALTERED: 2008-09-03 12:13:39
       LAST_EXECUTED: NULL
       EVENT_COMMENT: Saves total number of sessions then clears the
                      table each day
          ORIGINATOR: 1
CHARACTER_SET_CLIENT: latin1
COLLATION_CONNECTION: latin1_swedish_ci
  DATABASE_COLLATION: latin1_swedish_ci
```

Times in the EVENTS table are displayed using the event time zone or the current session time zone, as described in Section 18.4.4, “Event Metadata”.

See also Section 13.7.5.17, “SHOW EVENTS Syntax”.

19.8 The INFORMATION_SCHEMA FILES Table

The FILES table provides information about the files in which MySQL tablespace data is stored.

INFORMATION_SCHEMA Name	SHOW Name	Remarks
FILE_ID		MySQL extension
FILE_NAME		MySQL extension
FILE_TYPE		MySQL extension
TABLESPACE_NAME		MySQL extension
TABLE_CATALOG		MySQL extension
TABLE_SCHEMA		MySQL extension
TABLE_NAME		MySQL extension
LOGFILE_GROUP_NAME		MySQL extension
LOGFILE_GROUP_NUMBER		MySQL extension
ENGINE		MySQL extension
FULLTEXT_KEYS		MySQL extension

INFORMATION_SCHEMA Name	**SHOW Name**	**Remarks**
DELETED_ROWS		MySQL extension
UPDATE_COUNT		MySQL extension
FREE_EXTENTS		MySQL extension
TOTAL_EXTENTS		MySQL extension
EXTENT_SIZE		MySQL extension
INITIAL_SIZE		MySQL extension
MAXIMUM_SIZE		MySQL extension
AUTOEXTEND_SIZE		MySQL extension
CREATION_TIME		MySQL extension
LAST_UPDATE_TIME		MySQL extension
LAST_ACCESS_TIME		MySQL extension
RECOVER_TIME		MySQL extension
TRANSACTION_COUNTER		MySQL extension
VERSION		MySQL extension
ROW_FORMAT		MySQL extension
TABLE_ROWS		MySQL extension
AVG_ROW_LENGTH		MySQL extension
DATA_LENGTH		MySQL extension
MAX_DATA_LENGTH		MySQL extension
INDEX_LENGTH		MySQL extension
DATA_FREE		MySQL extension
CREATE_TIME		MySQL extension
UPDATE_TIME		MySQL extension
CHECK_TIME		MySQL extension
CHECKSUM		MySQL extension
STATUS		MySQL extension
EXTRA		MySQL extension

Notes:

- FILE_ID column values are auto-generated.
- FILE_NAME is the name of a data file created by CREATE TABLESPACE or ALTER TABLESPACE.
- FILE_TYPE is the tablespace file type.
- TABLESPACE_NAME is the name of the tablespace with which the file is associated.
- Currently, the value of the TABLESPACE_CATALOG column is always NULL.
- TABLE_NAME is the name of the table with which the file is associated, if any.
- The EXTENT_SIZE is always 0.
- There are no SHOW statements associated with the FILES table.

19.9 The INFORMATION_SCHEMA GLOBAL_STATUS and SESSION_STATUS Tables

The GLOBAL_STATUS and SESSION_STATUS tables provide information about server status variables. Their contents correspond to the information produced by the SHOW GLOBAL STATUS and SHOW SESSION STATUS statements (see Section 13.7.5.34, "SHOW STATUS Syntax").

INFORMATION_SCHEMA Name	**SHOW Name**	**Remarks**
VARIABLE_NAME	Variable_name	
VARIABLE_VALUE	Value	

Notes:

- The VARIABLE_VALUE column for each of these tables is defined as VARCHAR(1024).

19.10 The INFORMATION_SCHEMA GLOBAL_VARIABLES and SESSION_VARIABLES Tables

The GLOBAL_VARIABLES and SESSION_VARIABLES tables provide information about server status variables. Their contents correspond to the information produced by the SHOW GLOBAL VARIABLES and SHOW SESSION VARIABLES statements (see Section 13.7.5.38, "SHOW VARIABLES Syntax").

INFORMATION_SCHEMA Name	**SHOW Name**	**Remarks**
VARIABLE_NAME	Variable_name	
VARIABLE_VALUE	Value	

Notes:

- The VARIABLE_VALUE column for each of these tables is defined as VARCHAR(1024).

19.11 The INFORMATION_SCHEMA KEY_COLUMN_USAGE Table

The KEY_COLUMN_USAGE table describes which key columns have constraints.

INFORMATION_SCHEMA Name	**SHOW Name**	**Remarks**
CONSTRAINT_CATALOG		def
CONSTRAINT_SCHEMA		
CONSTRAINT_NAME		
TABLE_CATALOG		def
TABLE_SCHEMA		
TABLE_NAME		
COLUMN_NAME		
ORDINAL_POSITION		
POSITION_IN_UNIQUE_CONSTRAINT		
REFERENCED_TABLE_SCHEMA		
REFERENCED_TABLE_NAME		

INFORMATION_SCHEMA Name	SHOW Name	Remarks
REFERENCED_COLUMN_NAME		

Notes:

- If the constraint is a foreign key, then this is the column of the foreign key, not the column that the foreign key references.
- The value of ORDINAL_POSITION is the column's position within the constraint, not the column's position within the table. Column positions are numbered beginning with 1.
- The value of POSITION_IN_UNIQUE_CONSTRAINT is NULL for unique and primary-key constraints. For foreign-key constraints, it is the ordinal position in key of the table that is being referenced.

  Suppose that there are two tables name t1 and t3 that have the following definitions:

  ```
  CREATE TABLE t1
  (
      s1 INT,
      s2 INT,
      s3 INT,
      PRIMARY KEY(s3)
  ) ENGINE=InnoDB;

  CREATE TABLE t3
  (
      s1 INT,
      s2 INT,
      s3 INT,
      KEY(s1),
      CONSTRAINT CO FOREIGN KEY (s2) REFERENCES t1(s3)
  ) ENGINE=InnoDB;
  ```

  For those two tables, the KEY_COLUMN_USAGE table has two rows:

  - One row with CONSTRAINT_NAME = 'PRIMARY', TABLE_NAME = 't1', COLUMN_NAME = 's3', ORDINAL_POSITION = 1, POSITION_IN_UNIQUE_CONSTRAINT = NULL.
  - One row with CONSTRAINT_NAME = 'CO', TABLE_NAME = 't3', COLUMN_NAME = 's2', ORDINAL_POSITION = 1, POSITION_IN_UNIQUE_CONSTRAINT = 1.

19.12 The INFORMATION_SCHEMA OPTIMIZER_TRACE Table

The OPTIMIZER_TRACE table provides information produced by the optimizer tracing capability. To enable tracking, use the optimizer_trace system variable. For details, see MySQL Internals: Tracing the Optimizer.

19.13 The INFORMATION_SCHEMA PARAMETERS Table

The PARAMETERS table provides information about stored procedure and function parameters, and about return values for stored functions. Parameter information is similar to the contents of the param_list column in the mysql.proc table.

INFORMATION_SCHEMA Name	mysql.proc Name	Remarks
SPECIFIC_CATALOG		def
SPECIFIC_SCHEMA	db	routine database

INFORMATION_SCHEMA Name	**mysql.proc Name**	**Remarks**
SPECIFIC_NAME	name	routine name
ORDINAL_POSITION		1, 2, 3, ... for parameters, 0 for function RETURNS clause
PARAMETER_MODE		IN, OUT, INOUT (NULL for RETURNS)
PARAMETER_NAME		parameter name (NULL for RETURNS)
DATA_TYPE		same as for COLUMNS table
CHARACTER_MAXIMUM_LENGTH		same as for COLUMNS table
CHARACTER_OCTET_LENGTH		same as for COLUMNS table
NUMERIC_PRECISION		same as for COLUMNS table
NUMERIC_SCALE		same as for COLUMNS table
DATETIME_PRECISION		same as for COLUMNS table
CHARACTER_SET_NAME		same as for COLUMNS table
COLLATION_NAME		same as for COLUMNS table
DTD_IDENTIFIER		same as for COLUMNS table
ROUTINE_TYPE	type	same as for ROUTINES table

Notes:

- For successive parameters of a stored procedure or function, the ORDINAL_POSITION values are 1, 2, 3, and so forth. For a stored function, there is also a row that describes the data type for the RETURNS clause. The return value is not a true parameter, so the row that describes it has these unique characteristics:
  - The ORDINAL_POSITION value is 0.
  - The PARAMETER_NAME and PARAMETER_MODE values are NULL because the return value has no name and the mode does not apply.

19.14 The INFORMATION_SCHEMA PARTITIONS Table

The PARTITIONS table provides information about table partitions. See Chapter 17, *Partitioning*, for more information about partitioning tables.

INFORMATION_SCHEMA Name	**SHOW Name**	**Remarks**
TABLE_CATALOG		MySQL extension
TABLE_SCHEMA		MySQL extension
TABLE_NAME		MySQL extension
PARTITION_NAME		MySQL extension
SUBPARTITION_NAME		MySQL extension
PARTITION_ORDINAL_POSITION		MySQL extension
SUBPARTITION_ORDINAL_POSITION		MySQL extension
PARTITION_METHOD		MySQL extension
SUBPARTITION_METHOD		MySQL extension

INFORMATION_SCHEMA Name	**SHOW Name**	**Remarks**
PARTITION_EXPRESSION		MySQL extension
SUBPARTITION_EXPRESSION		MySQL extension
PARTITION_DESCRIPTION		MySQL extension
TABLE_ROWS		MySQL extension
AVG_ROW_LENGTH		MySQL extension
DATA_LENGTH		MySQL extension
MAX_DATA_LENGTH		MySQL extension
INDEX_LENGTH		MySQL extension
DATA_FREE		MySQL extension
CREATE_TIME		MySQL extension
UPDATE_TIME		MySQL extension
CHECK_TIME		MySQL extension
CHECKSUM		MySQL extension
PARTITION_COMMENT		MySQL extension
NODEGROUP		MySQL extension
TABLESPACE_NAME		MySQL extension

Notes:

- The PARTITIONS table is a nonstandard table.

  Each record in this table corresponds to an individual partition or subpartition of a partitioned table.

- TABLE_CATALOG: This column is always def.
- TABLE_SCHEMA: This column contains the name of the database to which the table belongs.
- TABLE_NAME: This column contains the name of the table containing the partition.
- PARTITION_NAME: The name of the partition.
- SUBPARTITION_NAME: If the PARTITIONS table record represents a subpartition, then this column contains the name of subpartition; otherwise it is NULL.
- PARTITION_ORDINAL_POSITION: All partitions are indexed in the same order as they are defined, with 1 being the number assigned to the first partition. The indexing can change as partitions are added, dropped, and reorganized; the number shown is this column reflects the current order, taking into account any indexing changes.
- SUBPARTITION_ORDINAL_POSITION: Subpartitions within a given partition are also indexed and reindexed in the same manner as partitions are indexed within a table.
- PARTITION_METHOD: One of the values RANGE, LIST, HASH, LINEAR HASH, KEY, or LINEAR KEY; that is, one of the available partitioning types as discussed in Section 17.2, “Partitioning Types”.
- SUBPARTITION_METHOD: One of the values HASH, LINEAR HASH, KEY, or LINEAR KEY; that is, one of the available subpartitioning types as discussed in Section 17.2.6, “Subpartitioning”.
- PARTITION_EXPRESSION: This is the expression for the partitioning function used in the CREATE TABLE or ALTER TABLE statement that created the table's current partitioning scheme.

For example, consider a partitioned table created in the test database using this statement:

```
CREATE TABLE tp (
    c1 INT,
    c2 INT,
    c3 VARCHAR(25)
)
PARTITION BY HASH(c1 + c2)
PARTITIONS 4;
```

The PARTITION_EXPRESSION column in a PARTITIONS table record for a partition from this table displays c1 + c2, as shown here:

```
mysql> SELECT DISTINCT PARTITION_EXPRESSION
     >     FROM INFORMATION_SCHEMA.PARTITIONS
     >     WHERE TABLE_NAME='tp' AND TABLE_SCHEMA='test';
+----------------------+
| PARTITION_EXPRESSION |
+----------------------+
| c1 + c2              |
+----------------------+
1 row in set (0.09 sec)
```

- SUBPARTITION_EXPRESSION: This works in the same fashion for the subpartitioning expression that defines the subpartitioning for a table as PARTITION_EXPRESSION does for the partitioning expression used to define a table's partitioning.

  If the table has no subpartitions, then this column is NULL.

- PARTITION_DESCRIPTION: This column is used for RANGE and LIST partitions. For a RANGE partition, it contains the value set in the partition's VALUES LESS THAN clause, which can be either an integer or MAXVALUE. For a LIST partition, this column contains the values defined in the partition's VALUES IN clause, which is a comma-separated list of integer values.

  For partitions whose PARTITION_METHOD is other than RANGE or LIST, this column is always NULL.

- TABLE_ROWS: The number of table rows in the partition.

  For partitioned InnoDB tables, the row count given in the TABLE_ROWS column is only an estimated value used in SQL optimization, and may not always be exact.

- AVG_ROW_LENGTH: The average length of the rows stored in this partition or subpartition, in bytes.

  This is the same as DATA_LENGTH divided by TABLE_ROWS.

- DATA_LENGTH: The total length of all rows stored in this partition or subpartition, in bytes—that is, the total number of bytes stored in the partition or subpartition.

- MAX_DATA_LENGTH: The maximum number of bytes that can be stored in this partition or subpartition.

- INDEX_LENGTH: The length of the index file for this partition or subpartition, in bytes.

- DATA_FREE: The number of bytes allocated to the partition or subpartition but not used.

- CREATE_TIME: The time of the partition's or subpartition's creation.

- UPDATE_TIME: The time that the partition or subpartition was last modified.

- CHECK_TIME: The last time that the table to which this partition or subpartition belongs was checked.

Note

Some storage engines do not update this time; for tables using these storage engines, this value is always NULL.

- CHECKSUM: The checksum value, if any; otherwise, this column is NULL.
- PARTITION_COMMENT: This column contains the text of any comment made for the partition.

  In MySQL 5.7, the maximum length for a partition comment is defined as 1024 characters, and the display width of the PARTITION_COMMENT column is also 1024, characters to match this limit (Bug #11748924, Bug #37728).

  The default value for this column is an empty string.
- NODEGROUP: This is the nodegroup to which the partition belongs. This is relevant only to MySQL Cluster tables; otherwise the value of this column is always 0.
- TABLESPACE_NAME: This column contains the name of the tablespace to which the partition belongs. Currently, the value of this column is always DEFAULT.
- A nonpartitioned table has one record in INFORMATION_SCHEMA.PARTITIONS; however, the values of the PARTITION_NAME, SUBPARTITION_NAME, PARTITION_ORDINAL_POSITION, SUBPARTITION_ORDINAL_POSITION, PARTITION_METHOD, SUBPARTITION_METHOD, PARTITION_EXPRESSION, SUBPARTITION_EXPRESSION, and PARTITION_DESCRIPTION columns are all NULL. (The PARTITION_COMMENT column in this case is blank.)

19.15 The INFORMATION_SCHEMA PLUGINS Table

The PLUGINS table provides information about server plugins.

INFORMATION_SCHEMA Name	SHOW Name	Remarks
PLUGIN_NAME	Name	MySQL extension
PLUGIN_VERSION		MySQL extension
PLUGIN_STATUS	Status	MySQL extension
PLUGIN_TYPE	Type	MySQL extension
PLUGIN_TYPE_VERSION		MySQL extension
PLUGIN_LIBRARY	Library	MySQL extension
PLUGIN_LIBRARY_VERSION		MySQL extension
PLUGIN_AUTHOR		MySQL extension
PLUGIN_DESCRIPTION		MySQL extension
PLUGIN_LICENSE	License	MySQL extension
LOAD_OPTION		MySQL extension

Notes:

- The PLUGINS table is a nonstandard table.
- PLUGIN_NAME is the name used to refer to the plugin in statements such as INSTALL PLUGIN and UNINSTALL PLUGIN.

- PLUGIN_VERSION is the version from the plugin's general type descriptor.
- PLUGIN_STATUS indicates the plugin status, one of ACTIVE, INACTIVE, DISABLED, or DELETED.
- PLUGIN_TYPE indicates the type of plugin, such as STORAGE ENGINE, INFORMATION_SCHEMA, or AUTHENTICATION.
- PLUGIN_TYPE_VERSION is the version from the plugin's type-specific descriptor.
- PLUGIN_LIBRARY is the name of the plugin shared object file. This is the name used to refer to the plugin file in statements such as INSTALL PLUGIN and UNINSTALL PLUGIN. This file is located in the directory named by the plugin_dir system variable. If the library name is NULL, the plugin is compiled in and cannot be uninstalled with UNINSTALL PLUGIN.
- PLUGIN_LIBRARY_VERSION indicates the plugin API interface version.
- PLUGIN_AUTHOR names the plugin author.
- PLUGIN_DESCRIPTION provides a short description of the plugin.
- PLUGIN_LICENSE indicates how the plugin is licensed; for example, GPL.
- LOAD_OPTION indicates how the plugin was loaded. The value is OFF, ON, FORCE, or FORCE_PLUS_PERMANENT. See Section 5.1.8.1, “Installing and Uninstalling Plugins”.

For plugins installed with INSTALL PLUGIN, the PLUGIN_NAME and PLUGIN_LIBRARY values are also registered in the mysql.plugin table.

These statements are equivalent:

```
SELECT
  PLUGIN_NAME, PLUGIN_STATUS, PLUGIN_TYPE,
  PLUGIN_LIBRARY, PLUGIN_LICENSE
FROM INFORMATION_SCHEMA.PLUGINS;

SHOW PLUGINS;
```

For information about plugin data structures that form the basis of the information in the PLUGINS table, see Section 22.2, “The MySQL Plugin API”.

Plugin information is also available using the SHOW PLUGINS statement. See Section 13.7.5.24, “SHOW PLUGINS Syntax”.

19.16 The INFORMATION_SCHEMA PROCESSLIST Table

The PROCESSLIST table provides information about which threads are running.

INFORMATION_SCHEMA Name	**SHOW Name**	**Remarks**
ID	Id	MySQL extension
USER	User	MySQL extension
HOST	Host	MySQL extension
DB	db	MySQL extension
COMMAND	Command	MySQL extension
TIME	Time	MySQL extension

INFORMATION_SCHEMA Name	**SHOW Name**	**Remarks**
STATE	State	MySQL extension
INFO	Info	MySQL extension

For an extensive description of the table columns, see Section 13.7.5.28, “SHOW PROCESSLIST Syntax”.

Notes:

- The PROCESSLIST table is a nonstandard table.
- Like the output from the corresponding SHOW statement, the PROCESSLIST table will only show information about your own threads, unless you have the PROCESS privilege, in which case you will see information about other threads, too. As an anonymous user, you cannot see any rows at all.
- If an SQL statement refers to INFORMATION_SCHEMA.PROCESSLIST, then MySQL will populate the entire table once, when statement execution begins, so there is read consistency during the statement. There is no read consistency for a multi-statement transaction, though.
- Process information is also available from the performance_schema.threads table. However, access to threads does not require a mutex and has minimal impact on server performance. INFORMATION_SCHEMA.PROCESSLIST and SHOW PROCESSLIST have negative performance consequences because they require a mutex. threads also shows information about background threads, which INFORMATION_SCHEMA.PROCESSLIST and SHOW PROCESSLIST do not. This means that threads can be used to monitor activity the other thread information sources cannot.

The following statements are equivalent:

```
SELECT * FROM INFORMATION_SCHEMA.PROCESSLIST

SHOW FULL PROCESSLIST
```

19.17 The INFORMATION_SCHEMA PROFILING Table

The PROFILING table provides statement profiling information. Its contents correspond to the information produced by the SHOW PROFILES and SHOW PROFILE statements (see Section 13.7.5.30, “SHOW PROFILES Syntax”). The table is empty unless the profiling session variable is set to 1.

Note

This table is deprecated as of MySQL 5.7.2 and will be removed in a future MySQL release. Use the Performance Schema instead; see Chapter 20, *MySQL Performance Schema*.

INFORMATION_SCHEMA Name	**SHOW Name**	**Remarks**
QUERY_ID	Query_ID	
SEQ		
STATE	Status	
DURATION	Duration	
CPU_USER	CPU_user	
CPU_SYSTEM	CPU_system	
CONTEXT_VOLUNTARY	Context_voluntary	

INFORMATION_SCHEMA Name	**SHOW Name**	**Remarks**
CONTEXT_INVOLUNTARY	Context_involuntary	
BLOCK_OPS_IN	Block_ops_in	
BLOCK_OPS_OUT	Block_ops_out	
MESSAGES_SENT	Messages_sent	
MESSAGES_RECEIVED	Messages_received	
PAGE_FAULTS_MAJOR	Page_faults_major	
PAGE_FAULTS_MINOR	Page_faults_minor	
SWAPS	Swaps	
SOURCE_FUNCTION	Source_function	
SOURCE_FILE	Source_file	
SOURCE_LINE	Source_line	

Notes:

- QUERY_ID is a numeric statement identifier.
- SEQ is a sequence number indicating the display order for rows with the same QUERY_ID value.
- STATE is the profiling state to which the row measurements apply.
- DURATION indicates how long statement execution remained in the given state, in seconds.
- CPU_USER and CPU_SYSTEM indicate user and system CPU use, in seconds.
- CONTEXT_VOLUNTARY and CONTEXT_INVOLUNTARY indicate how many voluntary and involuntary context switches occurred.
- BLOCK_OPS_IN and BLOCK_OPS_OUT indicate the number of block input and output operations.
- MESSAGES_SENT and MESSAGES_RECEIVED indicate the number of communication messages sent and received.
- PAGE_FAULTS_MAJOR and PAGE_FAULTS_MINOR indicate the number of major and minor page faults.
- SWAPS indicates how many swaps occurred.
- SOURCE_FUNCTION, SOURCE_FILE, and SOURCE_LINE provide information indicating where in the source code the profiled state executes.

19.18 The INFORMATION_SCHEMA REFERENTIAL_CONSTRAINTS Table

The REFERENTIAL_CONSTRAINTS table provides information about foreign keys.

INFORMATION_SCHEMA Name	**SHOW Name**	**Remarks**
CONSTRAINT_CATALOG		def
CONSTRAINT_SCHEMA		
CONSTRAINT_NAME		
UNIQUE_CONSTRAINT_CATALOG		def

INFORMATION_SCHEMA Name	**SHOW Name**	**Remarks**
UNIQUE_CONSTRAINT_SCHEMA		
UNIQUE_CONSTRAINT_NAME		
MATCH_OPTION		
UPDATE_RULE		
DELETE_RULE		
TABLE_NAME		
REFERENCED_TABLE_NAME		

Notes:

- TABLE_NAME has the same value as TABLE_NAME in INFORMATION_SCHEMA.TABLE_CONSTRAINTS.
- CONSTRAINT_SCHEMA and CONSTRAINT_NAME identify the foreign key.
- UNIQUE_CONSTRAINT_SCHEMA, UNIQUE_CONSTRAINT_NAME, and REFERENCED_TABLE_NAME identify the referenced key.
- The only valid value at this time for MATCH_OPTION is NONE.
- The possible values for UPDATE_RULE or DELETE_RULE are CASCADE, SET NULL, SET DEFAULT, RESTRICT, NO ACTION.

19.19 The INFORMATION_SCHEMA ROUTINES Table

The ROUTINES table provides information about stored routines (both procedures and functions). The ROUTINES table does not include user-defined functions (UDFs).

The column named “mysql.proc name” indicates the mysql.proc table column that corresponds to the INFORMATION_SCHEMA.ROUTINES table column, if any.

INFORMATION_SCHEMA Name	**mysql.proc Name**	**Remarks**
SPECIFIC_NAME	specific_name	
ROUTINE_CATALOG		def
ROUTINE_SCHEMA	db	
ROUTINE_NAME	name	
ROUTINE_TYPE	type	{PROCEDURE|FUNCTION}
DATA_TYPE		same as for COLUMNS table
CHARACTER_MAXIMUM_LENGTH		same as for COLUMNS table
CHARACTER_OCTET_LENGTH		same as for COLUMNS table
NUMERIC_PRECISION		same as for COLUMNS table
NUMERIC_SCALE		same as for COLUMNS table
DATETIME_PRECISION		same as for COLUMNS table
CHARACTER_SET_NAME		same as for COLUMNS table
COLLATION_NAME		same as for COLUMNS table
DTD_IDENTIFIER		data type descriptor

INFORMATION_SCHEMA Name	**mysql.proc Name**	**Remarks**
ROUTINE_BODY		SQL
ROUTINE_DEFINITION	body_utf8	
EXTERNAL_NAME		NULL
EXTERNAL_LANGUAGE	language	NULL
PARAMETER_STYLE		SQL
IS_DETERMINISTIC	is_deterministic	
SQL_DATA_ACCESS	sql_data_access	
SQL_PATH		NULL
SECURITY_TYPE	security_type	
CREATED	created	
LAST_ALTERED	modified	
SQL_MODE	sql_mode	MySQL extension
ROUTINE_COMMENT	comment	MySQL extension
DEFINER	definer	MySQL extension
CHARACTER_SET_CLIENT		MySQL extension
COLLATION_CONNECTION		MySQL extension
DATABASE_COLLATION		MySQL extension

Notes:

- MySQL calculates EXTERNAL_LANGUAGE thus:
  - If mysql.proc.language='SQL', EXTERNAL_LANGUAGE is NULL
  - Otherwise, EXTERNAL_LANGUAGE is what is in mysql.proc.language. However, we do not have external languages yet, so it is always NULL.
- CREATED: The date and time when the routine was created. This is a TIMESTAMP value.
- LAST_ALTERED: The date and time when the routine was last modified. This is a TIMESTAMP value. If the routine has not been modified since its creation, this column holds the same value as the CREATED column.
- SQL_MODE: The SQL mode in effect when the routine was created or altered, and under which the routine executes. For the permitted values, see Section 5.1.7, “Server SQL Modes”.
- CHARACTER_SET_CLIENT: The session value of the character_set_client system variable when the routine was created.
- COLLATION_CONNECTION: The session value of the collation_connection system variable when the routine was created.
- DATABASE_COLLATION: The collation of the database with which the routine is associated.
- The DATA_TYPE, CHARACTER_MAXIMUM_LENGTH, CHARACTER_OCTET_LENGTH, NUMERIC_PRECISION, NUMERIC_SCALE, DATETIME_PRECISION, CHARACTER_SET_NAME, and COLLATION_NAME columns provide information about the data type for the RETURNS clause of stored functions. If a stored routine is a stored procedure, these columns all are NULL.

- Information about stored function RETURNS data types is also available in the PARAMETERS table. The return value data type row for a function can be identified as the row that has an ORDINAL_POSITION value of 0.

19.20 The INFORMATION_SCHEMA SCHEMATA Table

A schema is a database, so the SCHEMATA table provides information about databases.

INFORMATION_SCHEMA Name	SHOW Name	Remarks
CATALOG_NAME		def
SCHEMA_NAME	Database	
DEFAULT_CHARACTER_SET_NAME		
DEFAULT_COLLATION_NAME		
SQL_PATH		NULL

The following statements are equivalent:

```
SELECT SCHEMA_NAME AS `Database`
  FROM INFORMATION_SCHEMA.SCHEMATA
  [WHERE SCHEMA_NAME LIKE 'wild']

SHOW DATABASES
  [LIKE 'wild']
```

19.21 The INFORMATION_SCHEMA SCHEMA_PRIVILEGES Table

The SCHEMA_PRIVILEGES table provides information about schema (database) privileges. This information comes from the mysql.db grant table.

INFORMATION_SCHEMA Name	SHOW Name	Remarks
GRANTEE		*'user_name'@'host_name'* value, MySQL extension
TABLE_CATALOG		def, MySQL extension
TABLE_SCHEMA		MySQL extension
PRIVILEGE_TYPE		MySQL extension
IS_GRANTABLE		MySQL extension

Notes:

- This is a nonstandard table. It takes its values from the mysql.db table.

19.22 The INFORMATION_SCHEMA STATISTICS Table

The STATISTICS table provides information about table indexes.

INFORMATION_SCHEMA Name	SHOW Name	Remarks
TABLE_CATALOG		def
TABLE_SCHEMA		= Database

INFORMATION_SCHEMA Name	**SHOW Name**	**Remarks**
TABLE_NAME	Table	
NON_UNIQUE	Non_unique	
INDEX_SCHEMA		= Database
INDEX_NAME	Key_name	
SEQ_IN_INDEX	Seq_in_index	
COLUMN_NAME	Column_name	
COLLATION	Collation	
CARDINALITY	Cardinality	
SUB_PART	Sub_part	MySQL extension
PACKED	Packed	MySQL extension
NULLABLE	Null	MySQL extension
INDEX_TYPE	Index_type	MySQL extension
COMMENT	Comment	MySQL extension

Notes:

- There is no standard table for indexes. The preceding list is similar to what SQL Server 2000 returns for sp_statistics, except that we replaced the name QUALIFIER with CATALOG and we replaced the name OWNER with SCHEMA.

  Clearly, the preceding table and the output from SHOW INDEX are derived from the same parent. So the correlation is already close.

The following statements are equivalent:

```
SELECT * FROM INFORMATION_SCHEMA.STATISTICS
  WHERE table_name = 'tbl_name'
  AND table_schema = 'db_name'

SHOW INDEX
  FROM tbl_name
  FROM db_name
```

19.23 The INFORMATION_SCHEMA TABLES Table

The TABLES table provides information about tables in databases.

INFORMATION_SCHEMA Name	**SHOW Name**	**Remarks**
TABLE_CATALOG		def
TABLE_SCHEMA	Table_...	
TABLE_NAME	Table_...	
TABLE_TYPE		
ENGINE	Engine	MySQL extension
VERSION	Version	The version number of the table's .frm file, MySQL extension

INFORMATION_SCHEMA Name	**SHOW Name**	**Remarks**
ROW_FORMAT	Row_format	MySQL extension
TABLE_ROWS	Rows	MySQL extension
AVG_ROW_LENGTH	Avg_row_length	MySQL extension
DATA_LENGTH	Data_length	MySQL extension
MAX_DATA_LENGTH	Max_data_length	MySQL extension
INDEX_LENGTH	Index_length	MySQL extension
DATA_FREE	Data_free	MySQL extension
AUTO_INCREMENT	Auto_increment	MySQL extension
CREATE_TIME	Create_time	MySQL extension
UPDATE_TIME	Update_time	MySQL extension
CHECK_TIME	Check_time	MySQL extension
TABLE_COLLATION	Collation	MySQL extension
CHECKSUM	Checksum	MySQL extension
CREATE_OPTIONS	Create_options	MySQL extension
TABLE_COMMENT	Comment	MySQL extension

Notes:

- TABLE_SCHEMA and TABLE_NAME are a single field in a SHOW display, for example Table_in_db1.
- TABLE_TYPE should be BASE TABLE or VIEW. Currently, the TABLES table does not list TEMPORARY tables.
- For partitioned tables, the ENGINE column shows the name of the storage engine used by all partitions. (Previously, this column showed PARTITION for such tables.)
- The TABLE_ROWS column is NULL if the table is in the INFORMATION_SCHEMA database.

  For InnoDB tables, the row count is only a rough estimate used in SQL optimization. (This is also true if the InnoDB table is partitioned.)
- The DATA_FREE column shows the free space in bytes for InnoDB tables.
- We have nothing for the table's default character set. TABLE_COLLATION is close, because collation names begin with a character set name.
- The CREATE_OPTIONS column shows partitioned if the table is partitioned.
- Beginning with MySQL 5.7.2, UPDATE_TIME displays a timestamp value for the last UPDATE, INSERT, or DELETE performed on InnoDB tables. Previously, UPDATE_TIME displayed a NULL value for InnoDB tables. For MVCC, the timestamp value reflects the COMMIT time, which is considered the last update time. Timestamps are not persisted when the server is restarted or when the table is evicted from the InnoDB data dictionary cache.

The following statements are equivalent:

```
SELECT table_name FROM INFORMATION_SCHEMA.TABLES
  WHERE table_schema = 'db_name'
  [AND table_name LIKE 'wild']
```

```
SHOW TABLES
  FROM db_name
  [LIKE 'wild']
```

19.24 The INFORMATION_SCHEMA TABLESPACES Table

The TABLESPACES table provides information about active tablespaces.

INFORMATION_SCHEMA Name	SHOW Name	Remarks
TABLESPACE_NAME		MySQL extension
ENGINE		MySQL extension
TABLESPACE_TYPE		MySQL extension
LOGFILE_GROUP_NAME		MySQL extension
EXTENT_SIZE		MySQL extension
AUTOEXTEND_SIZE		MySQL extension
MAXIMUM_SIZE		MySQL extension
NODEGROUP_ID		MySQL extension
TABLESPACE_COMMENT		MySQL extension

19.25 The INFORMATION_SCHEMA TABLE_CONSTRAINTS Table

The TABLE_CONSTRAINTS table describes which tables have constraints.

INFORMATION_SCHEMA Name	SHOW Name	Remarks
CONSTRAINT_CATALOG		def
CONSTRAINT_SCHEMA		
CONSTRAINT_NAME		
TABLE_SCHEMA		
TABLE_NAME		
CONSTRAINT_TYPE		

Notes:

- The CONSTRAINT_TYPE value can be UNIQUE, PRIMARY KEY, or FOREIGN KEY.
- The UNIQUE and PRIMARY KEY information is about the same as what you get from the Key_name field in the output from SHOW INDEX when the Non_unique field is 0.
- The CONSTRAINT_TYPE column can contain one of these values: UNIQUE, PRIMARY KEY, FOREIGN KEY, CHECK. This is a CHAR (not ENUM) column. The CHECK value is not available until we support CHECK.

19.26 The INFORMATION_SCHEMA TABLE_PRIVILEGES Table

The TABLE_PRIVILEGES table provides information about table privileges. This information comes from the mysql.tables_priv grant table.

INFORMATION_SCHEMA Name	**SHOW Name**	**Remarks**
GRANTEE		*'user_name'@'host_name'* value
TABLE_CATALOG		def
TABLE_SCHEMA		
TABLE_NAME		
PRIVILEGE_TYPE		
IS_GRANTABLE		

Notes:

- PRIVILEGE_TYPE can contain one (and only one) of these values: SELECT, INSERT, UPDATE, REFERENCES, ALTER, INDEX, DROP, CREATE VIEW.

The following statements are *not* equivalent:

```
SELECT ... FROM INFORMATION_SCHEMA.TABLE_PRIVILEGES

SHOW GRANTS ...
```

19.27 The INFORMATION_SCHEMA TRIGGERS Table

The TRIGGERS table provides information about triggers. You can see information only for databases and tables for which you have the TRIGGER privilege.

INFORMATION_SCHEMA Name	**SHOW Name**	**Remarks**
TRIGGER_CATALOG		def
TRIGGER_SCHEMA		
TRIGGER_NAME	Trigger	
EVENT_MANIPULATION	Event	
EVENT_OBJECT_CATALOG		def
EVENT_OBJECT_SCHEMA		
EVENT_OBJECT_TABLE	Table	
ACTION_ORDER		
ACTION_CONDITION		NULL
ACTION_STATEMENT	Statement	
ACTION_ORIENTATION		ROW
ACTION_TIMING	Timing	
ACTION_REFERENCE_OLD_TABLE		NULL
ACTION_REFERENCE_NEW_TABLE		NULL
ACTION_REFERENCE_OLD_ROW		OLD
ACTION_REFERENCE_NEW_ROW		NEW
CREATED	Created	
SQL_MODE	sql_mode	MySQL extension

INFORMATION_SCHEMA Name	**SHOW Name**	**Remarks**
DEFINER	Definer	MySQL extension
CHARACTER_SET_CLIENT	character_set_client	MySQL extension
COLLATION_CONNECTION	collation_connection	MySQL extension
DATABASE_COLLATION	Database Collation	MySQL extension

Notes:

- The names in the “SHOW Name” column refer to the SHOW TRIGGERS statement, not SHOW CREATE TRIGGER. See Section 13.7.5.37, “SHOW TRIGGERS Syntax”.
- TRIGGER_SCHEMA and TRIGGER_NAME: The name of the database in which the trigger occurs and the trigger name, respectively.
- EVENT_MANIPULATION: The trigger event. This is the type of operation on the associated table for which the trigger activates. The value is 'INSERT' (a row was inserted), 'DELETE' (a row was deleted), or 'UPDATE' (a row was modified).
- EVENT_OBJECT_SCHEMA and EVENT_OBJECT_TABLE: As noted in Section 18.3, “Using Triggers”, every trigger is associated with exactly one table. These columns indicate the database in which this table occurs, and the table name, respectively.
- ACTION_ORDER: The ordinal position of the trigger's action within the list of triggers on the same table with the same EVENT_MANIPULATION and ACTION_TIMING values. Before MySQL 5.7.2, this value is always 0 because it is not possible for a table to have more than one trigger with the same EVENT_MANIPULATION and ACTION_TIMING values.
- ACTION_STATEMENT: The trigger body; that is, the statement executed when the trigger activates. This text uses UTF-8 encoding.
- ACTION_ORIENTATION: Always contains the value 'ROW'.
- ACTION_TIMING: Whether the trigger activates before or after the triggering event. The value is 'BEFORE' or 'AFTER'.
- ACTION_REFERENCE_OLD_ROW and ACTION_REFERENCE_NEW_ROW: The old and new column identifiers, respectively. This means that ACTION_REFERENCE_OLD_ROW always contains the value 'OLD' and ACTION_REFERENCE_NEW_ROW always contains the value 'NEW'.
- CREATED: The date and time when the trigger was created. This is a TIMESTAMP(2) value (with a fractional part in hundredths of seconds) for triggers created in MySQL 5.7.2 or later, NULL for triggers created prior to 5.7.2.
- SQL_MODE: The SQL mode in effect when the trigger was created, and under which the trigger executes. For the permitted values, see Section 5.1.7, “Server SQL Modes”.
- DEFINER: The account of the user who created the trigger, in '*user_name*'@'*host_name*' format.
- CHARACTER_SET_CLIENT: The session value of the character_set_client system variable when the trigger was created.
- COLLATION_CONNECTION: The session value of the collation_connection system variable when the trigger was created.
- DATABASE_COLLATION: The collation of the database with which the trigger is associated.

- The following columns currently always contain NULL: ACTION_CONDITION, ACTION_REFERENCE_OLD_TABLE, and ACTION_REFERENCE_NEW_TABLE.

Example, using the ins_sum trigger defined in Section 18.3, “Using Triggers”:

```
mysql> SELECT * FROM INFORMATION_SCHEMA.TRIGGERS
    -> WHERE TRIGGER_SCHEMA='test' AND TRIGGER_NAME='ins_sum'\G
*************************** 1. row ***************************
           TRIGGER_CATALOG: def
            TRIGGER_SCHEMA: test
              TRIGGER_NAME: ins_sum
        EVENT_MANIPULATION: INSERT
      EVENT_OBJECT_CATALOG: def
       EVENT_OBJECT_SCHEMA: test
        EVENT_OBJECT_TABLE: account
              ACTION_ORDER: 1
          ACTION_CONDITION: NULL
          ACTION_STATEMENT: SET @sum = @sum + NEW.amount
        ACTION_ORIENTATION: ROW
             ACTION_TIMING: BEFORE
ACTION_REFERENCE_OLD_TABLE: NULL
ACTION_REFERENCE_NEW_TABLE: NULL
  ACTION_REFERENCE_OLD_ROW: OLD
  ACTION_REFERENCE_NEW_ROW: NEW
                   CREATED: 2013-07-05 07:41:21.26
                  SQL_MODE: NO_ENGINE_SUBSTITUTION
                   DEFINER: me@localhost
      CHARACTER_SET_CLIENT: utf8
      COLLATION_CONNECTION: utf8_general_ci
        DATABASE_COLLATION: latin1_swedish_ci
```

19.28 The INFORMATION_SCHEMA USER_PRIVILEGES Table

The USER_PRIVILEGES table provides information about global privileges. This information comes from the mysql.user grant table.

INFORMATION_SCHEMA Name	**SHOW Name**	**Remarks**
GRANTEE		'*user_name*'@'*host_name*' value, MySQL extension
TABLE_CATALOG		def, MySQL extension
PRIVILEGE_TYPE		MySQL extension
IS_GRANTABLE		MySQL extension

Notes:

- This is a nonstandard table. It takes its values from the mysql.user table.

19.29 The INFORMATION_SCHEMA VIEWS Table

The VIEWS table provides information about views in databases. You must have the SHOW VIEW privilege to access this table.

INFORMATION_SCHEMA Name	**SHOW Name**	**Remarks**
TABLE_CATALOG		def
TABLE_SCHEMA		

INFORMATION_SCHEMA Name	SHOW Name	Remarks
TABLE_NAME		
VIEW_DEFINITION		
CHECK_OPTION		
IS_UPDATABLE		
DEFINER		
SECURITY_TYPE		
CHARACTER_SET_CLIENT		MySQL extension
COLLATION_CONNECTION		MySQL extension

Notes:

- The VIEW_DEFINITION column has most of what you see in the Create Table field that SHOW CREATE VIEW produces. Skip the words before SELECT and skip the words WITH CHECK OPTION. Suppose that the original statement was:

  ```
  CREATE VIEW v AS
    SELECT s2,s1 FROM t
    WHERE s1 > 5
    ORDER BY s1
    WITH CHECK OPTION;
  ```

  Then the view definition looks like this:

  ```
  SELECT s2,s1 FROM t WHERE s1 > 5 ORDER BY s1
  ```

- The CHECK_OPTION column has a value of NONE, CASCADE, or LOCAL.

- MySQL sets a flag, called the view updatability flag, at CREATE VIEW time. The flag is set to YES (true) if UPDATE and DELETE (and similar operations) are legal for the view. Otherwise, the flag is set to NO (false). The IS_UPDATABLE column in the VIEWS table displays the status of this flag. It means that the server always knows whether a view is updatable. If the view is not updatable, statements such UPDATE, DELETE, and INSERT are illegal and will be rejected. (Note that even if a view is updatable, it might not be possible to insert into it; for details, refer to Section 13.1.16, “CREATE VIEW Syntax”.)

- DEFINER: The account of the user who created the view, in '*user_name*'@'*host_name*' format. SECURITY_TYPE has a value of DEFINER or INVOKER.

- CHARACTER_SET_CLIENT: The session value of the character_set_client system variable when the view was created.

- COLLATION_CONNECTION: The session value of the collation_connection system variable when the view was created.

MySQL lets you use different sql_mode settings to tell the server the type of SQL syntax to support. For example, you might use the ANSI SQL mode to ensure MySQL correctly interprets the standard SQL concatenation operator, the double bar (||), in your queries. If you then create a view that concatenates items, you might worry that changing the sql_mode setting to a value different from ANSI could cause the view to become invalid. But this is not the case. No matter how you write out a view definition, MySQL always stores it the same way, in a canonical form. Here is an example that shows how the server changes a double bar concatenation operator to a CONCAT() function:

```
mysql> SET sql_mode = 'ANSI';
Query OK, 0 rows affected (0.00 sec)

mysql> CREATE VIEW test.v AS SELECT 'a' || 'b' as col1;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT VIEW_DEFINITION FROM INFORMATION_SCHEMA.VIEWS
    -> WHERE TABLE_SCHEMA = 'test' AND TABLE_NAME = 'v';
+----------------------------------+
| VIEW_DEFINITION                  |
+----------------------------------+
| select concat('a','b') AS `col1` |
+----------------------------------+
1 row in set (0.00 sec)
```

The advantage of storing a view definition in canonical form is that changes made later to the value of sql_mode will not affect the results from the view. However an additional consequence is that comments prior to SELECT are stripped from the definition by the server.

19.30 INFORMATION_SCHEMA Tables for InnoDB

The InnoDB tables related to the InnoDB storage engine serve two purposes:

- You can monitor ongoing InnoDB activity, to detect inefficiencies before they turn into issues, or to troubleshoot performance and capacity issues that do occur. As your database becomes bigger and busier, running up against the limits of your hardware capacity, you monitor and tune these aspects to keep the database running smoothly. The monitoring information deals with:
    - InnoDB table compression, a feature whose use depends on a balance between I/O reduction, CPU usage, buffer pool management, and how much compression is possible for your data.
    - Transactions and locks, features that balance high performance for a single operation, against the ability to run multiple operations concurrently. (Transactions are the high-level, user-visible aspect of concurrency. Locks are the low-level mechanism that transactions use to avoid reading or writing unreliable data.)
- You can extract information about schema objects managed by InnoDB, using the INNODB_SYS_* tables. This information comes from the InnoDB data dictionary, which cannot be queried directly like regular InnoDB tables. Traditionally, you would get this type of information using the techniques from Section 14.2.12.4, “InnoDB Monitors”, setting up InnoDB monitors and parsing the output from the SHOW ENGINE INNODB STATUS command. The InnoDB interface offers a simpler, familiar technique to access this data.

19.30.1 The INFORMATION_SCHEMA INNODB_CMP and INNODB_CMP_RESET Tables

The INNODB_CMP and INNODB_CMP_RESET tables contain status information on operations related to compressed InnoDB tables.

Table 19.1 Columns of INNODB_CMP and INNODB_CMP_RESET

Column name	Description
PAGE_SIZE	Compressed page size in bytes.
COMPRESS_OPS	Number of times a B-tree page of the size PAGE_SIZE has been compressed. Pages are compressed whenever an empty page is created or the space for the uncompressed modification log runs out.

Column name	Description
COMPRESS_OPS_OK	Number of times a B-tree page of the size PAGE_SIZE has been successfully compressed. This count should never exceed COMPRESS_OPS.
COMPRESS_TIME	Total time in seconds spent in attempts to compress B-tree pages of the size PAGE_SIZE.
UNCOMPRESS_OPS	Number of times a B-tree page of the size PAGE_SIZE has been uncompressed. B-tree pages are uncompressed whenever compression fails or at first access when the uncompressed page does not exist in the buffer pool.
UNCOMPRESS_TIME	Total time in seconds spent in uncompressing B-tree pages of the size PAGE_SIZE.

Notes:

- Use these tables to measure the effectiveness of InnoDB table compression in your database.
- You must have the PROCESS privilege to query this table.
- For usage information, see Section 14.2.7.4, “Monitoring Compression at Runtime” and Using the Compression Information Schema Tables. For general information about InnoDB table compression, see Section 14.2.7, “InnoDB Compressed Tables”.

19.30.2 The INFORMATION_SCHEMA INNODB_CMP_PER_INDEX and INNODB_CMP_PER_INDEX_RESET Tables

The INNODB_CMP_PER_INDEX and INNODB_CMP_PER_INDEX_RESET tables contain status information on operations related to compressed InnoDB tables and indexes, with separate statistics for each combination of database, table, and index, to help you evaluate the performance and usefulness of compression for specific tables.

For a compressed InnoDB table, both the table data and all the secondary indexes are compressed. In this context, the table data is treated as just another index, one that happens to contain all the columns: the clustered index.

Table 19.2 Columns of INNODB_CMP_PER_INDEX and INNODB_CMP_PER_INDEX_RESET

Column name	Description
DATABASE_NAME	Database containing the applicable table.
TABLE_NAME	Table to monitor for compression statistics.
INDEX_NAME	Index to monitor for compression statistics.
COMPRESS_OPS	Number of compression operations attempted. Pages are compressed whenever an empty page is created or the space for the uncompressed modification log runs out.
COMPRESS_OPS_OK	Number of successful compression operations. Subtract from the COMPRESS_OPS value to get the number of compression failures. Divide by the COMPRESS_OPS value to get the percentage of compression failures.
COMPRESS_TIME	Total amount of CPU time, in seconds, used for compressing data in this index.
UNCOMPRESS_OPS	Number of uncompression operations performed. Compressed InnoDB pages are uncompressed whenever compression fails, or the first time a

Column name	Description
	compressed page is accessed in the buffer pool and the uncompressed page does not exist.
UNCOMPRESS_TIME	Total amount of CPU time, in seconds, used for uncompressing data in this index.

Notes:

- Use these tables to measure the effectiveness of InnoDB table compression for specific tables, indexes, or both.
- You must have the PROCESS privilege to query these tables.
- Because collecting separate measurements for every index imposes substantial performance overhead, enable the innodb_cmp_per_index_enabled configuration option before performing the operations on compressed tables that you want to monitor.
- For usage information, see Section 14.2.7.4, “Monitoring Compression at Runtime” and Using the Compression Information Schema Tables. For general information about InnoDB table compression, see Section 14.2.7, “InnoDB Compressed Tables”.

19.30.3 The INFORMATION_SCHEMA INNODB_CMPMEM and INNODB_CMPMEM_RESET Tables

The INNODB_CMPMEM and INNODB_CMPMEM_RESET tables contain status information on compressed pages within the InnoDB buffer pool.

Table 19.3 Columns of INNODB_CMPMEM and INNODB_CMPMEM_RESET

Column name	Description
PAGE_SIZE	Block size in bytes. Each record of this table describes blocks of this size.
BUFFER_POOL_INSTANCE	A unique identifier for the buffer pool instance.
PAGES_USED	Number of blocks of the size PAGE_SIZE that are currently in use.
PAGES_FREE	Number of blocks of the size PAGE_SIZE that are currently available for allocation. This column shows the external fragmentation in the memory pool. Ideally, these numbers should be at most 1.
RELOCATION_OPS	Number of times a block of the size PAGE_SIZE has been relocated. The buddy system can relocate the allocated “buddy neighbor” of a freed block when it tries to form a bigger freed block. Reading from the table INNODB_CMPMEM_RESET resets this count.
RELOCATION_TIME	Total time in microseconds spent in relocating blocks of the size PAGE_SIZE. Reading from the table INNODB_CMPMEM_RESET resets this count.

Notes:

- Use these tables to measure the effectiveness of InnoDB table compression in your database.
- You must have the PROCESS privilege to query this table.
- For usage information, see Section 14.2.7.4, “Monitoring Compression at Runtime” and Using the Compression Information Schema Tables. For general information about InnoDB table compression, see Section 14.2.7, “InnoDB Compressed Tables”.

19.30.4 The INFORMATION_SCHEMA INNODB_TRX Table

The INNODB_TRX table contains information about every transaction currently executing inside InnoDB, including whether the transaction is waiting for a lock, when the transaction started, and the SQL statement the transaction is executing.

Table 19.4 INNODB_TRX Columns

Column name	Description
TRX_ID	Unique transaction ID number, internal to InnoDB. (Starting in MySQL 5.6, these IDs are not created for transactions that are read-only and non-locking. See Optimizations for Read-Only Transactions for details.)
TRX_WEIGHT	The weight of a transaction, reflecting (but not necessarily the exact count of) the number of rows altered and the number of rows locked by the transaction. To resolve a deadlock, InnoDB selects the transaction with the smallest weight as the “victim” to rollback. Transactions that have changed non-transactional tables are considered heavier than others, regardless of the number of altered and locked rows.
TRX_STATE	Transaction execution state. One of RUNNING, LOCK WAIT, ROLLING BACK or COMMITTING.
TRX_STARTED	Transaction start time.
TRX_REQUESTED_LOCK_ID	ID of the lock the transaction is currently waiting for (if TRX_STATE is LOCK WAIT, otherwise NULL). Details about the lock can be found by joining with INNODB_LOCKS on LOCK_ID.
TRX_WAIT_STARTED	Time when the transaction started waiting on the lock (if TRX_STATE is LOCK WAIT, otherwise NULL).
TRX_MYSQL_THREAD_ID	MySQL thread ID. Can be used for joining with PROCESSLIST on ID. See Possible Inconsistency with PROCESSLIST.
TRX_QUERY	The SQL query that is being executed by the transaction.
TRX_OPERATION_STATE	The transaction's current operation, or NULL.
TRX_TABLES_IN_USE	The number of InnoDB tables used while processing the current SQL statement of this transaction.
TRX_TABLES_LOCKED	Number of InnoDB tables that the current SQL statement has row locks on. (Because these are row locks, not table locks, the tables can usually still be read from and written to by multiple transactions, despite some rows being locked.)
TRX_LOCK_STRUCTS	The number of locks reserved by the transaction.
TRX_LOCK_MEMORY_BYTES	Total size taken up by the lock structures of this transaction in memory.
TRX_ROWS_LOCKED	Approximate number or rows locked by this transaction. The value might include delete-marked rows that are physically present but not visible to the transaction.
TRX_ROWS_MODIFIED	The number of modified and inserted rows in this transaction.
TRX_CONCURRENCY_TICKETS	A value indicating how much work the current transaction can do before being swapped out, as specified by the innodb_concurrency_tickets option.

Column name	Description
TRX_ISOLATION_LEVEL	The isolation level of the current transaction.
TRX_UNIQUE_CHECKS	Whether unique checks are turned on or off for the current transaction. (They might be turned off during a bulk data load, for example.)
TRX_FOREIGN_KEY_CHECKS	Whether foreign key checks are turned on or off for the current transaction. (They might be turned off during a bulk data load, for example.)
TRX_LAST_FOREIGN_KEY_ERROR	Detailed error message for last FK error, or NULL.
TRX_ADAPTIVE_HASH_LATCHED	Whether or not the adaptive hash index is locked by the current transaction. (Only a single transaction at a time can modify the adaptive hash index.)
TRX_ADAPTIVE_HASH_TIMEOUT	Whether to relinquish the search latch immediately for the adaptive hash index, or reserve it across calls from MySQL. When there is no AHI contention, this value remains zero and statements reserve the latch until they finish. During times of contention, it counts down to zero, and statements release the latch immediately after each row lookup.
TRX_IS_READ_ONLY	A value of 1 indicates the transaction is read-only. (5.6.4 and up.)
TRX_AUTOCOMMIT_NON_LOCKING	A value of 1 indicates the transaction is a SELECT statement that does not use the FOR UPDATE or LOCK IN SHARED MODE clauses, and is executing with the autocommit setting turned on so that the transaction will only contain this one statement. (5.6.4 and up.) When this column and TRX_IS_READ_ONLY are both 1, InnoDB optimizes the transaction to reduce the overhead associated with transactions that change table data.

Notes:

- Use this table to help diagnose performance problems that occur during times of heavy concurrent load. Its contents are updated as described in Granularity of INFORMATION_SCHEMA Data.
- You must have the PROCESS privilege to query this table.
- For usage information, see Using the Transaction Information Schema Tables.

19.30.5 The INFORMATION_SCHEMA INNODB_LOCKS Table

The INNODB_LOCKS table contains information about each lock that an InnoDB transaction has requested but not yet acquired, and each lock that a transaction holds that is blocking another transaction.

Table 19.5 INNODB_LOCKS Columns

Column name	Description
LOCK_ID	Unique lock ID number, internal to InnoDB. Treat it as an opaque string. Although LOCK_ID currently contains TRX_ID, the format of the data in LOCK_ID is not guaranteed to remain the same in future releases. Do not write programs that parse the LOCK_ID value.
LOCK_TRX_ID	ID of the transaction holding this lock. Details about the transaction can be found by joining with INNODB_TRX on TRX_ID.

Column name	Description
LOCK_MODE	Mode of the lock. One of S, X, IS, IX, S_GAP, X_GAP, IS_GAP, IX_GAP, or AUTO_INC for shared, exclusive, intention shared, intention exclusive row locks, shared and exclusive gap locks, intention shared and intention exclusive gap locks, and auto-increment table level lock, respectively. Refer to the sections Section 14.2.2.3, "InnoDB Lock Modes" and Section 14.2.2.2, "The InnoDB Transaction Model and Locking" for information on InnoDB locking.
LOCK_TYPE	Type of the lock. One of RECORD or TABLE for record (row) level or table level locks, respectively.
LOCK_TABLE	Name of the table that has been locked or contains locked records.
LOCK_INDEX	Name of the index if LOCK_TYPE='RECORD', otherwise NULL.
LOCK_SPACE	Tablespace ID of the locked record if LOCK_TYPE='RECORD', otherwise NULL.
LOCK_PAGE	Page number of the locked record if LOCK_TYPE='RECORD', otherwise NULL.
LOCK_REC	Heap number of the locked record within the page if LOCK_TYPE='RECORD', otherwise NULL.
LOCK_DATA	Primary key of the locked record if LOCK_TYPE='RECORD', otherwise NULL. This column contains the value(s) of the primary key column(s) in the locked row, formatted as a valid SQL string (ready to be copied to SQL commands). If there is no primary key then the InnoDB internal unique row ID number is used. When the page containing the locked record is not in the buffer pool (in the case that it was paged out to disk while the lock was held), InnoDB does not fetch the page from disk, to avoid unnecessary disk operations. Instead, LOCK_DATA is set to NULL.

Notes:

- Use this table to help diagnose performance problems that occur during times of heavy concurrent load. Its contents are updated as described in Granularity of INFORMATION_SCHEMA Data.
- You must have the PROCESS privilege to query this table.
- For usage information, see Using the Transaction Information Schema Tables.

19.30.6 The INFORMATION_SCHEMA INNODB_LOCK_WAITS Table

The INNODB_LOCK_WAITS table contains one or more rows for each blocked InnoDB transaction, indicating the lock it has requested and any locks that are blocking that request.

Table 19.6 INNODB_LOCK_WAITS Columns

Column name	Description
REQUESTING_TRX_ID	ID of the requesting transaction.
REQUESTED_LOCK_ID	ID of the lock for which a transaction is waiting. Details about the lock can be found by joining with INNODB_LOCKS on LOCK_ID.
BLOCKING_TRX_ID	ID of the blocking transaction.
BLOCKING_LOCK_ID	ID of a lock held by a transaction blocking another transaction from proceeding. Details about the lock can be found by joining with INNODB_LOCKS on LOCK_ID.

Notes:

- Use this table to help diagnose performance problems that occur during times of heavy concurrent load. Its contents are updated as described in Granularity of INFORMATION_SCHEMA Data.
- You must have the PROCESS privilege to query this table.
- For usage information, see Using the Transaction Information Schema Tables.

19.30.7 The INFORMATION_SCHEMA INNODB_SYS_TABLES Table

The INNODB_SYS_TABLES table provides status information about InnoDB tables, equivalent to the information from the SYS_TABLES table in the InnoDB data dictionary.

Table 19.7 INNODB_SYS_TABLES Columns

Column name	Description
TABLE_ID	An identifier for each InnoDB table that is unique across all databases in the instance.
NAME	The name of the table. Preceded by the database name where appropriate, for example test/t1. InnoDB system table names are in all uppercase. Names of databases and user tables are in the same case as they were originally defined, possibly influenced by the lower_case_table_names setting.
FLAG	0 = InnoDB system table, 1 = user table.
N_COLS	The number of columns in the table.
SPACE	An identifier for the tablespace where the table resides. 0 means the InnoDB system tablespace. Any other number represents a table created in file-per-table mode with a separate .ibd file. This identifier stays the same after a TRUNCATE TABLE statement. Other than the zero value, this identifier is unique for tables across all the databases in the instance.

Notes:

- You must have the PROCESS privilege to query this table.

19.30.8 The INFORMATION_SCHEMA INNODB_SYS_INDEXES Table

The INNODB_SYS_INDEXES table provides status information about InnoDB indexes, equivalent to the information from the SYS_INDEXES table in the InnoDB data dictionary.

Table 19.8 INNODB_SYS_INDEXES Columns

Column name	Description
INDEX_ID	An identifier for each index that is unique across all the databases in an instance.
NAME	The name of the index. User-created indexes have names in all lowercase. Indexes created implicitly by InnoDB have names in all lowercase. The index names are not necessarily unique. Indexes created implicitly by InnoDB have consistent names: PRIMARY for a primary key index, GEN_CLUST_INDEX for the index representing a primary key when one is not specified, ID_IND, FOR_IND for validating a foreign key constraint, and REF_IND.
TABLE_ID	An identifier representing the table associated with the index; the same value from INNODB_SYS_TABLES.TABLE_ID.

Column name	Description
TYPE	A numeric identifier signifying the kind of index. 0 = Secondary Index, 1 = Clustered Index, 2 = Unique Index, 3 = Primary Index, 32 = Full-text Index.
N_FIELDS	The number of columns in the index key. For the GEN_CLUST_INDEX indexes, this value is 0 because the index is created using an artificial value rather than a real table column.
PAGE_NO	The root page number of the index B-tree. For full-text indexes, the PAGE_NO field is unused and set to -1 (FIL_NULL) because the full-text index is laid out in several B-trees (auxiliary tables).
SPACE	An identifier for the tablespace where the index resides. 0 means the InnoDB system tablespace. Any other number represents a table created in file-per-table mode with a separate .ibd file. This identifier stays the same after a TRUNCATE TABLE statement. Because all indexes for a table reside in the same tablespace as the table, this value is not necessarily unique.

Notes:

- You must have the PROCESS privilege to query this table.

19.30.9 The INFORMATION_SCHEMA INNODB_SYS_COLUMNS Table

The INNODB_SYS_COLUMNS table provides status information about InnoDB table columns, equivalent to the information from the SYS_COLUMNS table in the InnoDB data dictionary.

Table 19.9 INNODB_SYS_COLUMNS Columns

Column name	Description
TABLE_ID	An identifier representing the table associated with the column; the same value from INNODB_SYS_TABLES.TABLE_ID.
NAME	The name of each column in each table. These names can be uppercase or lowercase depending on the lower_case_table_names setting. There are no special system-reserved names for columns.
POS	The ordinal position of the column within the table, starting from 0 and incrementing sequentially. When a column is dropped, the remaining columns are reordered so that the sequence has no gaps.
MTYPE	A numeric identifier for the column type. 1 = VARCHAR, 2 = CHAR, 3 = FIXBINARY, 4 = BINARY, 5 = BLOB, 6 = INT, 7 = SYS_CHILD, 8 = SYS, 9 = FLOAT, 10 = DOUBLE, 11 = DECIMAL, 12 = VARMYSQL, 13 = MYSQL.
PRTYPE	The InnoDB “precise type”, a binary value with bits representing MySQL data type, character set code, and nullability.
LEN	The column length, for example 4 for INT and 8 for BIGINT. For character columns in multi-byte character sets, this length value is the maximum length in bytes needed to represent a definition such as VARCHAR(*N*); that is, it might be 2**N*, 3**N*, and so on depending on the character encoding.

Notes:

- You must have the PROCESS privilege to query this table.

19.30.10 The INFORMATION_SCHEMA INNODB_SYS_FIELDS Table

The INNODB_SYS_FIELDS table provides status information about the key columns (fields) of InnoDB indexes, equivalent to the information from the SYS_FIELDS table in the InnoDB data dictionary.

Table 19.10 INNODB_SYS_FIELDS Columns

Column name	Description
INDEX_ID	An identifier for the index associated with this key field, using the same value as in INNODB_SYS_INDEXES.INDEX_ID.
NAME	The name of the original column from the table, using the same value as in INNODB_SYS_COLUMNS.NAME.
POS	The ordinal position of the key field within the index, starting from 0 and incrementing sequentially. When a column is dropped, the remaining columns are reordered so that the sequence has no gaps.

Notes:

- You must have the PROCESS privilege to query this table.

19.30.11 The INFORMATION_SCHEMA INNODB_SYS_FOREIGN Table

The INNODB_SYS_FOREIGN table provides status information about InnoDB foreign keys, equivalent to the information from the SYS_FOREIGN table in the InnoDB data dictionary.

Table 19.11 INNODB_SYS_FOREIGN Columns

Column name	Description
ID	The name (not a numeric value) of the foreign key index. Preceded by the database name, for example, test/products_fk.
FOR_NAME	The name of the child table in this foreign key relationship.
REF_NAME	The name of the parent table in this foreign key relationship.
N_COLS	The number of columns in the foreign key index.
TYPE	A collection of bit flags with information about the foreign key column, ORed together. 1 = ON DELETE CASCADE, 2 = ON UPDATE SET NULL, 4 = ON UPDATE CASCADE, 8 = ON UPDATE SET NULL, 16 = ON DELETE NO ACTION, 32 = ON UPDATE NO ACTION.

Notes:

- You must have the PROCESS privilege to query this table.

19.30.12 The INFORMATION_SCHEMA INNODB_SYS_FOREIGN_COLS Table

The INNODB_SYS_FOREIGN_COLS table provides status information about the columns of InnoDB foreign keys, equivalent to the information from the SYS_FOREIGN_COLS table in the InnoDB data dictionary.

Table 19.12 INNODB_SYS_FOREIGN_COLS Columns

Column name	Description
ID	The foreign key index associated with this index key field, using the same value as INNODB_SYS_FOREIGN.ID.
FOR_COL_NAME	The name of the associated column in the child table.

Column name	Description
REF_COL_NAME	The name of the associated column in the parent table.
POS	The ordinal position of this key field within the foreign key index, starting from 0.

Notes:

- You must have the PROCESS privilege to query this table.

19.30.13 The INFORMATION_SCHEMA INNODB_SYS_TABLESTATS View

The INNODB_SYS_TABLESTATS view provides status information about performance statistics for InnoDB tables. These statistics represent low-level information used by the MySQL optimizer to calculate which index to use when querying an InnoDB table. This information is derived from in-memory data structures rather than corresponding to data stored on disk.

InnoDB tables are represented in this view if they have been opened since the last server restart, and not aged out of the table cache. Tables for which persistent stats are available are always represented in this view.

Table 19.13 INNODB_SYS_TABLESTATS Columns

Column name	Description
TABLE_ID	An identifier representing the table for which statistics are available, using the same value as INNODB_SYS_TABLES.TABLE_ID.
NAME	The name of the table, using the same value as INNODB_SYS_TABLES.NAME.
STATS_INITIALIZED	The value is Initialized if the statistics are already collected, Uninitialized if not.
NUM_ROWS	The current estimated number of rows in the table. Updated after each DML operation. Could be imprecise if uncommitted transactions are inserting into or deleting from the table.
CLUST_INDEX_SIZE	Number of pages on disk that store the clustered index, which holds the InnoDB table data in primary key order. This value might be null if no statistics are collected yet for the table.
OTHER_INDEX_SIZE	Number of pages on disk that store all secondary indexes for the table. This value might be null if no statistics are collected yet for the table.
MODIFIED_COUNTER	The number of rows modified by DML operations, such as INSERT, UPDATE, DELETE, and also cascade operations from foreign keys. This column is reset each time table statistics are recalculated
AUTOINC	
REF_COUNT	When this counter reaches zero, the table metadata can be evicted from the table cache.

Notes:

- This table is primarily useful for expert-level performance monitoring, or when developing performance-related extensions for MySQL.
- Since the INFORMATION_SCHEMA is a general-purpose way to monitor the MySQL server, use this table rather than the corresponding InnoDB system table for any new monitoring application you develop.

- You must have the PROCESS privilege to query this table.

19.30.14 The INFORMATION_SCHEMA INNODB_SYS_DATAFILES Table

The INNODB_SYS_DATAFILES table stores InnoDB datafile path information, allowing it to be queried through INFORMATION_SCHEMA.

Table 19.14 INNODB_SYS_DATAFILES Columns

Column name	Description
SPACE	The tablespace Space ID.
PATH	The tablespace datafile path (for example, “.\world\innodb\city.ibd”).

Notes:

- You must have the PROCESS privilege to query this table.

19.30.15 The INFORMATION_SCHEMA INNODB_SYS_TABLESPACES Table

The INNODB_SYS_TABLESPACES table stores information about InnoDB tablespaces, allowing it to be queried through INFORMATION_SCHEMA.

Table 19.15 INNODB_SYS_TABLESPACES Columns

Column name	Description
SPACE	Tablespace Space ID.
NAME	The database and table name (for example, world_innodb\city)
FLAG	The table was created with the CREATE TABLE ... DATA DIRECTORY (0 = false, 1 = true)
FILE_FORMAT	The tablespace file format (for example, Antelope or Barracuda). The data in this field is interpreted from the tablespace flags information that resides in the .ibd file. For more information about InnoDB file formats, see Section 14.2.8, “InnoDB File-Format Management”.
ROW_FORMAT	The tablespace row format (for example, Compact or Redundant). The data in this field is interpreted from the tablespace flags information that resides in the .ibd file.
PAGE_SIZE	The tablespace page size. The data in this field is interpreted from the tablespace flags information that resides in the .ibd file.
ZIP_PAGE_SIZE	The tablespace zip page size. The data in this field is interpreted from the tablespace flags information that resides in the .ibd file.

Notes:

- You must have the PROCESS privilege to query this table.
- Because tablespace flags are always zero for all Antelope file formats (unlike table flags), there is no way to determine from this flag integer if the tablespace row format is Redundant or Compact. As a result, the possible values for the ROW_FORMAT field are “Compact or Redundant”, “Compressed”, or “Dynamic.”

19.30.16 The INFORMATION_SCHEMA INNODB_BUFFER_PAGE Table

The INNODB_BUFFER_PAGE table holds information about each page in the InnoDB buffer pool.

Table 19.16 INNODB_BUFFER_PAGE Columns

Column name	Description
POOL_ID	Buffer Pool ID. An identifier to distinguish between multiple buffer pool instances.
BLOCK_ID	Buffer Pool Block ID.
SPACE	Tablespace ID. Uses the same value as in INNODB_SYS_TABLES.SPACE.
PAGE_NUMBER	Page number.
PAGE_TYPE	Page type string. One of allocated (Freshly allocated page), index (B-tree node), undo_log (Undo log page), inode (Index node), ibuf_free_list (Insert buffer free list), ibuf_bitmap (Insert buffer bitmap), system (System page), trx_system (Transaction system data), file_space_header (File space header), extent_descriptor (Extent descriptor page), blob (Uncompressed BLOB page), compressed_blob (First compressed BLOB page), compressed_blob2 (Subsequent comp BLOB page), unknown (unknown).
FLUSH_TYPE	Flush type.
FIX_COUNT	Number of threads using this block within the buffer pool. When zero, the block is eligible to be evicted.
IS_HASHED	Whether hash index has been built on this page.
NEWEST_MODIFICATION	Log Sequence Number of the youngest modification.
OLDEST_MODIFICATION	Log Sequence Number of the oldest modification.
ACCESS_TIME	An abstract number used to judge the first access time of the page.
TABLE_NAME	Name of the table the page belongs to.
INDEX_NAME	Name of the index the page belongs to. It can be the name of a clustered index or a secondary index.
NUMBER_RECORDS	Number of records within the page.
DATA_SIZE	Sum of the sizes of the records.
COMPRESSED_SIZE	Compressed page size. Null for pages that are not compressed.
PAGE_STATE	Page state. A page with valid data has one of the following states: FILE_PAGE (buffers a page of data from a file), MEMORY (buffers a page from an in-memory object), Other possible states (managed by InnoDB) are: null, READY_FOR_USE, NOT_USED, REMOVE_HASH.
IO_FIX	Specifies whether any I/O is pending for this page: IO_NONE = no pending I/O, IO_READ = read pending, IO_WRITE = write pending.
IS_OLD	bpage->old.
FREE_PAGE_CLOCK	bpage->freed_page_clock.

Notes:

- This table is primarily useful for expert-level performance monitoring, or when developing performance-related extensions for MySQL.

- Since the INFORMATION_SCHEMA is a general-purpose way to monitor the MySQL server, use this table rather than the corresponding InnoDB system table for any new monitoring application you develop.
- You must have the PROCESS privilege to query this table.

19.30.17 The INFORMATION_SCHEMA INNODB_BUFFER_PAGE_LRU Table

The INNODB_BUFFER_PAGE_LRU table holds information about the pages in the InnoDB buffer pool, in particular how they are ordered in the LRU list that determines which pages to evict from the buffer pool when it becomes full.

The definition for this page is the same as for INNODB_BUFFER_PAGE, except this table has an LRU_POSITION column instead of BLOCK_ID.

Notes:

- This table is primarily useful for expert-level performance monitoring, or when developing performance-related extensions for MySQL.

  Since the INFORMATION_SCHEMA is a general-purpose way to monitor the MySQL server, use this table rather than the corresponding InnoDB system table for any new monitoring application you develop.
- Querying this table can require MySQL to allocate a large block of contiguous memory, more than 64 bytes time the number of active pages in the buffer pool. This allocation could potentially cause an out-of-memory error, especially for systems with multi-gigabyte buffer pools.
- Querying this table requires MySQL to lock the data structure representing the buffer pool while traversing the LRU list, which can reduce concurrency, especially for systems with multi-gigabyte buffer pools.

19.30.18 The INFORMATION_SCHEMA INNODB_BUFFER_POOL_STATS Table

The INNODB_BUFFER_POOL_STATS table provides much of the same buffer pool information provided in SHOW ENGINE INNODB STATUS output. Much of the same information may also be obtained using InnoDB buffer pool server status variables.

The idea of making pages in the buffer pool “young” or “not young” refers to transferring them between the sublists at the head and tail of the buffer pool data structure. Pages made “young” take longer to age out of the buffer pool, while pages made “not young” are moved much closer to the point of eviction.

Table 19.17 INNODB_BUFFER_POOL_STATS Columns

Column name	Description
POOL_ID	Buffer Pool ID. A unique identifier to distinguish between multiple buffer pool instances.
POOL_SIZE	The InnoDB buffer pool size in pages.
FREE_BUFFERS	The number of free pages in the InnoDB buffer pool
DATABASE_PAGES	The number of pages in the InnoDB buffer pool containing data. The number includes both dirty and clean pages.
OLD_DATABASE_PAGES	The number of pages in the old buffer pool sublist.
MODIFIED_DATABASE_PAGES	The number of modified (dirty) database pages
PENDING_DECOMPRESS	The number of pages pending decompression

Column name	Description
PENDING_READS	The number of pending reads
PENDING_FLUSH_LRU	The number of pages pending flush in the LRU
PENDING_FLUSH_LIST	The number of pages pending flush in the flush list
PAGES_MADE_YOUNG	The number of pages made young
PAGES_NOT_MADE_YOUNG	The number of pages not made young
PAGES_MADE_YOUNG_RATE	The number of pages made young per second (pages made young since the last printout / time elapsed)
PAGES_MADE_NOT_YOUNG_RATE	The number of pages not made per second (pages not made young since the last printout / time elapsed)
NUMBER_PAGES_READ	The number of pages read
NUMBER_PAGES_CREATED	The number of pages created
NUMBER_PAGES_WRITTEN	The number of pages written
PAGES_READ_RATE	The number of pages read per second (pages read since the last printout / time elapsed)
PAGES_CREATE_RATE	The number of pages created per second (pages created since the last printout / time elapsed)
PAGES_WRITTEN_RATE	The number of pages written per second (pages written since the last printout / time elapsed)
NUMBER_PAGES_GET	The number of logical read requests.
HIT_RATE	The buffer pool hit rate
YOUNG_MAKE_PER_THOUSAND_GETS	The number of pages made young per thousand gets
NOT_YOUNG_MAKE_PER_THOUSAND_GETS	The number of pages not made young per thousand gets
NUMBER_PAGES_READ_AHEAD	The number of pages read ahead
NUMBER_READ_AHEAD_EVICTED	The number of pages read into the InnoDB buffer pool by the read-ahead background thread that were subsequently evicted without having been accessed by queries.
READ_AHEAD_RATE	The read ahead rate per second (pages read ahead since the last printout / time elapsed)
READ_AHEAD_EVICTED_RATE	The number of read ahead pages evicted without access per second (read ahead pages not accessed since the last printout / time elapsed)
LRU_IO_TOTAL	LRU IO total
LRU_IO_CURRENT	LRU IO for the current interval
UNCOMPRESS_TOTAL	Total number of pages decompressed
UNCOMPRESS_CURRENT	The number of pages decompressed in the current interval

Notes:

- This table is primarily useful for expert-level performance monitoring, or when developing performance-related extensions for MySQL.
- Since the INFORMATION_SCHEMA is a general-purpose way to monitor the MySQL server, use this table rather than the corresponding InnoDB system table for any new monitoring application you develop.
- You must have the PROCESS privilege to query this table.

19.30.19 The INFORMATION_SCHEMA INNODB_METRICS Table

This INFORMATION_SCHEMA table presents a wide variety of InnoDB performance information, complementing the specific focus areas of the PERFORMANCE_SCHEMA tables for InnoDB. With simple queries, you can check the overall health of the system. With more detailed queries, you can diagnose issues such as performance bottlenecks, resource shortages, and application issues.

Each monitor represents a point within the InnoDB source code that is instrumented to gather counter information. Each counter can be started, stopped, and reset. You can also perform these actions for a group of counters using their common module name.

Table 19.18 INNODB_METRICS Columns

Column name	Description
NAME	Unique name for the counter.
SUBSYSTEM	The aspect of InnoDB that the metric applies to. See the list following the table for the corresponding module names to use with the SET GLOBAL syntax.
COUNT	Value since the counter is enabled.
MAX_COUNT	Maximum value since the counter is enabled.
MIN_COUNT	Minimum value since the counter is enabled.
AVG_COUNT	Average value since the counter is enabled.
COUNT_RESET	Counter value since it was last reset. (The _RESET fields act like the lap counter on a stopwatch: you can measure the activity during some time interval, while the cumulative figures are still available in the COUNT, MAX_COUNT, and so on fields.)
MAX_COUNT_RESET	Maximum counter value since it was last reset.
MIN_COUNT_RESET	Minimum counter value since it was last reset.
AVG_COUNT_RESET	Average counter value since it was last reset.
TIME_ENABLED	Timestamp of last start.
TIME_DISABLED	Timestamp of last stop.
TIME_ELAPSED	Elapsed time in seconds since the counter started.
TIME_RESET	Timestamp of last stop.
STATUS	Whether the counter is still running () or stopped ().
TYPE	Whether the item is a cumulative counter, or measures the current value of some resource.
COMMENT	Additional description.

Notes:

- You must have the PROCESS privilege to query this table.
- By default, relatively little data is collected. To start, stop, and reset counters, you set one of the configuration options innodb_monitor_enable, innodb_monitor_disable, innodb_monitor_reset, or innodb_monitor_reset_all, using the name of the counter, the name of the module, a wildcard match for such a name using the “%” character, or the special keyword all. For example:

```
# Turn on a counter:
set global innodb_monitor_enable = counter-name;
-- For example:
set global innodb_monitor_enable = trx_undo_slots_used;

# Turn off a counter:
set global innodb_monitor_disable = counter-name;
-- For example:
set global innodb_monitor_disable = trx_undo_slots_used;

# Reset a counter:
set global innodb_monitor_reset = counter_name;
-- For example:
set global innodb_monitor_reset = trx_undo_slots_used;

# Reset all the counter values:
set global innodb_monitor_reset_all = [counter-name|module_name|pattern|all];
-- Examples:
set global innodb_monitor_reset_all = log_writes
set global innodb_monitor_reset_all = module_os;
set global innodb_monitor_reset_all = all;
set global innodb_monitor_reset_all = "trx%";

# Turn on a group of counters:
set global innodb_monitor_enable = module_name;
-- For example:
set global innodb_monitor_enable = module_os;

# Turn off a group of counters:
set global innodb_monitor_disable = module_name;
-- For example:
set global innodb_monitor_disable = module_os;

# Turn on monitor "log_writes":
set global innodb_monitor_enable = log_writes;
-- For example:

# Reset all values for "log_writes":
set global innodb_monitor_reset_all = log_writes;

# Reset the counters for all dml monitors
set global innodb_monitor_reset = "dml_%";

# Turn on all monitors for the transaction subsystem
set global innodb_monitor_enable = module_trx;

# Turn off all monitors
set global innodb_monitor_disable = all;
```

- You can also enable counters and modules using your configuration file. For example, to enable the log module, metadata_table_handles_opened and metadata_table_handles_closed counters, enter the following line in the [mysqld] section of your my.cnf configuration file.

  ```
  [mysqld]
  innodb_monitor_enable = module_recovery,metadata_table_handles_opened,metadata_table_handles_closed
  ```

  When enabling multiple counters or modules in your configuration file, you must specify the configuration option (innodb_monitor_enable) followed by counter and module names separated by a comma, as shown in the example above. Only the innodb_monitor_enable option can be used in your configuration file. The disable and reset configuration options are only supported at the command line.

- The module names correspond to, but are not identical to, the values from the SUBSYSTEM column. Here are the values you can use for *module_name* with the innodb_monitor_enable and related configuration options, along with the corresponding SUBSYSTEM names:

  - module_metadata (subsystem = metadata)
  - module_lock (subsystem = lock)
  - module_buffer (subsystem = buffer)
  - module_buf_page (subsystem = buffer_page_io)
  - module_os (subsystem = os)
  - module_trx (subsystem = transaction)
  - module_purge (subsystem = purge)
  - module_compress (subsystem = compression)
  - module_file (subsystem = file_system)
  - module_index (subsystem = index)
  - module_adaptive_hash (subsystem = adaptive_hash_index)
  - module_ibuf_system (subsystem = change_buffer)
  - module_srv (subsystem = server)
  - module_ddl (subsystem = ddl)
  - module_dml (subsystem = dml)
  - module_log (subsystem = recovery)
  - module_icp (subsystem = icp)

- Because each counter imposes some degree of runtime overhead on the server, typically you enable more counters on test and development servers during experimentation and benchmarking, and only enable counters on production servers to diagnose known issues or monitor aspects that are likely to be bottlenecks for a particular server and workload.

- Counters that are enabled by default correspond to those used by SHOW ENGINE INNODB STATUS. Counters used by SHOW ENGINE INNODB STATUS are always “on” at a system level but you can disable these counters for the innodb_metrics table, as required. Also, counter status is not persistent. Unless specified otherwise, counters revert to their default enabled or disabled status when the server is restarted.

  The items represented in the innodb_metrics table are subject to change, so for the most up-to-date list, query a running MySQL server. The following list shows items in the innodb_metrics table as of MySQL 5.7.4.

  If you run programs that would be affected by additions or changes to the innodb_metrics table, it is recommended that you review releases notes and query the innodb_metrics table for the new release prior to upgrading.

```
mysql> select name, subsystem, status from information_schema.innodb_metrics order by name;
+------------------------------------------+---------------------+----------+
| name                                     | subsystem           | status   |
+------------------------------------------+---------------------+----------+
| adaptive_hash_pages_added                | adaptive_hash_index | disabled |
| adaptive_hash_pages_removed              | adaptive_hash_index | disabled |
| adaptive_hash_rows_added                 | adaptive_hash_index | disabled |
| adaptive_hash_rows_deleted_no_hash_entry | adaptive_hash_index | disabled |
| adaptive_hash_rows_removed               | adaptive_hash_index | disabled |
| adaptive_hash_rows_updated               | adaptive_hash_index | disabled |
| adaptive_hash_searches                   | adaptive_hash_index | enabled  |
| adaptive_hash_searches_btree             | adaptive_hash_index | disabled |
| buffer_data_reads                        | buffer              | enabled  |
| buffer_data_written                      | buffer              | enabled  |
| buffer_flush_adaptive                    | buffer              | disabled |
| buffer_flush_adaptive_pages              | buffer              | disabled |
| buffer_flush_adaptive_total_pages        | buffer              | disabled |
| buffer_flush_avg_page_rate               | buffer              | disabled |
| buffer_flush_background                  | buffer              | disabled |
| buffer_flush_background_pages            | buffer              | disabled |
| buffer_flush_background_total_pages      | buffer              | disabled |
| buffer_flush_batches                     | buffer              | disabled |
| buffer_flush_batch_num_scan              | buffer              | disabled |
| buffer_flush_batch_pages                 | buffer              | disabled |
| buffer_flush_batch_scanned               | buffer              | disabled |
| buffer_flush_batch_scanned_per_call      | buffer              | disabled |
| buffer_flush_batch_total_pages           | buffer              | disabled |
| buffer_flush_lsn_avg_rate                | buffer              | disabled |
| buffer_flush_neighbor                    | buffer              | disabled |
| buffer_flush_neighbor_pages              | buffer              | disabled |
| buffer_flush_neighbor_total_pages        | buffer              | disabled |
| buffer_flush_n_to_flush_requested        | buffer              | disabled |
| buffer_flush_pct_for_dirty               | buffer              | disabled |
| buffer_flush_pct_for_lsn                 | buffer              | disabled |
| buffer_flush_sync                        | buffer              | disabled |
| buffer_flush_sync_pages                  | buffer              | disabled |
| buffer_flush_sync_total_pages            | buffer              | disabled |
| buffer_flush_sync_waits                  | buffer              | disabled |
| buffer_LRU_batches_evict                 | buffer              | disabled |
| buffer_LRU_batches_flush                 | buffer              | disabled |
| buffer_LRU_batch_evict_pages             | buffer              | disabled |
| buffer_LRU_batch_evict_total_pages       | buffer              | disabled |
| buffer_LRU_batch_flush_pages             | buffer              | disabled |
| buffer_LRU_batch_flush_total_pages       | buffer              | disabled |
| buffer_LRU_batch_num_scan                | buffer              | disabled |
| buffer_LRU_batch_scanned                 | buffer              | disabled |
| buffer_LRU_batch_scanned_per_call        | buffer              | disabled |
| buffer_LRU_get_free_search               | Buffer              | disabled |
| buffer_LRU_search_num_scan               | buffer              | disabled |
| buffer_LRU_search_scanned                | buffer              | disabled |
| buffer_LRU_search_scanned_per_call       | buffer              | disabled |
| buffer_LRU_single_flush_failure_count    | Buffer              | disabled |
| buffer_LRU_single_flush_num_scan         | buffer              | disabled |
| buffer_LRU_single_flush_scanned          | buffer              | disabled |
| buffer_LRU_single_flush_scanned_per_call | buffer              | disabled |
| buffer_LRU_unzip_search_num_scan         | buffer              | disabled |
| buffer_LRU_unzip_search_scanned          | buffer              | disabled |
| buffer_LRU_unzip_search_scanned_per_call | buffer              | disabled |
| buffer_pages_created                     | buffer              | enabled  |
| buffer_pages_read                        | buffer              | enabled  |
| buffer_pages_written                     | buffer              | enabled  |
| buffer_page_read_blob                    | buffer_page_io      | disabled |
| buffer_page_read_fsp_hdr                 | buffer_page_io      | disabled |
| buffer_page_read_ibuf_bitmap             | buffer_page_io      | disabled |
| buffer_page_read_ibuf_free_list          | buffer_page_io      | disabled |
| buffer_page_read_index_ibuf_leaf         | buffer_page_io      | disabled |
| buffer_page_read_index_ibuf_non_leaf     | buffer_page_io      | disabled |
| buffer_page_read_index_inode             | buffer_page_io      | disabled |
| buffer_page_read_index_leaf              | buffer_page_io      | disabled |
| buffer_page_read_index_non_leaf          | buffer_page_io      | disabled |
| buffer_page_read_other                   | buffer_page_io      | disabled |
| buffer_page_read_system_page             | buffer_page_io      | disabled |
| buffer_page_read_trx_system              | buffer_page_io      | disabled |
| buffer_page_read_undo_log                | buffer_page_io      | disabled |
| buffer_page_read_xdes                    | buffer_page_io      | disabled |
| buffer_page_read_zblob                   | buffer_page_io      | disabled |
| buffer_page_read_zblob2                  | buffer_page_io      | disabled |
| buffer_page_written_blob                 | buffer_page_io      | disabled |
| buffer_page_written_fsp_hdr              | buffer_page_io      | disabled |
| buffer_page_written_ibuf_bitmap          | buffer_page_io      | disabled |
| buffer_page_written_ibuf_free_list       | buffer_page_io      | disabled |
| buffer_page_written_index_ibuf_leaf      | buffer_page_io      | disabled |
| buffer_page_written_index_ibuf_non_leaf  | buffer_page_io      | disabled |
| buffer_page_written_index_inode          | buffer_page_io      | disabled |
| buffer_page_written_index_leaf           | buffer_page_io      | disabled |
| buffer_page_written_index_non_leaf       | buffer_page_io      | disabled |
| buffer_page_written_other                | buffer_page_io      | disabled |
| buffer_page_written_system_page          | buffer_page_io      | disabled |
| buffer_page_written_trx_system           | buffer_page_io      | disabled |
| buffer_page_written_undo_log             | buffer_page_io      | disabled |
| buffer_page_written_xdes                 | buffer_page_io      | disabled |
| buffer_page_written_zblob                | buffer_page_io      | disabled |
| buffer_page_written_zblob2               | buffer_page_io      | disabled |
| buffer_pool_bytes_data                   | buffer              | enabled  |
| buffer_pool_bytes_dirty                  | buffer              | enabled  |
| buffer_pool_pages_data                   | buffer              | enabled  |
| buffer_pool_pages_dirty                  | buffer              | enabled  |
| buffer_pool_pages_free                   | buffer              | enabled  |
| buffer_pool_pages_misc                   | buffer              | enabled  |
| buffer_pool_pages_total                  | buffer              | enabled  |
| buffer_pool_reads                        | buffer              | enabled  |
| buffer_pool_read_ahead                   | buffer              | enabled  |
| buffer_pool_read_ahead_evicted           | buffer              | enabled  |
| buffer_pool_read_requests                | buffer              | enabled  |
| buffer_pool_size                         | server              | enabled  |
| buffer_pool_wait_free                    | buffer              | enabled  |
| buffer_pool_write_requests               | buffer              | enabled  |
| compression_pad_decrements               | compression         | disabled |
| compression_pad_increments               | compression         | disabled |
| compress_pages_compressed                | compression         | disabled |
| compress_pages_decompressed              | compression         | disabled |
| ddl_background_drop_indexes              | ddl                 | disabled |
| ddl_background_drop_tables               | ddl                 | disabled |
| ddl_online_create_index                  | ddl                 | disabled |
| ddl_pending_alter_table                  | ddl                 | disabled |
| dml_deletes                              | dml                 | enabled  |
| dml_inserts                              | dml                 | enabled  |
| dml_reads                                | dml                 | disabled |
| dml_updates                              | dml                 | enabled  |
| file_num_open_files                      | file_system         | enabled  |
| ibuf_merges                              | change_buffer       | enabled  |
| ibuf_merges_delete                       | change_buffer       | enabled  |
| ibuf_merges_delete_mark                  | change_buffer       | enabled  |
| ibuf_merges_discard_delete               | change_buffer       | enabled  |
| ibuf_merges_discard_delete_mark          | change_buffer       | enabled  |
| ibuf_merges_discard_insert               | change_buffer       | enabled  |
| ibuf_merges_insert                       | change_buffer       | enabled  |
| ibuf_size                                | change_buffer       | enabled  |
| icp_attempts                             | icp                 | disabled |
| icp_match                                | icp                 | disabled |
| icp_no_match                             | icp                 | disabled |
| icp_out_of_range                         | icp                 | disabled |
| index_page_discards                      | index               | disabled |
| index_page_merge_attempts                | index               | disabled |
| index_page_merge_successful              | index               | disabled |
| index_page_reorg_attempts                | index               | disabled |
| index_page_reorg_successful              | index               | disabled |
| index_page_splits                        | index               | disabled |
| innodb_activity_count                    | server              | enabled  |
| innodb_background_drop_table_usec        | server              | disabled |
| innodb_checkpoint_usec                   | server              | disabled |
| innodb_dblwr_pages_written               | server              | enabled  |
| innodb_dblwr_writes                      | server              | enabled  |
| innodb_dict_lru_usec                     | server              | disabled |
| innodb_ibuf_merge_usec                   | server              | disabled |
| innodb_log_flush_usec                    | server              | disabled |
| innodb_master_active_loops               | server              | disabled |
| innodb_master_idle_loops                 | server              | disabled |
| innodb_master_purge_usec                 | server              | disabled |
| innodb_master_thread_sleeps              | server              | disabled |
| innodb_mem_validate_usec                 | server              | disabled |
| innodb_page_size                         | server              | enabled  |
| innodb_rwlock_sx_os_waits                | server              | enabled  |
| innodb_rwlock_sx_spin_rounds             | server              | enabled  |
| innodb_rwlock_sx_spin_waits              | server              | enabled  |
| innodb_rwlock_s_os_waits                 | server              | enabled  |
| innodb_rwlock_s_spin_rounds              | server              | enabled  |
| innodb_rwlock_s_spin_waits               | server              | enabled  |
| innodb_rwlock_x_os_waits                 | server              | enabled  |
| innodb_rwlock_x_spin_rounds              | server              | enabled  |
| innodb_rwlock_x_spin_waits               | server              | enabled  |
| lock_deadlocks                           | lock                | enabled  |
| lock_rec_locks                           | lock                | disabled |
| lock_rec_lock_created                    | lock                | disabled |
| lock_rec_lock_removed                    | lock                | disabled |
| lock_rec_lock_requests                   | lock                | disabled |
| lock_rec_lock_waits                      | lock                | disabled |
| lock_row_lock_current_waits              | lock                | enabled  |
| lock_row_lock_time                       | lock                | enabled  |
| lock_row_lock_time_avg                   | lock                | enabled  |
| lock_row_lock_time_max                   | lock                | enabled  |
| lock_row_lock_waits                      | lock                | enabled  |
| lock_table_locks                         | lock                | disabled |
| lock_table_lock_created                  | lock                | disabled |
| lock_table_lock_removed                  | lock                | disabled |
| lock_table_lock_waits                    | lock                | disabled |
| lock_timeouts                            | lock                | enabled  |
| log_checkpoints                          | recovery            | disabled |
| log_lsn_buf_pool_oldest                  | recovery            | disabled |
| log_lsn_checkpoint_age                   | recovery            | disabled |
| log_lsn_current                          | recovery            | disabled |
| log_lsn_last_checkpoint                  | recovery            | disabled |
| log_lsn_last_flush                       | recovery            | disabled |
| log_max_modified_age_async               | recovery            | disabled |
| log_max_modified_age_sync                | recovery            | disabled |
| log_num_log_io                           | recovery            | disabled |
| log_padded                               | recovery            | enabled  |
| log_pending_checkpoint_writes            | recovery            | disabled |
| log_pending_log_flushes                  | recovery            | disabled |
| log_waits                                | recovery            | enabled  |
| log_writes                               | recovery            | enabled  |
| log_write_requests                       | recovery            | enabled  |
| metadata_mem_pool_size                   | metadata            | enabled  |
| metadata_table_handles_closed            | metadata            | disabled |
| metadata_table_handles_opened            | metadata            | disabled |
| metadata_table_reference_count           | metadata            | disabled |
| os_data_fsyncs                           | os                  | enabled  |
| os_data_reads                            | os                  | enabled  |
| os_data_writes                           | os                  | enabled  |
| os_log_bytes_written                     | os                  | enabled  |
| os_log_fsyncs                            | os                  | enabled  |
| os_log_pending_fsyncs                    | os                  | enabled  |
| os_log_pending_writes                    | os                  | enabled  |
| os_pending_reads                         | os                  | disabled |
| os_pending_writes                        | os                  | disabled |
| purge_del_mark_records                   | purge               | disabled |
| purge_dml_delay_usec                     | purge               | disabled |
| purge_invoked                            | purge               | disabled |
| purge_resume_count                       | purge               | disabled |
| purge_stop_count                         | purge               | disabled |
| purge_undo_log_pages                     | purge               | disabled |
| purge_upd_exist_or_extern_records        | purge               | disabled |
| trx_active_transactions                  | transaction         | disabled |
| trx_commits_insert_update                | transaction         | disabled |
| trx_nl_ro_commits                        | transaction         | disabled |
| trx_rollbacks                            | transaction         | disabled |
| trx_rollbacks_savepoint                  | transaction         | disabled |
| trx_rollback_active                      | transaction         | disabled |
| trx_ro_commits                           | transaction         | disabled |
| trx_rseg_current_size                    | transaction         | disabled |
| trx_rseg_history_len                     | transaction         | enabled  |
| trx_rw_commits                           | transaction         | disabled |
| trx_undo_slots_cached                    | transaction         | disabled |
| trx_undo_slots_used                      | transaction         | disabled |
+------------------------------------------+---------------------+----------+
220 rows in set (0.02 sec)
```

19.30.20 The INFORMATION_SCHEMA INNODB_FT_CONFIG Table

The INNODB_FT_CONFIG table displays metadata about the FULLTEXT index and associated processing for an InnoDB table.

This table is only accessible to users with the SUPER privilege. Before you query this table, set the configuration variable innodb_ft_aux_table to the name (including the database name) of the table that contains the FULLTEXT index, for example test/articles.

Table 19.19 INNODB_FT_CONFIG Columns

Column name	Description
KEY	The name designating an item of metadata for an InnoDB table containing a FULLTEXT index.
VALUE	The value associated with the corresponding KEY column, reflecting some limit or current value for an aspect of a FULLTEXT index for an InnoDB table.

Notes:

- This table is only intended for internal configuration. It is not intended for statistical information purposes.
- You must have the PROCESS privilege to query this table.
- The values for the KEY column might evolve depending on the needs for performance tuning and debugging for InnoDB full-text processing. Currently, the key values include optimize_checkpoint_limit, synced_doc_id, deleted_doc_count, stopword_table_name, and use_stopword.
- For more information, see Section 12.9, “Full-Text Search Functions”.

19.30.21 The INFORMATION_SCHEMA INNODB_FT_DEFAULT_STOPWORD Table

The INNODB_FT_DEFAULT_STOPWORD table holds a list of stopwords that are used by default when creating a FULLTEXT index on an InnoDB table.

Table 19.20 INNODB_FT_DEFAULT_STOPWORD Columns

Column name	Description
value	A word that is used by default as a stopword for FULLTEXT indexes on InnoDB tables. Not used if you override the default stopword processing with either the innodb_ft_server_stopword_table or the innodb_ft_user_stopword_table option.

Notes:

- You must have the PROCESS privilege to query this table.
- For more information, see Section 12.9, “Full-Text Search Functions”.

19.30.22 The INFORMATION_SCHEMA INNODB_FT_INDEX_TABLE Table

The INNODB_FT_INDEX_TABLE table displays information about the inverted index used to process text searches against the FULLTEXT index of an InnoDB table.

This table is only accessible to users with the SUPER privilege. Before you query this table, set the configuration variable innodb_ft_aux_table to the name (including the database name) of the table that contains the FULLTEXT index, for example test/articles.

Table 19.21 INNODB_FT_INDEX_TABLE Columns

Column name	Description
WORD	A word extracted from the text of the columns that are part of a FULLTEXT.
FIRST_DOC_ID	The first document ID that this word appears in in the FULLTEXT index.
LAST_DOC_ID	The last document ID that this word appears in in the FULLTEXT index.
DOC_COUNT	The number of rows this word appears in in the FULLTEXT index. The same word can occur several times within the cache table, once for each combination of DOC_ID and POSITION values.
DOC_ID	The document ID of the row containing the word. This value might reflect the value of an ID column that you defined for the underlying table, or it can be a sequence value generated by InnoDB when the table does not contain a suitable column.
POSITION	The position of this particular instance of the word within the relevant document identified by the DOC_ID value.

Notes:

- This table initially appears empty, until you set the value of the configuration variable innodb_ft_aux_table.
- You must have the PROCESS privilege to query this table.
- For more information, see Section 12.9, “Full-Text Search Functions”.

19.30.23 The INFORMATION_SCHEMA INNODB_FT_INDEX_CACHE Table

The INNODB_FT_INDEX_CACHE table displays token information about newly inserted rows in a FULLTEXT index for an InnoDB table. To avoid expensive index reorganization during DML operations for an InnoDB FULLTEXT index, the information about newly indexed words is stored separately, and combined with the main search index only when you issue the OPTIMIZE TABLE statement for the InnoDB table.

This table is only accessible to users with the SUPER privilege. Before you query this table, set the configuration variable innodb_ft_aux_table to the name (including the database name) of the table that contains the FULLTEXT index, for example test/articles.

Table 19.22 INNODB_FT_INDEX_CACHE Columns

Column name	Description
WORD	A word extracted from the text of a newly inserted row.
FIRST_DOC_ID	The first document ID that this word appears in in the FULLTEXT index.
LAST_DOC_ID	The last document ID that this word appears in in the FULLTEXT index.
DOC_COUNT	The number of rows this word appears in in the FULLTEXT index. The same word can occur several times within the cache table, once for each combination of DOC_ID and POSITION values.
DOC_ID	The document ID of the newly inserted row. This value might reflect the value of an ID column that you defined for the underlying table, or it can be a sequence value generated by InnoDB when the table does not contain a suitable column.
POSITION	The position of this particular instance of the word within the relevant document identified by the DOC_ID value. The value does not represent an absolute position; it is an offset added to the POSITION of the previous instance of that word.

Notes:

- This table initially appears empty, until you set the value of the configuration variable innodb_ft_aux_table.
- You must have the PROCESS privilege to query this table.
- For more information, see Section 12.9, “Full-Text Search Functions”.

19.30.24 The INFORMATION_SCHEMA INNODB_FT_DELETED Table

The INNODB_FT_DELETED table records rows that are deleted from the FULLTEXT index for an InnoDB table. To avoid expensive index reorganization during DML operations for an InnoDB FULLTEXT index, the information about newly deleted words is stored separately, filtered out of search results when you do a text search, and removed from the main search index only when you issue the OPTIMIZE TABLE statement for the InnoDB table.

This table is only accessible to users with the SUPER privilege. Before you query this table, set the configuration variable innodb_ft_aux_table to the name (including the database name) of the table that contains the FULLTEXT index, for example test/articles.

Table 19.23 INNODB_FT_DELETED Columns

Column name	Description
DOC_ID	The document ID of the newly deleted row. This value might reflect the value of an ID column that you defined for the underlying table, or it can be a sequence value generated by InnoDB when the table does not contain a suitable column. This value is used to skip rows in the innodb_ft_index_table table, when you do text searches before data

Column name	Description
	for deleted rows is physically removed from the FULLTEXT index by an OPTIMIZE TABLE statement.

Notes:

- This table initially appears empty, until you set the value of the configuration variable innodb_ft_aux_table.
- You must have the PROCESS privilege to query this table.
- For more information, see Section 12.9, “Full-Text Search Functions”.

19.30.25 The INFORMATION_SCHEMA INNODB_FT_BEING_DELETED Table

The INNODB_FT_BEING_DELETED table is a temporary work table while document IDs in the INNODB_FT_DELETED table are being removed from an InnoDB FULLTEXT index during an OPTIMIZE TABLE operation. Because its contents typically have a short lifetime, this table has limited utility for monitoring or debugging.

This table is only accessible to users with the SUPER privilege. This table initially appears empty, until you set the value of the configuration variable innodb_ft_aux_table.

Table 19.24 INNODB_FT_BEING_DELETED Columns

Column name	Description
DOC_ID	The document ID of the row that is in the process of being deleted. This value might reflect the value of an ID column that you defined for the underlying table, or it can be a sequence value generated by InnoDB when the table does not contain a suitable column. This value is used to skip rows in the innodb_ft_index_table table, when you do text searches before data for deleted rows is physically removed from the FULLTEXT index by an OPTIMIZE TABLE statement.

Notes:

- This table initially appears empty, until you set the value of the configuration variable innodb_ft_aux_table.
- You must have the PROCESS privilege to query this table.
- For more information, see Section 12.9, “Full-Text Search Functions”.

19.30.26 The INFORMATION_SCHEMA INNODB_TEMP_TABLE_INFO Table

The INNODB_TEMP_TABLE_INFO contains metadata about active temporary tables and is created when the first select statement is run against it. The table reports on all user and system-created temporary tables that are active within a given InnoDB instance. This table is not persisted to disk.

As of MySQL 5.7.1, InnoDB temporary table metadata is no longer stored to InnoDB system tables. Instead, the new INNODB_TEMP_TABLE_INFO provides users with a snapshot of active temporary tables.

Table 19.25 INNODB_TEMP_TABLE_INFO Columns

Column name	Description
TABLE_ID	The table ID of the active temporary table.

Column name	Description
NAME	The name of the active temporary table.
N_COLS	The number of columns in the temporary table.
SPACE	The tablespace identifier where the temporary table resides. As of MySQL 5.7.1, all non-compressed InnoDB temporary tables reside in a shared temporary table tablespace. Compressed temporary tables reside in separate dedicated tablespaces. The temporary table tablespace space-id (also know as tablespace identifier) is always non-zero, and because it is dynamically generated, it can vary on server restart.
PER_TABLE_SPACE	Whether this table resides in the shared temporary tablespace or in a dedicated single tablespace. A value of true indicates a dedicated single tablespace. A value of false indicates that the temporary table resides in the shared temporary tablespace.
IS_COMPRESSED	Whether this temporary table is compressed.

Notes:

- This table is primarily useful for expert level monitoring.
- Since the INFORMATION_SCHEMA is a general-purpose way to monitor the MySQL server, use this table rather than the corresponding InnoDB system table for any new monitoring application you develop.
- You must have the PROCESS privilege to query this table.

19.31 Extensions to SHOW Statements

Some extensions to SHOW statements accompany the implementation of INFORMATION_SCHEMA:

- SHOW can be used to get information about the structure of INFORMATION_SCHEMA itself.
- Several SHOW statements accept a WHERE clause that provides more flexibility in specifying which rows to display.

INFORMATION_SCHEMA is an information database, so its name is included in the output from SHOW DATABASES. Similarly, SHOW TABLES can be used with INFORMATION_SCHEMA to obtain a list of its tables:

```
mysql> SHOW TABLES FROM INFORMATION_SCHEMA;
+---------------------------------------+
| Tables_in_INFORMATION_SCHEMA          |
+---------------------------------------+
| CHARACTER_SETS                        |
| COLLATIONS                            |
| COLLATION_CHARACTER_SET_APPLICABILITY |
| COLUMNS                               |
| COLUMN_PRIVILEGES                     |
| ENGINES                               |
| EVENTS                                |
| FILES                                 |
| GLOBAL_STATUS                         |
| GLOBAL_VARIABLES                      |
| KEY_COLUMN_USAGE                      |
| PARTITIONS                            |
| PLUGINS                               |
| PROCESSLIST                           |
| REFERENTIAL_CONSTRAINTS               |
| ROUTINES                              |
| SCHEMATA                              |
| SCHEMA_PRIVILEGES                     |
| SESSION_STATUS                        |
| SESSION_VARIABLES                     |
| STATISTICS                            |
| TABLES                                |
| TABLE_CONSTRAINTS                     |
| TABLE_PRIVILEGES                      |
| TRIGGERS                              |
| USER_PRIVILEGES                       |
| VIEWS                                 |
+---------------------------------------+
27 rows in set (0.00 sec)
```

SHOW COLUMNS and DESCRIBE can display information about the columns in individual INFORMATION_SCHEMA tables.

SHOW statements that accept a LIKE clause to limit the rows displayed also permit a WHERE clause that specifies more general conditions that selected rows must satisfy:

```
SHOW CHARACTER SET
SHOW COLLATION
SHOW COLUMNS
SHOW DATABASES
SHOW FUNCTION STATUS
SHOW INDEX
SHOW OPEN TABLES
SHOW PROCEDURE STATUS
SHOW STATUS
SHOW TABLE STATUS
SHOW TABLES
SHOW TRIGGERS
SHOW VARIABLES
```

The WHERE clause, if present, is evaluated against the column names displayed by the SHOW statement. For example, the SHOW CHARACTER SET statement produces these output columns:

```
mysql> SHOW CHARACTER SET;
+----------+-----------------------------+---------------------+--------+
| Charset  | Description                 | Default collation   | Maxlen |
+----------+-----------------------------+---------------------+--------+
| big5     | Big5 Traditional Chinese    | big5_chinese_ci     |      2 |
| dec8     | DEC West European           | dec8_swedish_ci     |      1 |
| cp850    | DOS West European           | cp850_general_ci    |      1 |
| hp8      | HP West European            | hp8_english_ci      |      1 |
| koi8r    | KOI8-R Relcom Russian       | koi8r_general_ci    |      1 |
| latin1   | cp1252 West European        | latin1_swedish_ci   |      1 |
| latin2   | ISO 8859-2 Central European | latin2_general_ci   |      1 |
...
```

To use a WHERE clause with SHOW CHARACTER SET, you would refer to those column names. As an example, the following statement displays information about character sets for which the default collation contains the string 'japanese':

```
mysql> SHOW CHARACTER SET WHERE `Default collation` LIKE '%japanese%';
+---------+---------------------------+---------------------+--------+
| Charset | Description               | Default collation   | Maxlen |
+---------+---------------------------+---------------------+--------+
| ujis    | EUC-JP Japanese           | ujis_japanese_ci    |      3 |
| sjis    | Shift-JIS Japanese        | sjis_japanese_ci    |      2 |
| cp932   | SJIS for Windows Japanese | cp932_japanese_ci   |      2 |
| eucjpms | UJIS for Windows Japanese | eucjpms_japanese_ci |      3 |
+---------+---------------------------+---------------------+--------+
```

This statement displays the multi-byte character sets:

```
mysql> SHOW CHARACTER SET WHERE Maxlen > 1;
+---------+---------------------------+---------------------+--------+
| Charset | Description               | Default collation   | Maxlen |
+---------+---------------------------+---------------------+--------+
| big5    | Big5 Traditional Chinese  | big5_chinese_ci     |      2 |
| ujis    | EUC-JP Japanese           | ujis_japanese_ci    |      3 |
| sjis    | Shift-JIS Japanese        | sjis_japanese_ci    |      2 |
| euckr   | EUC-KR Korean             | euckr_korean_ci     |      2 |
| gb2312  | GB2312 Simplified Chinese | gb2312_chinese_ci   |      2 |
| gbk     | GBK Simplified Chinese    | gbk_chinese_ci      |      2 |
| utf8    | UTF-8 Unicode             | utf8_general_ci     |      3 |
| ucs2    | UCS-2 Unicode             | ucs2_general_ci     |      2 |
| cp932   | SJIS for Windows Japanese | cp932_japanese_ci   |      2 |
| eucjpms | UJIS for Windows Japanese | eucjpms_japanese_ci |      3 |
+---------+---------------------------+---------------------+--------+
```

Chapter 20 MySQL Performance Schema

Table of Contents

The MySQL Performance Schema is a feature for monitoring MySQL Server execution at a low level. The Performance Schema has these characteristics:

- The Performance Schema provides a way to inspect internal execution of the server at runtime. It is implemented using the `PERFORMANCE_SCHEMA` storage engine and the `performance_schema` database. The Performance Schema focuses primarily on performance data. This differs from `INFORMATION_SCHEMA`, which serves for inspection of metadata.

- The Performance Schema monitors server events. An “event” is anything the server does that takes time and has been instrumented so that timing information can be collected. In general, an event could be a function call, a wait for the operating system, a stage of an SQL statement execution such as parsing or sorting, or an entire statement or group of statements. Currently, event collection provides access to information about synchronization calls (such as for mutexes) file and table I/O, table locks, and so forth for the server and for several storage engines.

- Performance Schema events are distinct from events written to the server's binary log (which describe data modifications) and Event Scheduler events (which are a type of stored program).
- Performance Schema events are specific to a given instance of the MySQL Server. Performance Schema tables are considered local to the server, and changes to them are not replicated or written to the binary log.
- Current events are available, as well as event histories and summaries. This enables you to determine how many times instrumented activities were performed and how much time they took. Event information is available to show the activities of specific threads, or activity associated with particular objects such as a mutex or file.
- The `PERFORMANCE_SCHEMA` storage engine collects event data using “instrumentation points” in server source code.
- Collected events are stored in tables in the `performance_schema` database. These tables can be queried using `SELECT` statements like other tables.
- Performance Schema configuration can be modified dynamically by updating tables in the `performance_schema` database through SQL statements. Configuration changes affect data collection immediately.
- Tables in the `performance_schema` database are views or temporary tables that use no persistent on-disk storage.
- Monitoring is available on all platforms supported by MySQL.

  Some limitations might apply: The types of timers might vary per platform. Instruments that apply to storage engines might not be implemented for all storage engines. Instrumentation of each third-party engine is the responsibility of the engine maintainer. See also Section E.8, “Restrictions on Performance Schema”.
- Data collection is implemented by modifying the server source code to add instrumentation. There are no separate threads associated with the Performance Schema, unlike other features such as replication or the Event Scheduler.

The Performance Schema is intended to provide access to useful information about server execution while having minimal impact on server performance. The implementation follows these design goals:

- Activating the Performance Schema causes no changes in server behavior. For example, it does not cause thread scheduling to change, and it does not cause query execution plans (as shown by `EXPLAIN`) to change.
- No memory allocation is done beyond that which occurs during server startup. By using early allocation of structures with a fixed size, it is never necessary to resize or reallocate them, which is critical for achieving good runtime performance.
- Server monitoring occurs continuously and unobtrusively with very little overhead. Activating the Performance Schema does not make the server unusable.
- The parser is unchanged. There are no new keywords or statements.
- Execution of server code proceeds normally even if the Performance Schema fails internally.
- When there is a choice between performing processing during event collection initially or during event retrieval later, priority is given to making collection faster. This is because collection is ongoing whereas retrieval is on demand and might never happen at all.
- It is easy to add new instrumentation points.

- Instrumentation is versioned. If the instrumentation implementation changes, previously instrumented code will continue to work. This benefits developers of third-party plugins because it is not necessary to upgrade each plugin to stay synchronized with the latest Performance Schema changes.

20.1 Performance Schema Quick Start

This section briefly introduces the Performance Schema with examples that show how to use it. For additional examples, see Section 20.15, “Using the Performance Schema to Diagnose Problems”.

For the Performance Schema to be available, support for it must have been configured when MySQL was built. You can verify whether this is the case by checking the server's help output. If the Performance Schema is available, the output will mention several variables with names that begin with `performance_schema`:

```
shell> mysqld --verbose --help
...
  --performance_schema
                      Enable the performance schema.
  --performance_schema_events_waits_history_long_size=#
                      Number of rows in events_waits_history_long.
...
```

If such variables do not appear in the output, your server has not been built to support the Performance Schema. In this case, see Section 20.2, “Performance Schema Configuration”.

Assuming that the Performance Schema is available, it is enabled by default. To enable or disable it explicitly, start the server with the `performance_schema` variable set to an appropriate value. For example, use these lines in your `my.cnf` file:

```
[mysqld]
performance_schema=on
```

When the server starts, it sees `performance_schema` and attempts to initialize the Performance Schema. To verify successful initialization, use this statement:

```
mysql> SHOW VARIABLES LIKE 'performance_schema';
+--------------------+-------+
| Variable_name      | Value |
+--------------------+-------+
| performance_schema | ON    |
+--------------------+-------+
```

A value of `ON` means that the Performance Schema initialized successfully and is ready for use. A value of `OFF` means that some error occurred. Check the server error log for information about what went wrong.

The Performance Schema is implemented as a storage engine. If this engine is available (which you should already have checked earlier), you should see it listed with a `SUPPORT` value of `YES` in the output from the `INFORMATION_SCHEMA.ENGINES` table or the `SHOW ENGINES` statement:

```
mysql> SELECT * FROM INFORMATION_SCHEMA.ENGINES
    -> WHERE ENGINE='PERFORMANCE_SCHEMA'\G
*************************** 1. row ***************************
      ENGINE: PERFORMANCE_SCHEMA
     SUPPORT: YES
     COMMENT: Performance Schema
TRANSACTIONS: NO
          XA: NO
  SAVEPOINTS: NO
```

```
mysql> SHOW ENGINES\G
...
      Engine: PERFORMANCE_SCHEMA
     Support: YES
     Comment: Performance Schema
Transactions: NO
          XA: NO
  Savepoints: NO
...
```

The PERFORMANCE_SCHEMA storage engine operates on tables in the performance_schema database. You can make performance_schema the default database so that references to its tables need not be qualified with the database name:

```
mysql> USE performance_schema;
```

Many examples in this chapter assume that performance_schema is the default database.

Performance Schema tables are stored in the performance_schema database. Information about the structure of this database and its tables can be obtained, as for any other database, by selecting from the INFORMATION_SCHEMA database or by using SHOW statements. For example, use either of these statements to see what Performance Schema tables exist:

```
mysql> SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES
    -> WHERE TABLE_SCHEMA = 'performance_schema';
+------------------------------------------------------+
| TABLE_NAME                                           |
+------------------------------------------------------+
| accounts                                             |
| cond_instances                                       |
| events_stages_current                                |
| events_stages_history                                |
| events_stages_history_long                           |
| events_stages_summary_by_account_by_event_name       |
| events_stages_summary_by_host_by_event_name          |
| events_stages_summary_by_thread_by_event_name        |
| events_stages_summary_by_user_by_event_name          |
| events_stages_summary_global_by_event_name           |
| events_statements_current                            |
| events_statements_history                            |
| events_statements_history_long                       |
...
| file_instances                                       |
| file_summary_by_event_name                           |
| file_summary_by_instance                             |
| host_cache                                           |
| hosts                                                |
| memory_summary_by_account_by_event_name              |
| memory_summary_by_host_by_event_name                 |
| memory_summary_by_thread_by_event_name               |
| memory_summary_by_user_by_event_name                 |
| memory_summary_global_by_event_name                  |
| metadata_locks                                       |
| mutex_instances                                      |
| objects_summary_global_by_type                       |
| performance_timers                                   |
| replication_connection_configuration                 |
| replication_connection_status                        |
| replication_execute_configuration                    |
| replication_execute_status                           |
| replication_execute_status_by_coordinator            |
| replication_execute_status_by_worker                 |
| rwlock_instances                                     |
| session_account_connect_attrs                        |
| session_connect_attrs                                |
| setup_actors                                         |
| setup_consumers                                      |
| setup_instruments                                    |
| setup_objects                                        |
| setup_timers                                         |
| socket_instances                                     |
| socket_summary_by_event_name                         |
| socket_summary_by_instance                           |
| table_handles                                        |
| table_io_waits_summary_by_index_usage                |
| table_io_waits_summary_by_table                      |
| table_lock_waits_summary_by_table                    |
| threads                                              |
| users                                                |
+------------------------------------------------------+

mysql> SHOW TABLES FROM performance_schema;
+------------------------------------------------------+
| Tables_in_performance_schema                         |
+------------------------------------------------------+
| accounts                                             |
| cond_instances                                       |
| events_stages_current                                |
| events_stages_history                                |
| events_stages_history_long                           |
...
```

The number of Performance Schema tables is expected to increase over time as implementation of additional instrumentation proceeds.

The name of the performance_schema database is lowercase, as are the names of tables within it. Queries should specify the names in lowercase.

To see the structure of individual tables, use SHOW CREATE TABLE:

```
mysql> SHOW CREATE TABLE setup_timers\G
*************************** 1. row ***************************
       Table: setup_timers
Create Table: CREATE TABLE `setup_timers` (
  `NAME` varchar(64) NOT NULL,
  `TIMER_NAME` enum('CYCLE','NANOSECOND','MICROSECOND','MILLISECOND','TICK')
   NOT NULL
) ENGINE=PERFORMANCE_SCHEMA DEFAULT CHARSET=utf8
```

Table structure is also available by selecting from tables such as INFORMATION_SCHEMA.COLUMNS or by using statements such as SHOW COLUMNS.

Tables in the performance_schema database can be grouped according to the type of information in them: Current events, event histories and summaries, object instances, and setup (configuration) information. The following examples illustrate a few uses for these tables. For detailed information about the tables in each group, see Section 20.9, “Performance Schema Table Descriptions”.

Initially, not all instruments and consumers are enabled, so the performance schema does not collect all events. To turn all of these on and enable event timing, execute two statements (the row counts may differ depending on MySQL version):

```
mysql> UPDATE setup_instruments SET ENABLED = 'YES', TIMED = 'YES';
Query OK, 560 rows affected (0.04 sec)
mysql> UPDATE setup_consumers SET ENABLED = 'YES';
Query OK, 10 rows affected (0.00 sec)
```

To see what the server is doing at the moment, examine the events_waits_current table. It contains one row per thread showing each thread's most recent monitored event:

```
mysql> SELECT * FROM events_waits_current\G
*************************** 1. row ***************************
            THREAD_ID: 0
             EVENT_ID: 5523
           EVENT_NAME: wait/synch/mutex/mysys/THR_LOCK::mutex
               SOURCE: thr_lock.c:525
          TIMER_START: 201660494489586
            TIMER_END: 201660494576112
           TIMER_WAIT: 86526
                SPINS: NULL
        OBJECT_SCHEMA: NULL
          OBJECT_NAME: NULL
          OBJECT_TYPE: NULL
OBJECT_INSTANCE_BEGIN: 142270668
     NESTING_EVENT_ID: NULL
            OPERATION: lock
      NUMBER_OF_BYTES: NULL
                FLAGS: 0
...
```

This event indicates that thread 0 was waiting for 86,526 picoseconds to acquire a lock on THR_LOCK::mutex, a mutex in the mysys subsystem. The first few columns provide the following information:

- The ID columns indicate which thread the event comes from and the event number.
- EVENT_NAME indicates what was instrumented and SOURCE indicates which source file contains the instrumented code.
- The timer columns show when the event started and stopped and how long it took. If an event is still in progress, the TIMER_END and TIMER_WAIT values are NULL. Timer values are approximate and expressed in picoseconds. For information about timers and event time collection, see Section 20.2.3.1, “Performance Schema Event Timing”.

The history tables contain the same kind of rows as the current-events table but have more rows and show what the server has been doing “recently” rather than “currently.” The events_waits_history and events_waits_history_long tables contain the most recent 10 events per thread and most recent 10,000 events, respectively. For example, to see information for recent events produced by thread 13, do this:

```
mysql> SELECT EVENT_ID, EVENT_NAME, TIMER_WAIT
    -> FROM events_waits_history WHERE THREAD_ID = 13
    -> ORDER BY EVENT_ID;
+----------+-----------------------------------------+------------+
| EVENT_ID | EVENT_NAME                              | TIMER_WAIT |
+----------+-----------------------------------------+------------+
|       86 | wait/synch/mutex/mysys/THR_LOCK::mutex  |     686322 |
|       87 | wait/synch/mutex/mysys/THR_LOCK_malloc  |     320535 |
|       88 | wait/synch/mutex/mysys/THR_LOCK_malloc  |     339390 |
|       89 | wait/synch/mutex/mysys/THR_LOCK_malloc  |     377100 |
|       90 | wait/synch/mutex/sql/LOCK_plugin        |     614673 |
|       91 | wait/synch/mutex/sql/LOCK_open          |     659925 |
|       92 | wait/synch/mutex/sql/THD::LOCK_thd_data |     494001 |
|       93 | wait/synch/mutex/mysys/THR_LOCK_malloc  |     222489 |
|       94 | wait/synch/mutex/mysys/THR_LOCK_malloc  |     214947 |
|       95 | wait/synch/mutex/mysys/LOCK_alarm       |     312993 |
+----------+-----------------------------------------+------------+
```

As new events are added to a history table, older events are discarded if the table is full.

Summary tables provide aggregated information for all events over time. The tables in this group summarize event data in different ways. To see which instruments have been executed the most times or have taken the most wait time, sort the `events_waits_summary_global_by_event_name` table on the `COUNT_STAR` or `SUM_TIMER_WAIT` column, which correspond to a `COUNT(*)` or `SUM(TIMER_WAIT)` value, respectively, calculated over all events:

```
mysql> SELECT EVENT_NAME, COUNT_STAR
    -> FROM events_waits_summary_global_by_event_name
    -> ORDER BY COUNT_STAR DESC LIMIT 10;
+---------------------------------------------------+------------+
| EVENT_NAME                                        | COUNT_STAR |
+---------------------------------------------------+------------+
| wait/synch/mutex/mysys/THR_LOCK_malloc            |       6419 |
| wait/io/file/sql/FRM                              |        452 |
| wait/synch/mutex/sql/LOCK_plugin                  |        337 |
| wait/synch/mutex/mysys/THR_LOCK_open              |        187 |
| wait/synch/mutex/mysys/LOCK_alarm                 |        147 |
| wait/synch/mutex/sql/THD::LOCK_thd_data           |        115 |
| wait/io/file/myisam/kfile                         |        102 |
| wait/synch/mutex/sql/LOCK_global_system_variables |         89 |
| wait/synch/mutex/mysys/THR_LOCK::mutex            |         89 |
| wait/synch/mutex/sql/LOCK_open                    |         88 |
+---------------------------------------------------+------------+

mysql> SELECT EVENT_NAME, SUM_TIMER_WAIT
    -> FROM events_waits_summary_global_by_event_name
    -> ORDER BY SUM_TIMER_WAIT DESC LIMIT 10;
+----------------------------------------+----------------+
| EVENT_NAME                             | SUM_TIMER_WAIT |
+----------------------------------------+----------------+
| wait/io/file/sql/MYSQL_LOG             |     1599816582 |
| wait/synch/mutex/mysys/THR_LOCK_malloc |     1530083250 |
| wait/io/file/sql/binlog_index          |     1385291934 |
| wait/io/file/sql/FRM                   |     1292823243 |
| wait/io/file/myisam/kfile              |      411193611 |
| wait/io/file/myisam/dfile              |      322401645 |
| wait/synch/mutex/mysys/LOCK_alarm      |      145126935 |
| wait/io/file/sql/casetest              |      104324715 |
| wait/synch/mutex/sql/LOCK_plugin       |       86027823 |
| wait/io/file/sql/pid                   |       72591750 |
+----------------------------------------+----------------+
```

These results show that the `THR_LOCK_malloc` mutex is “hot,” both in terms of how often it is used and amount of time that threads wait attempting to acquire it.

Note

The `THR_LOCK_malloc` mutex is used only in debug builds. In production builds it is not hot because it is nonexistent.

Instance tables document what types of objects are instrumented. An instrumented object, when used by the server, produces an event. These tables provide event names and explanatory notes or status information. For example, the `file_instances` table lists instances of instruments for file I/O operations and their associated files:

```
mysql> SELECT * FROM file_instances\G
*************************** 1. row ***************************
 FILE_NAME: /opt/mysql-log/60500/binlog.000007
EVENT_NAME: wait/io/file/sql/binlog
OPEN_COUNT: 0
*************************** 2. row ***************************
 FILE_NAME: /opt/mysql/60500/data/mysql/tables_priv.MYI
```

```
EVENT_NAME: wait/io/file/myisam/kfile
OPEN_COUNT: 1
*************************** 3. row ***************************
 FILE_NAME: /opt/mysql/60500/data/mysql/columns_priv.MYI
EVENT_NAME: wait/io/file/myisam/kfile
OPEN_COUNT: 1
...
```

Setup tables are used to configure and display monitoring characteristics. For example, to see which event timers are selected, query the `setup_timers` tables:

```
mysql> SELECT * FROM setup_timers;
+-----------+-------------+
| NAME      | TIMER_NAME  |
+-----------+-------------+
| idle      | MICROSECOND |
| wait      | CYCLE       |
| stage     | NANOSECOND  |
| statement | NANOSECOND  |
+-----------+-------------+
```

`setup_instruments` lists the set of instruments for which events can be collected and shows which of them are enabled:

```
mysql> SELECT * FROM setup_instruments;
+------------------------------------------------------------+---------+-------+
| NAME                                                       | ENABLED | TIMED |
+------------------------------------------------------------+---------+-------+
...
| wait/synch/mutex/sql/LOCK_global_read_lock                 | YES     | YES   |
| wait/synch/mutex/sql/LOCK_global_system_variables          | YES     | YES   |
| wait/synch/mutex/sql/LOCK_lock_db                          | YES     | YES   |
| wait/synch/mutex/sql/LOCK_manager                          | YES     | YES   |
...
| wait/synch/rwlock/sql/LOCK_grant                           | YES     | YES   |
| wait/synch/rwlock/sql/LOGGER::LOCK_logger                  | YES     | YES   |
| wait/synch/rwlock/sql/LOCK_sys_init_connect                | YES     | YES   |
| wait/synch/rwlock/sql/LOCK_sys_init_slave                  | YES     | YES   |
...
| wait/io/file/sql/binlog                                    | YES     | YES   |
| wait/io/file/sql/binlog_index                              | YES     | YES   |
| wait/io/file/sql/casetest                                  | YES     | YES   |
| wait/io/file/sql/dbopt                                     | YES     | YES   |
...
```

To understand how to interpret instrument names, see Section 20.4, “Performance Schema Instrument Naming Conventions”.

To control whether events are collected for an instrument, set its `ENABLED` value to `YES` or `NO`. For example:

```
mysql> UPDATE setup_instruments SET ENABLED = 'NO'
    -> WHERE NAME = 'wait/synch/mutex/sql/LOCK_mysql_create_db';
```

The Performance Schema uses collected events to update tables in the `performance_schema` database, which act as “consumers” of event information. The `setup_consumers` table lists the available consumers and which are enabled:

```
mysql> SELECT * FROM setup_consumers;
+----------------------------------+---------+
| NAME                             | ENABLED |
+----------------------------------+---------+
| events_stages_current            | NO      |
| events_stages_history            | NO      |
| events_stages_history_long       | NO      |
| events_statements_current        | YES     |
| events_statements_history        | NO      |
| events_statements_history_long   | NO      |
| events_transactions_current      | YES     |
| events_transactions_history      | NO      |
| events_transactions_history_long | NO      |
| events_waits_current             | NO      |
| events_waits_history             | NO      |
| events_waits_history_long        | NO      |
| global_instrumentation           | YES     |
| thread_instrumentation           | YES     |
| statements_digest                | YES     |
+----------------------------------+---------+
```

To control whether the Performance Schema maintains a consumer as a destination for event information, set its ENABLED value.

For more information about the setup tables and how to use them to control event collection, see Section 20.2.3.2, “Performance Schema Event Filtering”.

There are some miscellaneous tables that do not fall into any of the previous groups. For example, performance_timers lists the available event timers and their characteristics. For information about timers, see Section 20.2.3.1, “Performance Schema Event Timing”.

20.2 Performance Schema Configuration

To use the MySQL Performance Schema, these configuration considerations apply:

- The Performance Schema must be configured into MySQL Server at build time to make it available. Performance Schema support is included in binary MySQL distributions. If you are building from source, you must ensure that it is configured into the build as described in Section 20.2.1, “Performance Schema Build Configuration”.

- The Performance Schema must be enabled at server startup to enable event collection to occur. Specific Performance Schema features can be enabled at server startup or at runtime to control which types of event collection occur. See Section 20.2.2, “Performance Schema Startup Configuration”, Section 20.2.3, “Performance Schema Runtime Configuration”, and Section 20.2.3.2, “Performance Schema Event Filtering”.

20.2.1 Performance Schema Build Configuration

For the Performance Schema to be available, it must be configured into the MySQL server at build time. Binary MySQL distributions provided by Oracle Corporation are configured to support the Performance Schema. If you use a binary MySQL distribution from another provider, check with the provider whether the distribution has been appropriately configured.

If you build MySQL from a source distribution, enable the Performance Schema by running CMake with the WITH_PERFSCHEMA_STORAGE_ENGINE option enabled:

```
shell> cmake . -DWITH_PERFSCHEMA_STORAGE_ENGINE=1
```

Configuring MySQL with the -DWITHOUT_PERFSCHEMA_STORAGE_ENGINE=1 option prevents inclusion of the Performance Schema, so if you want it included, do not use this option. See Section 2.8.4, “MySQL Source-Configuration Options”.

As of MySQL 5.7.3, it is also possible to enable the Performance Schema but exclude certain parts of the instrumentation. For example, to enable the Performance Schema but exclude stage and statement instrumentation, do this:

```
shell> cmake . -DWITH_PERFSCHEMA_STORAGE_ENGINE=1 \
        -DDISABLE_PSI_STAGE=1 \
        -DDISABLE_PSI_STATEMENT=1
```

For more information, see the descriptions of the `DISABLE_PSI_XXX` `CMake` options in Section 2.8.4, “MySQL Source-Configuration Options”.

If you install MySQL over a previous installation that was configured without the Performance Schema (or with an older version of the Performance Schema that may not have all the current tables), run `mysql_upgrade` after starting the server to ensure that the `performance_schema` database exists with all current tables. Then restart the server. One indication that you need to do this is the presence of messages such as the following in the error log:

```
[ERROR] Native table 'performance_schema'.'events_waits_history'
has the wrong structure
[ERROR] Native table 'performance_schema'.'events_waits_history_long'
has the wrong structure
...
```

To verify whether a server was built with Performance Schema support, check its help output. If the Performance Schema is available, the output will mention several variables with names that begin with `performance_schema`:

```
shell> mysqld --verbose --help
...
  --performance_schema
                      Enable the performance schema.
  --performance_schema_events_waits_history_long_size=#
                      Number of rows in events_waits_history_long.
...
```

You can also connect to the server and look for a line that names the `PERFORMANCE_SCHEMA` storage engine in the output from `SHOW ENGINES`:

```
mysql> SHOW ENGINES\G
...
      Engine: PERFORMANCE_SCHEMA
     Support: YES
     Comment: Performance Schema
Transactions: NO
          XA: NO
  Savepoints: NO
...
```

If the Performance Schema was not configured into the server at build time, no row for `PERFORMANCE_SCHEMA` will appear in the output from `SHOW ENGINES`. You might see `performance_schema` listed in the output from `SHOW DATABASES`, but it will have no tables and you will not be able to use it.

A line for `PERFORMANCE_SCHEMA` in the `SHOW ENGINES` output means that the Performance Schema is available, not that it is enabled. To enable it, you must do so at server startup, as described in the next section.

20.2.2 Performance Schema Startup Configuration

Assuming that the Performance Schema is available, it is enabled by default. To enable or disable it explicitly, start the server with the `performance_schema` variable set to an appropriate value. For example, use these lines in your `my.cnf` file:

```
[mysqld]
performance_schema=on
```

If the server is unable to allocate any internal buffer during Performance Schema initialization, the Performance Schema disables itself and sets `performance_schema` to `OFF`, and the server runs without instrumentation.

The Performance Schema also permits instrument and consumer configuration at server startup.

To control an instrument at server startup, use an option of this form:

```
--performance-schema-instrument='instrument_name=value'
```

Here, *`instrument_name`* is an instrument name such as `wait/synch/mutex/sql/LOCK_open`, and *`value`* is one of these values:

- `off`, `false`, or `0`: Disable the instrument
- `on`, `true`, or `1`: Enable and time the instrument
- `counted`: Enable and count (rather than time) the instrument

Each `--performance-schema-instrument` option can specify only one instrument name, but multiple instances of the option can be given to configure multiple instruments. In addition, patterns are permitted in instrument names to configure instruments that match the pattern. To configure all condition synchronization instruments as enabled and counted, use this option:

```
--performance-schema-instrument='wait/synch/cond/%=counted'
```

To disable all instruments, use this option:

```
--performance-schema-instrument='%=off'
```

Longer instrument name strings take precedence over shorter pattern names, regardless of order. For information about specifying patterns to select instruments, see Section 20.2.3.4, “Naming Instruments or Consumers for Filtering Operations”.

An unrecognized instrument name is ignored. It is possible that a plugin installed later may create the instrument, at which time the name is recognized and configured.

To control a consumer at server startup, use an option of this form:

```
--performance-schema-consumer-consumer_name=value
```

Here, *`consumer_name`* is a consumer name such as `events_waits_history`, and *`value`* is one of these values:

- `off`, `false`, or `0`: Do not collect events for the consumer
- `on`, `true`, or `1`: Collect events for the consumer

For example, to enable the `events_waits_history` consumer, use this option:

```
--performance-schema-consumer-events-waits-history=on
```

The permitted consumer names can be found by examining the `setup_consumers` table. Patterns are not permitted. Consumer names in the `setup_consumers` table use underscores, but for consumers set at startup, dashes and underscores within the name are equivalent.

The Performance Schema includes several system variables that provide configuration information:

```
mysql> SHOW VARIABLES LIKE 'perf%';
+--------------------------------------------------------+---------+
| Variable_name                                          | Value   |
+--------------------------------------------------------+---------+
| performance_schema                                     | ON      |
| performance_schema_accounts_size                       | 100     |
| performance_schema_digests_size                        | 200     |
| performance_schema_events_stages_history_long_size     | 10000   |
| performance_schema_events_stages_history_size          | 10      |
| performance_schema_events_statements_history_long_size | 10000   |
| performance_schema_events_statements_history_size      | 10      |
| performance_schema_events_waits_history_long_size      | 10000   |
| performance_schema_events_waits_history_size           | 10      |
| performance_schema_hosts_size                          | 100     |
| performance_schema_max_cond_classes                    | 80      |
| performance_schema_max_cond_instances                  | 1000    |
...
```

The `performance_schema` variable is `ON` or `OFF` to indicate whether the Performance Schema is enabled or disabled. The other variables indicate table sizes (number of rows) or memory allocation values.

Note

With the Performance Schema enabled, the number of Performance Schema instances affects the server memory footprint, perhaps to a large extent. It may be necessary to tune the values of Performance Schema system variables to find the number of instances that balances insufficient instrumentation against excessive memory consumption.

To change the value of Performance Schema system variables, set them at server startup. For example, put the following lines in a `my.cnf` file to change the sizes of the history tables:

```
[mysqld]
performance_schema
performance_schema_events_waits_history_size=20
performance_schema_events_waits_history_long_size=15000
```

The Performance Schema automatically sizes the values of several of its parameters at server startup if they are not set explicitly. For example, it sizes the parameters that control the sizes of the events waits tables this way. To see which parameters are sized under this policy, use `mysqld --verbose --help` and look for those with a default value of –1, or see Section 20.12, “Performance Schema System Variables”.

For each autosized parameter that is not set at server startup (or is set to –1), the Performance Schema determines how to set its value based on the value of the following system values, which are considered as “hints” about how you have configured your MySQL server:

```
max_connections
open_files_limit
table_definition_cache
table_open_cache
```

To override autosizing for a given parameter, set it a value other than –1 at startup. In this case, the Performance Schema assigns it the specified value.

At runtime, SHOW VARIABLES displays the actual values that autosized parameters were set to.

If the Performance Schema is disabled, its autosized parameters remain set to –1 and SHOW VARIABLES displays –1.

20.2.3 Performance Schema Runtime Configuration

Performance Schema setup tables contain information about monitoring configuration:

```
mysql> SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES
    -> WHERE TABLE_SCHEMA = 'performance_schema'
    -> AND TABLE_NAME LIKE 'setup%';
+-------------------+
| TABLE_NAME        |
+-------------------+
| setup_actors      |
| setup_consumers   |
| setup_instruments |
| setup_objects     |
| setup_timers      |
+-------------------+
```

You can examine the contents of these tables to obtain information about Performance Schema monitoring characteristics. If you have the UPDATE privilege, you can change Performance Schema operation by modifying setup tables to affect how monitoring occurs. For additional details about these tables, see Section 20.9.2, “Performance Schema Setup Tables”.

To see which event timers are selected, query the setup_timers tables:

```
mysql> SELECT * FROM setup_timers;
+-----------+-------------+
| NAME      | TIMER_NAME  |
+-----------+-------------+
| idle      | MICROSECOND |
| wait      | CYCLE       |
| stage     | NANOSECOND  |
| statement | NANOSECOND  |
+-----------+-------------+
```

The NAME value indicates the type of instrument to which the timer applies, and TIMER_NAME indicates which timer applies to those instruments. The timer applies to instruments where their name begins with a component matching the NAME value.

To change the timer, update the NAME value. For example, to use the NANOSECOND timer for the wait timer:

```
mysql> UPDATE setup_timers SET TIMER_NAME = 'NANOSECOND'
    -> WHERE NAME = 'wait';
mysql> SELECT * FROM setup_timers;
+-----------+-------------+
| NAME      | TIMER_NAME  |
+-----------+-------------+
| idle      | MICROSECOND |
| wait      | NANOSECOND  |
| stage     | NANOSECOND  |
| statement | NANOSECOND  |
+-----------+-------------+
```

For discussion of timers, see Section 20.2.3.1, “Performance Schema Event Timing”.

The `setup_instruments` and `setup_consumers` tables list the instruments for which events can be collected and the types of consumers for which event information actually is collected, respectively. Other setup tables enable further modification of the monitoring configuration. Section 20.2.3.2, “Performance Schema Event Filtering”, discusses how you can modify these tables to affect event collection.

If there are Performance Schema configuration changes that must be made at runtime using SQL statements and you would like these changes to take effect each time the server starts, put the statements in a file and start the server with the `--init-file=file_name` option. This strategy can also be useful if you have multiple monitoring configurations, each tailored to produce a different kind of monitoring, such as casual server health monitoring, incident investigation, application behavior troubleshooting, and so forth. Put the statements for each monitoring configuration into their own file and specify the appropriate file as the `--init-file` argument when you start the server.

20.2.3.1 Performance Schema Event Timing

Events are collected by means of instrumentation added to the server source code. Instruments time events, which is how the Performance Schema provides an idea of how long events take. It is also possible to configure instruments not to collect timing information. This section discusses the available timers and their characteristics, and how timing values are represented in events.

Two tables provide timer information:

- `performance_timers` lists the available timers and their characteristics.
- `setup_timers` indicates which timers are used for which instruments.

Each timer row in `setup_timers` must refer to one of the timers listed in `performance_timers`.

Timers vary in precision and the amount of overhead they involve. To see what timers are available and their characteristics, check the `performance_timers` table:

```
mysql> SELECT * FROM performance_timers;
+-------------+-----------------+------------------+----------------+
| TIMER_NAME  | TIMER_FREQUENCY | TIMER_RESOLUTION | TIMER_OVERHEAD |
+-------------+-----------------+------------------+----------------+
| CYCLE       |      2389029850 |                1 |             72 |
| NANOSECOND  |            NULL |             NULL |           NULL |
| MICROSECOND |         1000000 |                1 |            585 |
| MILLISECOND |            1035 |                1 |            738 |
| TICK        |             101 |                1 |            630 |
+-------------+-----------------+------------------+----------------+
```

The `TIMER_NAME` column shows the names of the available timers. `CYCLE` refers to the timer that is based on the CPU (processor) cycle counter. If the values associated with a given timer name are `NULL`, that timer is not supported on your platform. The rows that do not have `NULL` indicate which timers you can use in `setup_timers`.

`TIMER_FREQUENCY` indicates the number of timer units per second. For a cycle timer, the frequency is generally related to the CPU speed. The value shown was obtained on a system with a 2.4GHz processor. The other timers are based on fixed fractions of seconds. For `TICK`, the frequency may vary by platform (for example, some use 100 ticks/second, others 1000 ticks/second).

TIMER_RESOLUTION indicates the number of timer units by which timer values increase at a time. If a timer has a resolution of 10, its value increases by 10 each time.

TIMER_OVERHEAD is the minimal number of cycles of overhead to obtain one timing with the given timer. The overhead per event is twice the value displayed because the timer is invoked at the beginning and end of the event.

To see which timer is in effect or to change the timer, access the setup_timers table:

```
mysql> SELECT * FROM setup_timers;
+-----------+-------------+
| NAME      | TIMER_NAME  |
+-----------+-------------+
| idle      | MICROSECOND |
| wait      | NANOSECOND  |
| stage     | NANOSECOND  |
| statement | NANOSECOND  |
+-----------+-------------+

mysql> UPDATE setup_timers SET TIMER_NAME = 'MICROSECOND'
    -> WHERE NAME = 'wait';
mysql> SELECT * FROM setup_timers;
+-----------+-------------+
| NAME      | TIMER_NAME  |
+-----------+-------------+
| idle      | MICROSECOND |
| wait      | MICROSECOND |
| stage     | NANOSECOND  |
| statement | NANOSECOND  |
+-----------+-------------+
```

By default, the Performance Schema uses the best timer available for each instrument type, but you can select a different one.

To time waits, the most important criterion is to reduce overhead, at the possible expense of the timer accuracy, so using the CYCLE timer is the best.

The time a statement (or stage) takes to execute is in general orders of magnitude larger than the time it takes to execute a single wait. To time statements, the most important criterion is to have an accurate measure, which is not affected by changes in processor frequency, so using a timer which is not based on cycles is the best. The default timer for statements is NANOSECOND. The extra “overhead” compared to the cycle timer is not significant, because the overhead caused by calling a timer twice (once when the statement starts, once when it ends) is orders of magnitude less compared to the CPU time used to execute the statement itself. Using the CYCLE timer has no benefit here, only drawbacks.

The precision offered by the cycle counter depends on processor speed. If the processor runs at 1 GHz (one billion cycles/second) or higher, the cycle counter delivers sub-nanosecond precision. Using the cycle counter is much cheaper than getting the actual time of day. For example, the standard gettimeofday() function can take hundreds of cycles, which is an unacceptable overhead for data gathering that may occur thousands or millions of times per second.

Cycle counters also have disadvantages:

- End users expect to see timings in wall-clock units, such as fractions of a second. Converting from cycles to fractions of seconds can be expensive. For this reason, the conversion is a quick and fairly rough multiplication operation.
- Processor cycle rate might change, such as when a laptop goes into power-saving mode or when a CPU slows down to reduce heat generation. If a processor's cycle rate fluctuates, conversion from cycles to real-time units is subject to error.

- Cycle counters might be unreliable or unavailable depending on the processor or the operating system. For example, on Pentiums, the instruction is `RDTSC` (an assembly-language rather than a C instruction) and it is theoretically possible for the operating system to prevent user-mode programs from using it.

- Some processor details related to out-of-order execution or multiprocessor synchronization might cause the counter to seem fast or slow by up to 1000 cycles.

Currently, MySQL works with cycle counters on x386 (Windows, Mac OS X, Linux, Solaris, and other Unix flavors), PowerPC, and IA-64.

The `setup_instruments` table has an `ENABLED` column to indicate the instruments for which to collect events. The table also has a `TIMED` column to indicate which instruments are timed. If an instrument is not enabled, it produces no events. If an enabled instrument is not timed, events produced by the instrument have `NULL` for the `TIMER_START`, `TIMER_END`, and `TIMER_WAIT` timer values. This in turn causes those values to be ignored when calculating the sum, minimum, maximum, and average time values in summary tables.

Within events, times are stored in units given by the timer in effect when event timing begins. For display, times are shown in picoseconds (trillionths of a second) to normalize them to a standard unit, regardless of which timer is selected.

Modifications to the `setup_timers` table affect monitoring immediately. Events already in progress may use the original timer for the begin time and the new timer for the end time, which may lead to unpredictable results. If you make timer changes, you may want to use `TRUNCATE TABLE` to reset Performance Schema statistics.

The timer baseline (“time zero”) occurs at Performance Schema initialization during server startup. `TIMER_START` and `TIMER_END` values in events represent picoseconds since the baseline. `TIMER_WAIT` values are durations in picoseconds.

Picosecond values in events are approximate. Their accuracy is subject to the usual forms of error associated with conversion from one unit to another. If the `CYCLE` timer is used and the processor rate varies, there might be drift. For these reasons, it is not reasonable to look at the `TIMER_START` value for an event as an accurate measure of time elapsed since server startup. On the other hand, it is reasonable to use `TIMER_START` or `TIMER_WAIT` values in `ORDER BY` clauses to order events by start time or duration.

The choice of picoseconds in events rather than a value such as microseconds has a performance basis. One implementation goal was to show results in a uniform time unit, regardless of the timer. In an ideal world this time unit would look like a wall-clock unit and be reasonably precise; in other words, microseconds. But to convert cycles or nanoseconds to microseconds, it would be necessary to perform a division for every instrumentation. Division is expensive on many platforms. Multiplication is not expensive, so that is what is used. Therefore, the time unit is an integer multiple of the highest possible `TIMER_FREQUENCY` value, using a multiplier large enough to ensure that there is no major precision loss. The result is that the time unit is “picoseconds.” This precision is spurious, but the decision enables overhead to be minimized.

20.2.3.2 Performance Schema Event Filtering

Events are processed in a producer/consumer fashion:

- Instrumented code is the source for events and produces events to be collected. The `setup_instruments` table lists the instruments for which events can be collected, whether they are enabled, and (for enabled instruments) whether to collect timing information:

```
mysql> SELECT * FROM setup_instruments;
+------------------------------------------------------------+---------+-------+
| NAME                                                       | ENABLED | TIMED |
+------------------------------------------------------------+---------+-------+
...
| wait/synch/mutex/sql/LOCK_global_read_lock                 | YES     | YES   |
| wait/synch/mutex/sql/LOCK_global_system_variables          | YES     | YES   |
| wait/synch/mutex/sql/LOCK_lock_db                          | YES     | YES   |
| wait/synch/mutex/sql/LOCK_manager                          | YES     | YES   |
...
```

The `setup_instruments` table provides the most basic form of control over event production. To further refine event production based on the type of object or thread being monitored, other tables may be used as described in Section 20.2.3.3, “Event Pre-Filtering”.

- Performance Schema tables are the destinations for events and consume events. The `setup_consumers` table lists the types of consumers to which event information can be sent and whether they are enabled:

```
mysql> SELECT * FROM setup_consumers;
+----------------------------------+---------+
| NAME                             | ENABLED |
+----------------------------------+---------+
| events_stages_current            | NO      |
| events_stages_history            | NO      |
| events_stages_history_long       | NO      |
| events_statements_current        | YES     |
| events_statements_history        | NO      |
| events_statements_history_long   | NO      |
| events_transactions_current      | YES     |
| events_transactions_history      | NO      |
| events_transactions_history_long | NO      |
| events_waits_current             | NO      |
| events_waits_history             | NO      |
| events_waits_history_long        | NO      |
| global_instrumentation           | YES     |
| thread_instrumentation           | YES     |
| statements_digest                | YES     |
+----------------------------------+---------+
```

Filtering can be done at different stages of performance monitoring:

- **Pre-filtering.** This is done by modifying Performance Schema configuration so that only certain types of events are collected from producers, and collected events update only certain consumers. To do this, enable or disable instruments or consumers. Pre-filtering is done by the Performance Schema and has a global effect that applies to all users.

  Reasons to use pre-filtering:

  - To reduce overhead. Performance Schema overhead should be minimal even with all instruments enabled, but perhaps you want to reduce it further. Or you do not care about timing events and want to disable the timing code to eliminate timing overhead.
  - To avoid filling the current-events or history tables with events in which you have no interest. Pre-filtering leaves more “room” in these tables for instances of rows for enabled instrument types. If you enable only file instruments with pre-filtering, no rows are collected for nonfile instruments. With post-filtering, nonfile events are collected, leaving fewer rows for file events.
  - To avoid maintaining some kinds of event tables. If you disable a consumer, the server does not spend time maintaining destinations for that consumer. For example, if you do not care about event histories, you can disable the history table consumers to improve performance.

- **Post-filtering.** This involves the use of WHERE clauses in queries that select information from Performance Schema tables, to specify which of the available events you want to see. Post-filtering is performed on a per-user basis because individual users select which of the available events are of interest.

  Reasons to use post-filtering:

  - To avoid making decisions for individual users about which event information is of interest.
  - To use the Performance Schema to investigate a performance issue when the restrictions to impose using pre-filtering are not known in advance.

The following sections provide more detail about pre-filtering and provide guidelines for naming instruments or consumers in filtering operations. For information about writing queries to retrieve information (post-filtering), see Section 20.3, “Performance Schema Queries”.

20.2.3.3 Event Pre-Filtering

Pre-filtering is done by the Performance Schema and has a global effect that applies to all users. Pre-filtering can be applied to either the producer or consumer stage of event processing:

- To configure pre-filtering at the producer stage, several tables can be used:
  - setup_instruments indicates which instruments are available. An instrument disabled in this table produces no events regardless of the contents of the other production-related setup tables. An instrument enabled in this table is permitted to produce events, subject to the contents of the other tables.
  - setup_objects controls whether the Performance Schema monitors particular table and stored program objects.
  - threads indicates whether monitoring is enabled for each server thread.
  - setup_actors determines the initial monitoring state for new foreground threads.
- To configure pre-filtering at the consumer stage, modify the setup_consumers table. This determines the destinations to which events are sent. setup_consumers also implicitly affects event production. If a given event will not be sent to any destination (that is, will not be consumed), the Performance Schema does not produce it.

Modifications to any of these tables affect monitoring immediately, with the exception of setup_actors. A change to setup_actors affects only foreground threads created subsequent to the change.

When you change the monitoring configuration, the Performance Schema does not flush the history tables. Events already collected remain in the current-events and history tables until displaced by newer events. If you disable instruments, you might need to wait a while before events for them are displaced by newer events of interest. Alternatively, use TRUNCATE TABLE to empty the history tables.

After making instrumentation changes, you might want to truncate the summary tables to clear aggregate information for previously collected events. Except for events_statements_summary_by_digest and the memory summary tables, the effect of TRUNCATE TABLE for summary tables is to reset the summary columns to 0 or NULL, not to remove rows.

The following sections describe how to use specific tables to control Performance Schema pre-filtering.

Pre-Filtering by Instrument

The setup_instruments table lists the available instruments:

```
mysql> SELECT * FROM setup_instruments;
+------------------------------------------------------------+---------+-------+
| NAME                                                       | ENABLED | TIMED |
+------------------------------------------------------------+---------+-------+
...
| wait/synch/mutex/sql/LOCK_global_read_lock                 | YES     | YES   |
| wait/synch/mutex/sql/LOCK_global_system_variables          | YES     | YES   |
| wait/synch/mutex/sql/LOCK_lock_db                          | YES     | YES   |
| wait/synch/mutex/sql/LOCK_manager                          | YES     | YES   |
...
| wait/synch/rwlock/sql/LOCK_grant                           | YES     | YES   |
| wait/synch/rwlock/sql/LOGGER::LOCK_logger                  | YES     | YES   |
| wait/synch/rwlock/sql/LOCK_sys_init_connect                | YES     | YES   |
| wait/synch/rwlock/sql/LOCK_sys_init_slave                  | YES     | YES   |
...
| wait/io/file/sql/binlog                                    | YES     | YES   |
| wait/io/file/sql/binlog_index                              | YES     | YES   |
| wait/io/file/sql/casetest                                  | YES     | YES   |
| wait/io/file/sql/dbopt                                     | YES     | YES   |
...
```

To control whether an instrument is enabled, set its ENABLED column to YES or NO. To configure whether to collect timing information for an enabled instrument, set its TIMED value to YES or NO. Setting the TIMED column affects Performance Schema table contents as described in Section 20.2.3.1, “Performance Schema Event Timing”.

Modifications to the setup_instruments table affect monitoring immediately.

The setup_instruments table provides the most basic form of control over event production. To further refine event production based on the type of object or thread being monitored, other tables may be used as described in Section 20.2.3.3, “Event Pre-Filtering”.

The following examples demonstrate possible operations on the setup_instruments table. These changes, like other pre-filtering operations, affect all users. Some of these queries use the LIKE operator and a pattern match instrument names. For additional information about specifying patterns to select instruments, see Section 20.2.3.4, “Naming Instruments or Consumers for Filtering Operations”.

- Disable all instruments:

  ```
  mysql> UPDATE setup_instruments SET ENABLED = 'NO';
  ```

  Now no events will be collected.

- Disable all file instruments, adding them to the current set of disabled instruments:

  ```
  mysql> UPDATE setup_instruments SET ENABLED = 'NO'
      -> WHERE NAME LIKE 'wait/io/file/%';
  ```

- Disable only file instruments, enable all other instruments:

  ```
  mysql> UPDATE setup_instruments
      -> SET ENABLED = IF(NAME LIKE 'wait/io/file/%', 'NO', 'YES');
  ```

- Enable all but those instruments in the mysys library:

  ```
  mysql> UPDATE setup_instruments
      -> SET ENABLED = CASE WHEN NAME LIKE '%/mysys/%' THEN 'YES' ELSE 'NO' END;
  ```

- Disable a specific instrument:

```
mysql> UPDATE setup_instruments SET ENABLED = 'NO'
    -> WHERE NAME = 'wait/synch/mutex/mysys/TMPDIR_mutex';
```

- To toggle the state of an instrument, “flip” its ENABLED value:

```
mysql> UPDATE setup_instruments
    -> SET ENABLED = IF(ENABLED = 'YES', 'NO', 'YES')
    -> WHERE NAME = 'wait/synch/mutex/mysys/TMPDIR_mutex';
```

- Disable timing for all events:

```
mysql> UPDATE setup_instruments SET TIMED = 'NO';
```

Pre-Filtering by Object

The setup_objects table controls whether the Performance Schema monitors particular table and stored program objects. The initial setup_objects contents look like this:

```
mysql> SELECT * FROM setup_objects;
+-------------+--------------------+-------------+---------+-------+
| OBJECT_TYPE | OBJECT_SCHEMA      | OBJECT_NAME | ENABLED | TIMED |
+-------------+--------------------+-------------+---------+-------+
| EVENT       | mysql              | %           | NO      | NO    |
| EVENT       | performance_schema | %           | NO      | NO    |
| EVENT       | information_schema | %           | NO      | NO    |
| EVENT       | %                  | %           | YES     | YES   |
| FUNCTION    | mysql              | %           | NO      | NO    |
| FUNCTION    | performance_schema | %           | NO      | NO    |
| FUNCTION    | information_schema | %           | NO      | NO    |
| FUNCTION    | %                  | %           | YES     | YES   |
| PROCEDURE   | mysql              | %           | NO      | NO    |
| PROCEDURE   | performance_schema | %           | NO      | NO    |
| PROCEDURE   | information_schema | %           | NO      | NO    |
| PROCEDURE   | %                  | %           | YES     | YES   |
| TABLE       | mysql              | %           | NO      | NO    |
| TABLE       | performance_schema | %           | NO      | NO    |
| TABLE       | information_schema | %           | NO      | NO    |
| TABLE       | %                  | %           | YES     | YES   |
| TRIGGER     | mysql              | %           | NO      | NO    |
| TRIGGER     | performance_schema | %           | NO      | NO    |
| TRIGGER     | information_schema | %           | NO      | NO    |
| TRIGGER     | %                  | %           | YES     | YES   |
+-------------+--------------------+-------------+---------+-------+
```

Modifications to the setup_objects table affect object monitoring immediately.

The OBJECT_TYPE column indicates the type of object to which a row applies. TABLE filtering affects table I/O events (wait/io/table/sql/handler instrument) and table lock events (wait/lock/table/sql/handler instrument).

The OBJECT_SCHEMA and OBJECT_NAME columns should contain a literal schema or object name, or '%' to match any name.

The ENABLED column indicates whether matching objects are monitored, and TIMED indicates whether to collect timing information.

The effect of the default object configuration is to instrument all objects except those in the mysql, INFORMATION_SCHEMA, and performance_schema databases. (Tables in the INFORMATION_SCHEMA database are not instrumented regardless of the contents of setup_objects; the row for information_schema.% simply makes this default explicit.)

When the Performance Schema checks for a match in setup_objects, it tries to find more specific matches first. For example, with a table db1.t1, it looks in TABLE rows for a match for 'db1' and 't1', then for 'db1' and '%', then for '%' and '%'. The order in which matching occurs matters because different matching setup_objects rows can have different ENABLED and TIMED values.

For object-related events, the Performance Schema combines the contents of setup_objects with setup_instruments to determine whether to enable instruments and whether to time enabled instruments:

- For objects that match a row in setup_objects, object instruments produce events only if they are enabled in both setup_instruments and setup_objects.
- The TIMED values in the two tables are combined, so that timing information is collected only when both values are YES.

Suppose that setup_objects contains the following rows that apply to db1, db2, and db3:

```
+-------------+---------------+-------------+---------+-------+
| OBJECT_TYPE | OBJECT_SCHEMA | OBJECT_NAME | ENABLED | TIMED |
+-------------+---------------+-------------+---------+-------+
| TABLE       | db1           | t1          | YES     | YES   |
| TABLE       | db1           | t2          | YES     | NO    |
| TABLE       | db2           | %           | YES     | YES   |
| TABLE       | db3           | %           | YES     | NO    |
| TABLE       | %             | %           | YES     | YES   |
+-------------+---------------+-------------+---------+-------+
```

If an object-related instrument in setup_instruments has a TIMED value of NO, no events for the instrument are timed. If the TIMED value is YES, event timing occurs according to the TIMED value in the relevant setup_objects row:

- db1.t1 events are timed
- db1.t2 events are not timed
- db2.t3 events are timed
- db3.t4 events are not timed
- db4.t5 events are timed

If a persistent table and a temporary table have the same name, matching against setup_objects rows occurs the same way for both. It is not possible to enable monitoring for one table but not the other. However, each table is instrumented separately.

Pre-Filtering by Thread

The threads table contains a row for each server thread. Each row contains information about a thread and indicates whether monitoring is enabled for it. For the Performance Schema to monitor a thread, these things must be true:

- The thread_instrumentation consumer in the setup_consumers table must be YES.
- The thread.INSTRUMENTED column must be YES.
- Monitoring occurs only for those thread events produced from instruments that are enabled in the setup_instruments table.

The INSTRUMENTED column in the threads table indicates the monitoring state for each thread. For foreground threads (resulting from client connections), the initial INSTRUMENTED value is determined by whether the user account associated with the thread matches any row in the setup_actors table. For background threads, INSTRUMENTED is YES by default and setup_actors is not consulted because there is no associated user for background threads. For any thread, its INSTRUMENTED value can be changed during the life of the thread.

The initial setup_actors contents look like this:

```
mysql> SELECT * FROM setup_actors;
+------+------+------+
| HOST | USER | ROLE |
+------+------+------+
| %    | %    | %    |
+------+------+------+
```

The Performance Schema uses the HOST and USER columns to match each new foreground thread. (ROLE is unused.) The INSTRUMENTED value for the thread becomes YES if any row matches, NO otherwise. This enables instrumenting to be applied selectively per host, user, or combination of host and user.

The HOST and USER columns should contain a literal host or user name, or '%' to match any name. By default, monitoring is enabled for all new foreground threads because the setup_actors table initially contains a row with '%' for both HOST and USER. To perform more limited matching such as to enable monitoring only for some foreground threads, you must delete this row because it matches any connection.

Suppose that you modify setup_actors as follows:

```
DELETE FROM setup_actors;
```

Now setup_actors is empty and there are no rows that could match incoming connections. Consequently, the Performance Schema will set the INSTRUMENTED column to NO for all new foreground threads.

Suppose that you further modify setup_actors:

```
INSERT INTO setup_actors (HOST,USER,ROLE) VALUES('localhost','joe','%');
INSERT INTO setup_actors (HOST,USER,ROLE) VALUES('%','sam','%');
```

Now the Performance Schema determines how to set the INSTRUMENTED value for new connection threads as follows:

- If joe connects from the local host, the connection matches the first inserted row.
- If joe connects from any other host, there is no match.
- If sam connects from any host, the connection matches the second inserted row.
- For any other connection, there is no match.

Modifications to the setup_actors table do not affect existing threads.

Pre-Filtering by Consumer

The setup_consumers table lists the available consumer types and which are enabled:

```
mysql> SELECT * FROM setup_consumers;
+----------------------------------+---------+
| NAME                             | ENABLED |
+----------------------------------+---------+
| events_stages_current            | NO      |
| events_stages_history            | NO      |
| events_stages_history_long       | NO      |
| events_statements_current        | YES     |
| events_statements_history        | NO      |
| events_statements_history_long   | NO      |
| events_transactions_current      | YES     |
| events_transactions_history      | NO      |
| events_transactions_history_long | NO      |
| events_waits_current             | NO      |
| events_waits_history             | NO      |
| events_waits_history_long        | NO      |
| global_instrumentation           | YES     |
| thread_instrumentation           | YES     |
| statements_digest                | YES     |
+----------------------------------+---------+
```

Modify the setup_consumers table to affect pre-filtering at the consumer stage and determine the destinations to which events are sent. To enable or disable a consumer, set its ENABLED value to YES or NO.

Modifications to the setup_consumers table affect monitoring immediately.

If you disable a consumer, the server does not spend time maintaining destinations for that consumer. For example, if you do not care about historical event information, disable the history consumers:

```
mysql> UPDATE setup_consumers
    -> SET ENABLED = 'NO' WHERE NAME LIKE '%history%';
```

The consumer settings in the setup_consumers table form a hierarchy from higher levels to lower. The following principles apply:

- Destinations associated with a consumer receive no events unless the Performance Schema checks the consumer and the consumer is enabled.
- A consumer is checked only if all consumers it depends on (if any) are enabled.
- If a consumer is not checked, or is checked but is disabled, other consumers that depend on it are not checked.
- Dependent consumers may have their own dependent consumers.
- If an event would not be sent to any destination, the Performance Schema does not produce it.

The following lists describe the available consumer values. For discussion of several representative consumer configurations and their effect on instrumentation, see Example Consumer Configurations.

Global and Thread Consumers

- global_instrumentation is the highest level consumer. If global_instrumentation is NO, it disables global instrumentation. All other settings are lower level and are not checked; it does not matter what they are set to. No global or per thread information is maintained and no individual events are collected in the current-events or event-history tables. If global_instrumentation is YES, the Performance Schema maintains information for global states and also checks the thread_instrumentation consumer.
- thread_instrumentation is checked only if global_instrumentation is YES. Otherwise, if thread_instrumentation is NO, it disables thread-specific instrumentation and all lower-level settings are ignored. No information is maintained per thread and no individual events are collected in the current-events or event-history tables. If thread_instrumentation is YES, the Performance Schema maintains thread-specific information and also checks events_*xxx*_current consumers.

Statement Digest Consumer

This consumer requires global_instrumentation to be YES or it is not checked. There is no dependency on the statement event consumers, so you can obtain statistics per digest without having to collect statistics in events_statements_current, which is advantageous in terms of overhead.

Wait Event Consumers

These consumers require both global_instrumentation and thread_instrumentation to be YES or they are not checked. If checked, they act as follows:

- events_waits_current, if NO, disables collection of individual wait events in the events_waits_current table. If YES, it enables wait event collection and the Performance Schema checks the events_waits_history and events_waits_history_long consumers.
- events_waits_history is not checked if event_waits_current is NO. Otherwise, an events_waits_history value of NO or YES disables or enables collection of wait events in the events_waits_history table.
- events_waits_history_long is not checked if event_waits_current is NO. Otherwise, an events_waits_history_long value of NO or YES disables or enables collection of wait events in the events_waits_history_long table.

Stage Event Consumers

These consumers require both global_instrumentation and thread_instrumentation to be YES or they are not checked. If checked, they act as follows:

- events_stages_current, if NO, disables collection of individual stage events in the events_stages_current table. If YES, it enables stage event collection and the Performance Schema checks the events_stages_history and events_stages_history_long consumers.
- events_stages_history is not checked if event_stages_current is NO. Otherwise, an events_stages_history value of NO or YES disables or enables collection of stage events in the events_stages_history table.
- events_stages_history_long is not checked if event_stages_current is NO. Otherwise, an events_stages_history_long value of NO or YES disables or enables collection of stage events in the events_stages_history_long table.

Statement Event Consumers

These consumers require both global_instrumentation and thread_instrumentation to be YES or they are not checked. If checked, they act as follows:

- events_statements_current, if NO, disables collection of individual statement events in the events_statements_current table. If YES, it enables statement event collection and the Performance Schema checks the events_statements_history and events_statements_history_long consumers.
- events_statements_history is not checked if events_statements_current is NO. Otherwise, an events_statements_history value of NO or YES disables or enables collection of statement events in the events_statements_history table.
- events_statements_history_long is not checked if events_statements_current is NO. Otherwise, an events_statements_history_long value of NO or YES disables or enables collection of statement events in the events_statements_history_long table.

Transaction Event Consumers

These consumers require both global_instrumentation and thread_instrumentation to be YES or they are not checked. If checked, they act as follows:

- events_transactions_current, if NO, disables collection of individual transaction events in the events_transactions_current table. If YES, it enables transaction event collection and the Performance Schema checks the events_transactions_history and events_transactions_history_long consumers.
- events_transactions_history is not checked if events_transactions_current is NO. Otherwise, an events_transactions_history value of NO or YES disables or enables collection of transaction events in the events_transactions_history table.
- events_transactions_history_long is not checked if events_transactions_current is NO. Otherwise, an events_transactions_history_long value of NO or YES disables or enables collection of transaction events in the events_transactions_history_long table.

Example Consumer Configurations

The consumer settings in the setup_consumers table form a hierarchy from higher levels to lower. The following discussion describes how consumers work, showing specific configurations and their effects as consumer settings are enabled progressively from high to low. The consumer values shown are representative. The general principles described here apply to other consumer values that may be available.

The configuration descriptions occur in order of increasing functionality and overhead. If you do not need the information provided by enabling lower-level settings, disable them and the Performance Schema will execute less code on your behalf and you will have less information to sift through.

The setup_consumers table contains the following hierarchy of values:

```
global_instrumentation
 thread_instrumentation
   events_waits_current
     events_waits_history
     events_waits_history_long
   events_stages_current
     events_stages_history
     events_stages_history_long
   events_statements_current
     events_statements_history
     events_statements_history_long
   events_transactions_current
     events_transactions_history
     events_transactions_history_long
```

Note

In the consumer hierarchy, the consumers for waits, stages, statements, and transactions are all at the same level. This differs from the event nesting hierarchy, for which wait events nest within stage events, which nest within statement events, which nest within transaction events.

If a given consumer setting is NO, the Performance Schema disables the instrumentation associated with the consumer and ignores all lower-level settings. If a given setting is YES, the Performance Schema enables the instrumentation associated with it and checks the settings at the next lowest level. For a description of the rules for each consumer, see Pre-Filtering by Consumer.

For example, if global_instrumentation is enabled, thread_instrumentation is checked. If thread_instrumentation is enabled, the events_*xxx*_current consumers are checked. If of these events_waits_current is enabled, events_waits_history and events_waits_history_long are checked.

Each of the following configuration descriptions indicates which setup elements the Performance Schema checks and which output tables it maintains (that is, for which tables it collects information).

No Instrumentation

Server configuration state:

```
mysql> SELECT * FROM setup_consumers;
+---------------------------+---------+
| NAME                      | ENABLED |
+---------------------------+---------+
| global_instrumentation    | NO      |
...
+---------------------------+---------+
```

In this configuration, nothing is instrumented.

Setup elements checked:

- Table setup_consumers, consumer global_instrumentation

Output tables maintained:

- None

Global Instrumentation Only

Server configuration state:

```
mysql> SELECT * FROM setup_consumers;
+---------------------------+---------+
| NAME                      | ENABLED |
+---------------------------+---------+
| global_instrumentation    | YES     |
| thread_instrumentation    | NO      |
...
+---------------------------+---------+
```

In this configuration, instrumentation is maintained only for global states. Per-thread instrumentation is disabled.

Additional setup elements checked, relative to the preceding configuration:

- Table setup_consumers, consumer thread_instrumentation
- Table setup_instruments
- Table setup_objects
- Table setup_timers

Additional output tables maintained, relative to the preceding configuration:

- mutex_instances
- rwlock_instances
- cond_instances

- file_instances
- users
- hosts
- accounts
- socket_summary_by_event_name
- file_summary_by_instance
- file_summary_by_event_name
- objects_summary_global_by_type
- memory_summary_global_by_event_name
- table_lock_waits_summary_by_table
- table_io_waits_summary_by_index_usage
- table_io_waits_summary_by_table
- events_waits_summary_by_instance
- events_waits_summary_global_by_event_name
- events_stages_summary_global_by_event_name
- events_statements_summary_global_by_event_name
- events_transactions_summary_global_by_event_name

Global and Thread Instrumentation Only

Server configuration state:

```
mysql> SELECT * FROM setup_consumers;
+----------------------------------+---------+
| NAME                             | ENABLED |
+----------------------------------+---------+
| global_instrumentation           | YES     |
| thread_instrumentation           | YES     |
| events_waits_current             | NO      |
...
| events_stages_current            | NO      |
...
| events_statements_current        | NO      |
...
| events_transactions_current      | NO      |
...
+----------------------------------+---------+
```

In this configuration, instrumentation is maintained globally and per thread. No individual events are collected in the current-events or event-history tables.

Additional setup elements checked, relative to the preceding configuration:

- Table setup_consumers, consumers events_*xxx*_current, where *xxx* is waits, stages, statements, transactions
- Table setup_actors

- Column threads.instrumented

Additional output tables maintained, relative to the preceding configuration:

- events_*xxx*_summary_by_*yyy*_by_event_name, where *xxx* is waits, stages, statements, transactions; and *yyy* is thread, user, host, account

Global, Thread, and Current-Event Instrumentation

Server configuration state:

```
mysql> SELECT * FROM setup_consumers;
+----------------------------------+---------+
| NAME                             | ENABLED |
+----------------------------------+---------+
| global_instrumentation           | YES     |
| thread_instrumentation           | YES     |
| events_waits_current             | YES     |
| events_waits_history             | NO      |
| events_waits_history_long        | NO      |
| events_stages_current            | YES     |
| events_stages_history            | NO      |
| events_stages_history_long       | NO      |
| events_statements_current        | YES     |
| events_statements_history        | NO      |
| events_statements_history_long   | NO      |
| events_transactions_current      | YES     |
| events_transactions_history      | NO      |
| events_transactions_history_long | NO      |
...
+----------------------------------+---------+
```

In this configuration, instrumentation is maintained globally and per thread. Individual events are collected in the current-events table, but not in the event-history tables.

Additional setup elements checked, relative to the preceding configuration:

- Consumers events_*xxx*_history, where *xxx* is waits, stages, statements, transactions
- Consumers events_*xxx*_history_long, where *xxx* is waits, stages, statements, transactions

Additional output tables maintained, relative to the preceding configuration:

- events_*xxx*_current, where *xxx* is waits, stages, statements, transactions

Global, Thread, Current-Event, and Event-History instrumentation

The preceding configuration collects no event history because the events_*xxx*_history and events_*xxx*_history_long consumers are disabled. Those consumers can be enabled separately or together to collect event history per thread, globally, or both.

This configuration collects event history per thread, but not globally:

```
mysql> SELECT * FROM setup_consumers;
+----------------------------------+---------+
| NAME                             | ENABLED |
+----------------------------------+---------+
| global_instrumentation           | YES     |
| thread_instrumentation           | YES     |
| events_waits_current             | YES     |
| events_waits_history             | YES     |
| events_waits_history_long        | NO      |
| events_stages_current            | YES     |
| events_stages_history            | YES     |
| events_stages_history_long       | NO      |
| events_statements_current        | YES     |
| events_statements_history        | YES     |
| events_statements_history_long   | NO      |
| events_transactions_current      | YES     |
| events_transactions_history      | YES     |
| events_transactions_history_long | NO      |
...
+----------------------------------+---------+
```

Event-history tables maintained for this configuration:

- `events_xxx_history`, where *`xxx`* is `waits`, `stages`, `statements`, `transactions`

This configuration collects event history globally, but not per thread:

```
mysql> SELECT * FROM setup_consumers;
+----------------------------------+---------+
| NAME                             | ENABLED |
+----------------------------------+---------+
| global_instrumentation           | YES     |
| thread_instrumentation           | YES     |
| events_waits_current             | YES     |
| events_waits_history             | NO      |
| events_waits_history_long        | YES     |
| events_stages_current            | YES     |
| events_stages_history            | NO      |
| events_stages_history_long       | YES     |
| events_statements_current        | YES     |
| events_statements_history        | NO      |
| events_statements_history_long   | YES     |
| events_transactions_current      | YES     |
| events_transactions_history      | NO      |
| events_transactions_history_long | YES     |
...
+----------------------------------+---------+
```

Event-history tables maintained for this configuration:

- `events_xxx_history_long`, where *`xxx`* is `waits`, `stages`, `statements`, `transactions`

This configuration collects event history per thread and globally:

```
mysql> SELECT * FROM setup_consumers;
+----------------------------------+---------+
| NAME                             | ENABLED |
+----------------------------------+---------+
| global_instrumentation           | YES     |
| thread_instrumentation           | YES     |
| events_waits_current             | YES     |
| events_waits_history             | YES     |
| events_waits_history_long        | YES     |
| events_stages_current            | YES     |
| events_stages_history            | YES     |
| events_stages_history_long       | YES     |
| events_statements_current        | YES     |
| events_statements_history        | YES     |
| events_statements_history_long   | YES     |
| events_transactions_current      | YES     |
| events_transactions_history      | YES     |
| events_transactions_history_long | YES     |
...
+----------------------------------+---------+
```

Event-history tables maintained for this configuration:

- `events_xxx_history`, where *`xxx`* is `waits`, `stages`, `statements`, `transactions`
- `events_xxx_history_long`, where *`xxx`* is `waits`, `stages`, `statements`, `transactions`

20.2.3.4 Naming Instruments or Consumers for Filtering Operations

Names given for filtering operations can be as specific or general as required. To indicate a single instrument or consumer, specify its name in full:

```
mysql> UPDATE setup_instruments
    -> SET ENABLED = 'NO'
    -> WHERE NAME = 'wait/synch/mutex/myisammrg/MYRG_INFO::mutex';

mysql> UPDATE setup_consumers
    -> SET ENABLED = 'NO' WHERE NAME = 'events_waits_current';
```

To specify a group of instruments or consumers, use a pattern that matches the group members:

```
mysql> UPDATE setup_instruments
    -> SET ENABLED = 'NO'
    -> WHERE NAME LIKE 'wait/synch/mutex/%';

mysql> UPDATE setup_consumers
    -> SET ENABLED = 'NO' WHERE NAME LIKE '%history%';
```

If you use a pattern, it should be chosen so that it matches all the items of interest and no others. For example, to select all file I/O instruments, it is better to use a pattern that includes the entire instrument name prefix:

```
... WHERE NAME LIKE 'wait/io/file/%';
```

A pattern of `'%/file/%'` will match other instruments that have a component of `'/file/'` anywhere in the name. Even less suitable is the pattern `'%file%'` because it will match instruments with `'file'` anywhere in the name, such as `wait/synch/mutex/sql/LOCK_des_key_file`.

To check which instrument or consumer names a pattern matches, perform a simple test:

```
mysql> SELECT NAME FROM setup_instruments WHERE NAME LIKE 'pattern';

mysql> SELECT NAME FROM setup_consumers WHERE NAME LIKE 'pattern';
```

For information about the types of names that are supported, see Section 20.4, “Performance Schema Instrument Naming Conventions”.

20.2.3.5 Determining What Is Instrumented

It is always possible to determine what instruments the Performance Schema includes by checking the `setup_instruments` table. For example, to see what file-related events are instrumented for the `InnoDB` storage engine, use this query:

```
mysql> SELECT * FROM setup_instruments WHERE NAME LIKE 'wait/io/file/innodb/%';
+--------------------------------------+---------+-------+
| NAME                                 | ENABLED | TIMED |
+--------------------------------------+---------+-------+
| wait/io/file/innodb/innodb_data_file | YES     | YES   |
| wait/io/file/innodb/innodb_log_file  | YES     | YES   |
| wait/io/file/innodb/innodb_temp_file | YES     | YES   |
+--------------------------------------+---------+-------+
```

An exhaustive description of precisely what is instrumented is not given in this documentation, for several reasons:

- What is instrumented is the server code. Changes to this code occur often, which also affects the set of instruments.
- It is not practical to list all the instruments because there are hundreds of them.
- As described earlier, it is possible to find out by querying the `setup_instruments` table. This information is always up to date for your version of MySQL, also includes instrumentation for instrumented plugins you might have installed that are not part of the core server, and can be used by automated tools.

20.3 Performance Schema Queries

Pre-filtering limits which event information is collected and is independent of any particular user. By contrast, post-filtering is performed by individual users through the use of queries with appropriate `WHERE` clauses that restrict what event information to select from the events available after pre-filtering has been applied.

In Section 20.2.3.3, “Event Pre-Filtering”, an example showed how to pre-filter for file instruments. If the event tables contain both file and nonfile information, post-filtering is another way to see information only for file events. Add a `WHERE` clause to queries to restrict event selection appropriately:

```
mysql> SELECT THREAD_ID, NUMBER_OF_BYTES
    -> FROM events_waits_history
    -> WHERE EVENT_NAME LIKE 'wait/io/file/%'
    -> AND NUMBER_OF_BYTES IS NOT NULL;
+-----------+-----------------+
| THREAD_ID | NUMBER_OF_BYTES |
+-----------+-----------------+
|        11 |              66 |
|        11 |              47 |
|        11 |             139 |
|         5 |              24 |
|         5 |             834 |
+-----------+-----------------+
```

20.4 Performance Schema Instrument Naming Conventions

An instrument name consists of a sequence of components separated by `'/'` characters. Example names:

```
wait/io/file/myisam/log
wait/io/file/mysys/charset
wait/lock/table/sql/handler
wait/synch/cond/mysys/COND_alarm
wait/synch/cond/sql/BINLOG::update_cond
wait/synch/mutex/mysys/BITMAP_mutex
wait/synch/mutex/sql/LOCK_delete
wait/synch/rwlock/sql/Query_cache_query::lock
stage/sql/closing tables
stage/sql/Sorting result
statement/com/Execute
statement/com/Query
statement/sql/create_table
```

```
statement/sql/lock_tables
```

The instrument name space has a tree-like structure. The components of an instrument name from left to right provide a progression from more general to more specific. The number of components a name has depends on the type of instrument.

The interpretation of a given component in a name depends on the components to the left of it. For example, `myisam` appears in both of the following names, but `myisam` in the first name is related to file I/O, whereas in the second it is related to a synchronization instrument:

```

wait/io/file/myisam/log
wait/synch/cond/myisam/MI_SORT_INFO::cond
```

Instrument names consist of a prefix with a structure defined by the Performance Schema implementation and a suffix defined by the developer implementing the instrument code. The top-level component of an instrument prefix indicates the type of instrument. This component also determines which event timer in the `setup_timers` table applies to the instrument. For the prefix part of instrument names, the top level indicates the type of instrument.

The suffix part of instrument names comes from the code for the instruments themselves. Suffixes may include levels such as these:

- A name for the major component (a server module such as `myisam`, `innodb`, `mysys`, or `sql`) or a plugin name.
- The name of a variable in the code, in the form *`XXX`* (a global variable) or *`CCC`*`::`*`MMM`* (a member *`MMM`* in class *`CCC`*). Examples: `COND_thread_cache`, `THR_LOCK_myisam`, `BINLOG::LOCK_index`.

Top-Level Instrument Components

- `idle`: An instrumented idle event. This instrument has no further components.
- `memory`: An instrumented memory event.
- `stage`: An instrumented stage event.
- `statement`: An instrumented statement event.
- `transaction`: An instrumented transaction event. This instrument has no further components.
- `wait`: An instrumented wait event.

Idle Instrument Components

- `idle`

  The idle instrument. The Performance Schema generates idle events as discussed in the description of the `socket_instances.STATE` column in Section 20.9.3.5, “The `socket_instances` Table”.

Memory Instrument Components

Memory instrumentation is disabled by default, and can be enabled or disabled dynamically by updating the `ENABLED` column of the relevant instruments in the `setup_instruments` table. Memory instruments have names of the form `memory/`*`code_area`*`/`*`instrument_name`* where *`code_area`* is a value such as `sql` or `myisam`, and *`instrument_name`* is the instrument detail.

Stage Instrument Components

Stage instruments have names of the form `stage/`*`code_area`*`/`*`stage_name`*, where *`code_area`* is a value such as `sql` or `myisam`, and *`stage_name`* indicates the stage of statement processing, such as `Sorting result` or `Sending data`. Stages correspond to the thread states displayed by `SHOW PROCESSLIST` or that are visible in the `INFORMATION_SCHEMA.PROCESSLIST` table.

Statement Instrument Components

- `statement/abstract/*`: An abstract instrument for statement operations. Abstract instruments are used during the early stages of statement classification before the exact statement type is known, then changed to a more specific statement instrument when the type is known. For a description of this process, see Section 20.9.6, “Performance Schema Statement Event Tables”.
- `statement/com`: An instrumented command operation. These have names corresponding to `COM_xxx` operations (see the `mysql_com.h` header file and `sql/sql_parse.cc`. For example, the `statement/com/Connect` and `statement/com/Init DB` instruments correspond to the `COM_CONNECT` and `COM_INIT_DB` commands.
- `statement/scheduler/event`: A single instrument to track all events executed by the Event Scheduler. This instrument comes into play when a scheduled event begins executing.
- `statement/sp`: An instrumented internal instruction executed by a stored program. For example, the `statement/sp/cfetch` and `statement/sp/freturn` instruments are used cursor fetch and function return instructions.
- `statement/sql`: An instrumented SQL statement operation. For example, the `statement/sql/create_db` and `statement/sql/select` instruments are used for `CREATE DATABASE` and `SELECT` statements.

Wait Instrument Components

- `wait/io`

  An instrumented I/O operation.

  - `wait/io/file`

    An instrumented file I/O operation. For files, the wait is the time waiting for the file operation to complete (for example, a call to `fwrite()`). Due to caching, the physical file I/O on the disk might not happen within this call.

  - `wait/io/socket`

    An instrumented socket operation. Socket instruments have names of the form `wait/io/socket/sql/socket_type`. The server has a listening socket for each network protocol that it supports. The instruments associated with listening sockets for TCP/IP or Unix socket file connections have a `socket_type` value of `server_tcpip_socket` or `server_unix_socket`, respectively. When a listening socket detects a connection, the server transfers the connection to a new socket managed by a separate thread. The instrument for the new connection thread has a `socket_type` value of `client_connection`.

  - `wait/io/table`

    An instrumented table I/O operation. These include row-level accesses to persistent base tables or temporary tables. Operations that affect rows are fetch, insert, update, and delete. For a view, waits are associated with base tables referenced by the view.

    Unlike most waits, a table I/O wait can include other waits. For example, table I/O might include file I/O or memory operations. Thus, `events_waits_current` for a table I/O wait usually has two rows. For more information, see Section 20.6, “Performance Schema Atom and Molecule Events”.

Some row operations might cause multiple table I/O waits. For example, an insert might activate a trigger that causes an update.

- `wait/lock`

  An instrumented lock operation.

  - `wait/lock/table`

    An instrumented table lock operation.

  - `wait/lock/metadata/sql/mdl`

    An instrumented metadata lock operation (disabled by default).

- `wait/synch`

  An instrumented synchronization object. For synchronization objects, the `TIMER_WAIT` time includes the amount of time blocked while attempting to acquire a lock on the object, if any.

  - `wait/synch/cond`

    A condition is used by one thread to signal to other threads that something they were waiting for has happened. If a single thread was waiting for a condition, it can wake up and proceed with its execution. If several threads were waiting, they can all wake up and compete for the resource for which they were waiting.

  - `wait/synch/mutex`

    A mutual exclusion object used to permit access to a resource (such as a section of executable code) while preventing other threads from accessing the resource.

  - `wait/synch/rwlock`

    A read/write lock object used to lock a specific variable for access while preventing its use by other threads. A shared read lock can be acquired simultaneously by multiple threads. An exclusive write lock can be acquired by only one thread at a time.

20.5 Performance Schema Status Monitoring

There are several status variables associated with the Performance Schema:

```
mysql> SHOW STATUS LIKE 'perf%';
+-----------------------------------------------+-------+
| Variable_name                                 | Value |
+-----------------------------------------------+-------+
| Performance_schema_accounts_lost              | 0     |
| Performance_schema_cond_classes_lost          | 0     |
| Performance_schema_cond_instances_lost        | 0     |
| Performance_schema_digest_lost                | 0     |
| Performance_schema_file_classes_lost          | 0     |
| Performance_schema_file_handles_lost          | 0     |
| Performance_schema_file_instances_lost        | 0     |
| Performance_schema_hosts_lost                 | 0     |
| Performance_schema_locker_lost                | 0     |
| Performance_schema_memory_classes_lost        | 0     |
| Performance_schema_metadata_lock_lost         | 0     |
| Performance_schema_mutex_classes_lost         | 0     |
| Performance_schema_mutex_instances_lost       | 0     |
| Performance_schema_nested_statement_lost      | 0     |
| Performance_schema_program_lost               | 0     |
| Performance_schema_rwlock_classes_lost        | 0     |
| Performance_schema_rwlock_instances_lost      | 0     |
| Performance_schema_session_connect_attrs_lost | 0     |
| Performance_schema_socket_classes_lost        | 0     |
| Performance_schema_socket_instances_lost      | 0     |
| Performance_schema_stage_classes_lost         | 0     |
| Performance_schema_statement_classes_lost     | 0     |
| Performance_schema_table_handles_lost         | 0     |
| Performance_schema_table_instances_lost       | 0     |
| Performance_schema_thread_classes_lost        | 0     |
| Performance_schema_thread_instances_lost      | 0     |
| Performance_schema_users_lost                 | 0     |
+-----------------------------------------------+-------+
```

The Performance Schema status variables provide information about instrumentation that could not be loaded or created due to memory constraints. Names for these variables have several forms:

- `Performance_schema_xxx_classes_lost` indicates how many instruments of type *xxx* could not be loaded.

- `Performance_schema_xxx_instances_lost` indicates how many instances of object type *xxx* could not be created.

- `Performance_schema_xxx_handles_lost` indicates how many instances of object type *xxx* could not be opened.

- `Performance_schema_locker_lost` indicates how many events are “lost” or not recorded.

For example, if a mutex is instrumented in the server source but the server cannot allocate memory for the instrumentation at runtime, it increments `Performance_schema_mutex_classes_lost`. The mutex still functions as a synchronization object (that is, the server continues to function normally), but performance data for it will not be collected. If the instrument can be allocated, it can be used for initializing instrumented mutex instances. For a singleton mutex such as a global mutex, there will be only one instance. Other mutexes have an instance per connection, or per page in various caches and data buffers, so the number of instances varies over time. Increasing the maximum number of connections or the maximum size of some buffers will increase the maximum number of instances that might be allocated at once. If the server cannot create a given instrumented mutex instance, it increments `Performance_schema_mutex_instances_lost`.

Suppose that the following conditions hold:

- The server was started with the `--performance_schema_max_mutex_classes=200` option and thus has room for 200 mutex instruments.

- 150 mutex instruments have been loaded already.

- The plugin named `plugin_a` contains 40 mutex instruments.

- The plugin named `plugin_b` contains 20 mutex instruments.

The server allocates mutex instruments for the plugins depending on how many they need and how many are available, as illustrated by the following sequence of statements:

```
INSTALL PLUGIN plugin_a
```

The server now has 150+40 = 190 mutex instruments.

```
UNINSTALL PLUGIN plugin_a;
```

The server still has 190 instruments. All the historical data generated by the plugin code is still available, but new events for the instruments are not collected.

```
INSTALL PLUGIN plugin_a;
```

The server detects that the 40 instruments are already defined, so no new instruments are created, and previously assigned internal memory buffers are reused. The server still has 190 instruments.

```
INSTALL PLUGIN plugin_b;
```

The server has room for 200-190 = 10 instruments (in this case, mutex classes), and sees that the plugin contains 20 new instruments. 10 instruments are loaded, and 10 are discarded or “lost.” The `Performance_schema_mutex_classes_lost` indicates the number of instruments (mutex classes) lost:

```
mysql> SHOW STATUS LIKE "perf%mutex_classes_lost";
+---------------------------------------+-------+
| Variable_name                         | Value |
+---------------------------------------+-------+
| Performance_schema_mutex_classes_lost | 10    |
+---------------------------------------+-------+
1 row in set (0.10 sec)
```

The instrumentation still works and collects (partial) data for `plugin_b`.

When the server cannot create a mutex instrument, these results occur:

- No row for the instrument is inserted into the `setup_instruments` table.
- `Performance_schema_mutex_classes_lost` increases by 1.
- `Performance_schema_mutex_instances_lost` does not change. (When the mutex instrument is not created, it cannot be used to create instrumented mutex instances later.)

The pattern just described applies to all types of instruments, not just mutexes.

A value of `Performance_schema_mutex_classes_lost` greater than 0 can happen in two cases:

- To save a few bytes of memory, you start the server with `--performance_schema_max_mutex_classes=N`, where *N* is less than the default value. The default value is chosen to be sufficient to load all the plugins provided in the MySQL distribution, but this can be reduced if some plugins are never loaded. For example, you might choose not to load some of the storage engines in the distribution.
- You load a third-party plugin that is instrumented for the Performance Schema but do not allow for the plugin's instrumentation memory requirements when you start the server. Because it comes from a third party, the instrument memory consumption of this engine is not accounted for in the default value chosen for `performance_schema_max_mutex_classes`.

  If the server has insufficient resources for the plugin's instruments and you do not explicitly allocate more using `--performance_schema_max_mutex_classes=N`, loading the plugin leads to starvation of instruments.

If the value chosen for `performance_schema_max_mutex_classes` is too small, no error is reported in the error log and there is no failure at runtime. However, the content of the tables in the performance_schema database will miss events. The Performance_schema_mutex_classes_lost status variable is the only visible sign to indicate that some events were dropped internally due to failure to create instruments.

If an instrument is not lost, it is known to the Performance Schema, and is used when instrumenting instances. For example, wait/synch/mutex/sql/LOCK_delete is the name of a mutex instrument in the setup_instruments table. This single instrument is used when creating a mutex in the code (in THD::LOCK_delete) however many instances of the mutex are needed as the server runs. In this case, LOCK_delete is a mutex that is per connection (THD), so if a server has 1000 connections, there are 1000 threads, and 1000 instrumented LOCK_delete mutex instances (THD::LOCK_delete).

If the server does not have room for all these 1000 instrumented mutexes (instances), some mutexes are created with instrumentation, and some are created without instrumentation. If the server can create only 800 instances, 200 instances are lost. The server continues to run, but increments Performance_schema_mutex_instances_lost by 200 to indicate that instances could not be created.

A value of Performance_schema_mutex_instances_lost greater than 0 can happen when the code initializes more mutexes at runtime than were allocated for --performance_schema_max_mutex_instances=*N*.

The bottom line is that if SHOW STATUS LIKE 'perf%' says that nothing was lost (all values are zero), the Performance Schema data is accurate and can be relied upon. If something was lost, the data is incomplete, and the Performance Schema could not record everything given the insufficient amount of memory it was given to use. In this case, the specific Performance_schema_*xxx*_lost variable indicates the problem area.

It might be appropriate in some cases to cause deliberate instrument starvation. For example, if you do not care about performance data for file I/O, you can start the server with all Performance Schema parameters related to file I/O set to 0. No memory will be allocated for file-related classes, instances, or handles, and all file events will be lost.

Use SHOW ENGINE PERFORMANCE_SCHEMA STATUS to inspect the internal operation of the Performance Schema code:

```
mysql> SHOW ENGINE PERFORMANCE_SCHEMA STATUS\G
...
*************************** 3. row ***************************
  Type: performance_schema
  Name: events_waits_history.size
Status: 76
*************************** 4. row ***************************
  Type: performance_schema
  Name: events_waits_history.count
Status: 10000
*************************** 5. row ***************************
  Type: performance_schema
  Name: events_waits_history.memory
Status: 760000
...
*************************** 57. row ***************************
  Type: performance_schema
  Name: performance_schema.memory
Status: 26459600
...
```

This statement is intended to help the DBA understand the effects that different Performance Schema options have on memory requirements. For a description of the field meanings, see Section 13.7.5.14, “SHOW ENGINE Syntax”.

20.6 Performance Schema Atom and Molecule Events

For a table I/O event, there are usually two rows in events_waits_current, not one. For example, a row fetch might result in rows like this:

```
Row# EVENT_NAME                 TIMER_START TIMER_END
---- ----------                 ----------- ---------
   1 wait/io/file/myisam/dfile        10001 10002
   2 wait/io/table/sql/handler        10000 NULL
```

The row fetch causes a file read. In the example, the table I/O fetch event started before the file I/O event but has not finished (its TIMER_END value is NULL). The file I/O event is “nested” within the table I/O event.

This occurs because, unlike other “atomic” wait events such as for mutexes or file I/O, table I/O events are “molecular” and include (overlap with) other events. In events_waits_current, the table I/O event usually has two rows:

- One row for the most recent table I/O wait event

- One row for the most recent wait event of any kind

Usually, but not always, the “of any kind” wait event differs from the table I/O event. As each subsidiary event completes, it disappears from events_waits_current. At this point, and until the next subsidiary event begins, the table I/O wait is also the most recent wait of any kind.

20.7 Performance Schema Statement Digests

The Performance Schema maintains statement digest information. Digesting converts a SQL statement to normalized form and computes a hash value for the result. Normalization permits statements that are similar to be grouped and summarized to expose information about the types of statements the server is executing and how often they occur. This section describes how statement normalizing occurs and how it can be useful.

Statement digesting involves these Performance Schema components:

- A statement_digest consumer in the setup_consumers table controls whether the Performance Schema maintains digest information.

- The statement event tables (events_statements_current, events_statements_history, and events_statements_history_long) have DIGEST and DIGEST_TEXT columns that contain digest MD5 values and the corresponding normalized statement text strings.

- A events_statements_summary_by_digest table provides aggregated statement digest information.

Normalizing a statement transforms the statement text to a more standardized string representation that preserves the general statement structure while removing information not essential to the structure. Object identifiers such as database and table names are preserved. Values and comments are removed, and whitespace is adjusted. The Performance Schema does not retain information such as names, passwords, dates, and so forth.

Consider these statements:

```
SELECT * FROM orders WHERE customer_id=10 AND quantity>20
SELECT * FROM orders WHERE customer_id = 20 AND quantity > 100
```

To normalize these statements, the Performance Schema replaces data values by ? and adjusts whitespace. Both statements yield the same normalized form and thus are considered “the same”:

```
SELECT * FROM orders WHERE customer_id = ? AND quantity > ?
```

The normalized statement contains less information but is still representative of the original statement. Other similar statements that have different comparison values have the same normalized form.

Now consider these statements:

```
SELECT * FROM customers WHERE customer_id = 1000
SELECT * FROM orders WHERE customer_id = 1000
```

In this case, the statements are not “the same.” The object identifiers differ, so the statements yield different normalized forms:

```
SELECT * FROM customers WHERE customer_id = ?
SELECT * FROM orders WHERE customer_id = ?
```

Normalized statements have a fixed length. The maximum length of a DIGEST_TEXT value is 1024 bytes. There is no option to change this maximum. If normalization produces a statement that exceeds this length, the text ends with “...”. Long statements that differ only in the part that occurs following the “...” are considered to be the same. Consider these statements:

```
SELECT * FROM mytable WHERE cola = 10 AND colb = 20
SELECT * FROM mytable WHERE cola = 10 AND colc = 20
```

If the cutoff happened to be right after the AND, both statements would have this normalized form:

```
SELECT * FROM mytable WHERE cola = ? AND ...
```

In this case, the difference in the second column name is lost and both statements are considered the same.

For each normalized statement, the Performance Schema computes a hash digest value and stores that value and the statement in the DIGEST and DIGEST_TEXT columns of the statement event tables (events_statements_current, events_statements_history, and events_statements_history_long). In addition, information for statements with the same SCHEMA_NAME and DIGEST values are aggregated in the events_statements_summary_by_digest summary table. The Performance Schema uses MD5 hash values because they are fast to compute and have a favorable statistical distribution that minimizes collisions.

The events_statements_summary_by_digest summary table has a fixed size, so when it becomes full, statements that have SCHEMA_NAME and DIGEST values not matching existing values in the table are grouped in a special row with SCHEMA_NAME and DIGEST set to NULL. This permits all statements to be counted. However, if the special row accounts for a significant percentage of the statements executed, it might be desirable to increase the size of the summary table. To do this, set the performance_schema_digests_size system variable to a larger value at server startup. If no performance_schema_digests_size value is given, the server estimates the value to use at startup.

The statement digest summary table provides a profile of the statements executed by the server. It shows what kinds of statements an application is executing and how often. An application developer can use this information together with other information in the table to assess the application's performance characteristics. For example, table columns that show wait times, lock times, or index use may highlight types of queries that are inefficient. This gives the developer insight into which parts of the application need attention.

20.8 Performance Schema General Table Characteristics

The name of the performance_schema database is lowercase, as are the names of tables within it. Queries should specify the names in lowercase.

Most tables in the performance_schema database are read only and cannot be modified. Some of the setup tables have columns that can be modified to affect Performance Schema operation; some also permit rows to be inserted or deleted. Truncation is permitted to clear collected events, so TRUNCATE TABLE can be used on tables containing those kinds of information, such as tables named with a prefix of events_waits_.

TRUNCATE TABLE can also be used with summary tables, but except for events_statements_summary_by_digest and the memory summary tables, the effect is to reset the summary columns to 0 or NULL, not to remove rows.

Privileges are as for other databases and tables:

- To retrieve from performance_schema tables, you must have the SELECT privilege.
- To change those columns that can be modified, you must have the UPDATE privilege.
- To truncate tables that can be truncated, you must have the DROP privilege.

20.9 Performance Schema Table Descriptions

Tables in the performance_schema database can be grouped as follows:

- Setup tables. These tables are used to configure and display monitoring characteristics.
- Current events tables. The events_waits_current table contains the most recent event for each thread. Other similar tables contain current events at different levels of the event hierarchy: events_stages_current for stage events, events_statements_current for statement events, and events_transactions_current for transaction events.
- History tables. These tables have the same structure as the current events tables, but contain more rows. For example, for wait events, events_waits_history table contains the most recent 10 events per thread. events_waits_history_long contains the most recent 10,000 events. Other similar tables exist for stage, statement, and transaction histories.

  To change the sizes of the history tables, set the appropriate system variables at server startup. For example, to set the sizes of the wait event history tables, set performance_schema_events_waits_history_size and performance_schema_events_waits_history_long_size.
- Summary tables. These tables contain information aggregated over groups of events, including those that have been discarded from the history tables.
- Instance tables. These tables document what types of objects are instrumented. An instrumented object, when used by the server, produces an event. These tables provide event names and explanatory notes or status information.
- Miscellaneous tables. These do not fall into any of the other table groups.

20.9.1 Performance Schema Table Index

The following table lists each Performance Schema table and provides a short description of each one.

Table 20.1 Performance Schema Tables

Table Name	Description
accounts	Connection statistics per client account
cond_instances	synchronization object instances
events_stages_current	Current stage events
events_stages_history	Most recent stage events for each thread
events_stages_history_long	Most recent stage events overall
events_stages_summary_by_account_by_event_name	Stage events per account and event name
events_stages_summary_by_host_by_event_name	Stage events per host name and event name
events_stages_summary_by_thread_by_event_name	Stage waits per thread and event name
events_stages_summary_by_user_by_event_name	Stage events per user name and event name
events_stages_summary_global_by_event_name	Stage waits per event name
events_statements_current	Current statement events
events_statements_history	Most recent statement events for each thread
events_statements_history_long	Most recent statement events overall
events_statements_summary_by_account_by_event_name	Statement events per account and event name
events_statements_summary_by_digest	Statement events per schema and digest value
events_statements_summary_by_host_by_event_name	Statement events per host name and event name
events_statements_summary_by_program	Statement events per stored program
events_statements_summary_by_thread_by_event_name	Statement events per thread and event name
events_statements_summary_by_user_by_event_name	Statement events per user name and event name
events_statements_summary_global_by_event_name	Statement events per event name
events_transactions_current	Current transaction events
events_transactions_history	Most recent transaction events for each thread
events_transactions_history_long	Most recent transaction events overall
events_transactions_summary_by_account_by_event_name	Transaction events per account and event name
events_transactions_summary_by_host_by_event_name	Transaction events per host name and event name
events_transactions_summary_by_thread_by_event_name	Transaction events per thread and event name
events_transactions_summary_by_user_by_event_name	Transaction events per user name and event name
events_transactions_summary_global_by_event_name	Transaction events per event name
events_waits_current	Current wait events
events_waits_history	Most recent wait events for each thread
events_waits_history_long	Most recent wait events overall
events_waits_summary_by_account_by_event_name	Wait events per account and event name
events_waits_summary_by_host_by_event_name	Wait events per host name and event name

Table Name	Description
events_waits_summary_by_instance	Wait events per instance
events_waits_summary_by_thread_by_event_name	Wait events per thread and event name
events_waits_summary_by_user_by_event_name	Wait events per user name and event name
events_waits_summary_global_by_event_name	Wait events per event name
file_instances	File instances
file_summary_by_event_name	File events per event name
file_summary_by_instance	File events per file instance
host_cache	Information from the internal host cache
hosts	Connection statistics per client host name
memory_summary_by_account_by_event_name	Memory operations per account and event name
memory_summary_by_host_by_event_name	Memory operations per host and event name
memory_summary_by_thread_by_event_name	Memory operations per thread and event name
memory_summary_by_user_by_event_name	Memory operations per user and event name
memory_summary_global_by_event_name	Memory operations globally per event name
metadata_locks	Metadata locks and lock requests
mutex_instances	Mutex synchronization object instances
objects_summary_global_by_type	Object summaries
performance_timers	Which event timers are available
prepared_statements_instances	Prepared statement instances and statistics
replication_connection_configuration	Configuration parameters for connecting to the master
replication_connection_status	Current status of the connection to the master
replication_execute_configuration	Configuration parameters for transaction execution on the slave
replication_execute_status	Current transaction execution status on the slave
replication_execute_status_by_coordinator	SQL or coordinator thread execution status
replication_execute_status_by_worker	Worker thread execution status (empty unless slave is multi-threaded)
rwlock_instances	Lock synchronization object instances
session_account_connect_attrs	Connection attributes per for the current session
session_connect_attrs	Connection attributes for all sessions
setup_actors	How to initialize monitoring for new foreground threads
setup_consumers	Consumers for which event information can be stored
setup_instruments	Classes of instrumented objects for which events can be collected

Table Name	Description
setup_objects	Which objects should be monitored
setup_timers	Current event timer
socket_instances	Active connection instances
socket_summary_by_event_name	Socket waits and I/O per event name
socket_summary_by_instance	Socket waits and I/O per instance
table_handles	Table locks and lock requests
table_io_waits_summary_by_index_usage	Table I/O waits per index
table_io_waits_summary_by_table	Table I/O waits per table
table_lock_waits_summary_by_table	Table lock waits per table
threads	Information about server threads
users	Connection statistics per client user name

20.9.2 Performance Schema Setup Tables

The setup tables provide information about the current instrumentation and enable the monitoring configuration to be changed. For this reason, some columns in these tables can be changed if you have the UPDATE privilege.

The use of tables rather than individual variables for setup information provides a high degree of flexibility in modifying Performance Schema configuration. For example, you can use a single statement with standard SQL syntax to make multiple simultaneous configuration changes.

These setup tables are available:

- setup_actors: How to initialize monitoring for new foreground threads
- setup_consumers: The destinations to which event information can be sent and stored
- setup_instruments: The classes of instrumented objects for which events can be collected
- setup_objects: Which objects should be monitored
- setup_timers: The current event timer

20.9.2.1 The setup_actors Table

The setup_actors table contains information that determines whether to enable monitoring for new foreground server threads; that is, threads associated with client connections. This table has a maximum size of 100 rows by default. To change the table size, modify the performance_schema_setup_actors_size system variable at server startup.

For each new foreground thread, the Performance Schema matches the user and host for the the thread against the rows of the setup_actors table. Based on whether any row matches, the INSTRUMENTED column of the threads table row for the thread is set to YES or NO. This enables instrumenting to be applied selectively per host, user, or combination of host and user.

The initial contents of the setup_actors table match any user and host combination, so monitoring for all foreground threads is enabled by default:

```
mysql> SELECT * FROM setup_actors;
+------+------+------+
| HOST | USER | ROLE |
+------+------+------+
| %    | %    | %    |
+------+------+------+
```

Modifications to the setup_actors table do not affect existing threads.

For information about how to use the setup_actors table in event monitoring, see Pre-Filtering by Thread.

The setup_actors table has these columns:

- HOST

  The host name. This should be a literal name, or '%' to mean “any host.”

- USER

  The user name. This should be a literal name, or '%' to mean “any user.”

- ROLE

  Unused.

20.9.2.2 The setup_consumers Table

The setup_consumers table lists the types of consumers for which event information can be stored and which are enabled:

```
mysql> SELECT * FROM setup_consumers;
+----------------------------------+---------+
| NAME                             | ENABLED |
+----------------------------------+---------+
| events_stages_current            | NO      |
| events_stages_history            | NO      |
| events_stages_history_long       | NO      |
| events_statements_current        | YES     |
| events_statements_history        | NO      |
| events_statements_history_long   | NO      |
| events_transactions_current      | YES     |
| events_transactions_history      | NO      |
| events_transactions_history_long | NO      |
| events_waits_current             | NO      |
| events_waits_history             | NO      |
| events_waits_history_long        | NO      |
| global_instrumentation           | YES     |
| thread_instrumentation           | YES     |
| statements_digest                | YES     |
+----------------------------------+---------+
```

The consumer settings in the setup_consumers table form a hierarchy from higher levels to lower. For detailed information about the effect of enabling different consumers, see Pre-Filtering by Consumer.

Modifications to the setup_consumers table affect monitoring immediately.

The setup_consumers table has these columns:

- NAME

  The consumer name.

- ENABLED

Whether the consumer is enabled. This column can be modified. If you disable a consumer, the server does not spend time adding event information to it.

20.9.2.3 The setup_instruments Table

The setup_instruments table lists classes of instrumented objects for which events can be collected:

```
mysql> SELECT * FROM setup_instruments;
+------------------------------------------------------------+---------+-------+
| NAME                                                       | ENABLED | TIMED |
+------------------------------------------------------------+---------+-------+
...
| wait/synch/mutex/sql/LOCK_global_read_lock                 | YES     | YES   |
| wait/synch/mutex/sql/LOCK_global_system_variables          | YES     | YES   |
| wait/synch/mutex/sql/LOCK_lock_db                          | YES     | YES   |
| wait/synch/mutex/sql/LOCK_manager                          | YES     | YES   |
...
| wait/synch/rwlock/sql/LOCK_grant                           | YES     | YES   |
| wait/synch/rwlock/sql/LOGGER::LOCK_logger                  | YES     | YES   |
| wait/synch/rwlock/sql/LOCK_sys_init_connect                | YES     | YES   |
| wait/synch/rwlock/sql/LOCK_sys_init_slave                  | YES     | YES   |
...
| wait/io/file/sql/binlog                                    | YES     | YES   |
| wait/io/file/sql/binlog_index                              | YES     | YES   |
| wait/io/file/sql/casetest                                  | YES     | YES   |
| wait/io/file/sql/dbopt                                     | YES     | YES   |
...
```

Each instrument added to the source code provides a row for this table, even when the instrumented code is not executed. When an instrument is enabled and executed, instrumented instances are created, which are visible in the *_instances tables.

Modifications to the setup_instruments table affect monitoring immediately.

For more information about the role of the setup_instruments table in event filtering, see Section 20.2.3.3, “Event Pre-Filtering”.

The setup_instruments table has these columns:

- NAME

  The instrument name. Instrument names may have multiple parts and form a hierarchy, as discussed in Section 20.4, “Performance Schema Instrument Naming Conventions”. Events produced from execution of an instrument have an EVENT_NAME value that is taken from the instrument NAME value. (Events do not really have a “name,” but this provides a way to associate events with instruments.)

- ENABLED

  Whether the instrument is enabled. This column can be modified. A disabled instrument produces no events.

- TIMED

  Whether the instrument is timed. This column can be modified.

  For memory instruments, the TIMED column in setup_instruments is ignored because memory operations are not timed.

  If an enabled instrument is not timed, the instrument code is enabled, but the timer is not. Events produced by the instrument have NULL for the TIMER_START, TIMER_END, and TIMER_WAIT timer values. This in turn causes those values to be ignored when calculating the sum, minimum, maximum, and average time values in summary tables.

20.9.2.4 The setup_objects Table

The setup_objects table controls whether the Performance Schema monitors particular objects. This table has a maximum size of 100 rows by default. To change the table size, modify the performance_schema_setup_objects_size system variable at server startup.

The initial setup_objects contents look like this:

```
mysql> SELECT * FROM setup_objects;
+-------------+--------------------+-------------+---------+-------+
| OBJECT_TYPE | OBJECT_SCHEMA      | OBJECT_NAME | ENABLED | TIMED |
+-------------+--------------------+-------------+---------+-------+
| EVENT       | mysql              | %           | NO      | NO    |
| EVENT       | performance_schema | %           | NO      | NO    |
| EVENT       | information_schema | %           | NO      | NO    |
| EVENT       | %                  | %           | YES     | YES   |
| FUNCTION    | mysql              | %           | NO      | NO    |
| FUNCTION    | performance_schema | %           | NO      | NO    |
| FUNCTION    | information_schema | %           | NO      | NO    |
| FUNCTION    | %                  | %           | YES     | YES   |
| PROCEDURE   | mysql              | %           | NO      | NO    |
| PROCEDURE   | performance_schema | %           | NO      | NO    |
| PROCEDURE   | information_schema | %           | NO      | NO    |
| PROCEDURE   | %                  | %           | YES     | YES   |
| TABLE       | mysql              | %           | NO      | NO    |
| TABLE       | performance_schema | %           | NO      | NO    |
| TABLE       | information_schema | %           | NO      | NO    |
| TABLE       | %                  | %           | YES     | YES   |
| TRIGGER     | mysql              | %           | NO      | NO    |
| TRIGGER     | performance_schema | %           | NO      | NO    |
| TRIGGER     | information_schema | %           | NO      | NO    |
| TRIGGER     | %                  | %           | YES     | YES   |
+-------------+--------------------+-------------+---------+-------+
```

Modifications to the setup_objects table affect object monitoring immediately.

For object types listed in setup_objects, the Performance Schema uses the table to how to monitor them. Object matching is based on the OBJECT_SCHEMA and OBJECT_NAME columns. Objects for which there is no match are not monitored.

The effect of the default object configuration is to instrument all tables except those in the mysql, INFORMATION_SCHEMA, and performance_schema databases. (Tables in the INFORMATION_SCHEMA database are not instrumented regardless of the contents of setup_objects; the row for information_schema.% simply makes this default explicit.)

When the Performance Schema checks for a match in setup_objects, it tries to find more specific matches first. For example, with a table db1.t1, it looks for a match for 'db1' and 't1', then for 'db1' and '%', then for '%' and '%'. The order in which matching occurs matters because different matching setup_objects rows can have different ENABLED and TIMED values.

Rows can be inserted into or deleted from setup_objects by users with the INSERT or DELETE privilege on the table. For existing rows, only the ENABLED and TIMED columns can be modified, by users with the UPDATE privilege on the table.

For more information about the role of the setup_objects table in event filtering, see Section 20.2.3.3, “Event Pre-Filtering”.

The setup_objects table has these columns:

- OBJECT_TYPE

  The type of object to instrument. The value is one of 'EVENT' (Event Scheduler event), 'FUNCTION' (stored function), 'PROCEDURE' (stored procedure), 'TABLE' (base table), or 'TRIGGER' (trigger). Before MySQL 5.7.2, the value is always 'TABLE'.

  TABLE filtering affects table I/O events (wait/io/table/sql/handler instrument) and table lock events (wait/lock/table/sql/handler instrument).

- OBJECT_SCHEMA

  The schema that contains the object. This should be a literal name, or '%' to mean “any schema.”

- OBJECT_NAME

  The name of the instrumented object. This should be a literal name, or '%' to mean “any object.”

- ENABLED

  Whether events for the object are instrumented. This column can be modified.

- TIMED

  Whether events for the object are timed. This column can be modified.

20.9.2.5 The setup_timers Table

The setup_timers table shows the currently selected event timers:

```
mysql> SELECT * FROM setup_timers;
+-------------+-------------+
| NAME        | TIMER_NAME  |
+-------------+-------------+
| idle        | MICROSECOND |
| wait        | CYCLE       |
| stage       | NANOSECOND  |
| statement   | NANOSECOND  |
| transaction | NANOSECOND  |
+-------------+-------------+
```

The setup_timers.TIMER_NAME value can be changed to select a different timer. The value can be any of the values in the performance_timers.TIMER_NAME column. For an explanation of how event timing occurs, see Section 20.2.3.1, “Performance Schema Event Timing”.

Modifications to the setup_timers table affect monitoring immediately. Events already in progress may use the original timer for the begin time and the new timer for the end time, which may lead to unpredictable results. If you make timer changes, you may want to use TRUNCATE TABLE to reset Performance Schema statistics.

The setup_timers table has these columns:

- NAME

  The type of instrument the timer is used for.

- TIMER_NAME

  The timer that applies to the instrument type. This column can be modified.

20.9.3 Performance Schema Instance Tables

Instance tables document what types of objects are instrumented. They provide event names and explanatory notes or status information:

- `cond_instances`: Condition synchronization object instances
- `file_instances`: File instances
- `mutex_instances`: Mutex synchronization object instances
- `rwlock_instances`: Lock synchronization object instances
- `socket_instances`: Active connection instances

These tables list instrumented synchronization objects, files, and connections. There are three types of synchronization objects: `cond`, `mutex`, and `rwlock`. Each instance table has an `EVENT_NAME` or `NAME` column to indicate the instrument associated with each row. Instrument names may have multiple parts and form a hierarchy, as discussed in Section 20.4, “Performance Schema Instrument Naming Conventions”.

The `mutex_instances.LOCKED_BY_THREAD_ID` and `rwlock_instances.WRITE_LOCKED_BY_THREAD_ID` columns are extremely important for investigating performance bottlenecks or deadlocks. For examples of how to use them for this purpose, see Section 20.15, “Using the Performance Schema to Diagnose Problems”

20.9.3.1 The `cond_instances` Table

The `cond_instances` table lists all the conditions seen by the Performance Schema while the server executes. A condition is a synchronization mechanism used in the code to signal that a specific event has happened, so that a thread waiting for this condition can resume work.

When a thread is waiting for something to happen, the condition name is an indication of what the thread is waiting for, but there is no immediate way to tell which other thread, or threads, will cause the condition to happen.

The `cond_instances` table has these columns:

- `NAME`

  The instrument name associated with the condition.

- `OBJECT_INSTANCE_BEGIN`

  The address in memory of the instrumented condition.

20.9.3.2 The `file_instances` Table

The `file_instances` table lists all the files seen by the Performance Schema when executing file I/O instrumentation. If a file on disk has never been opened, it will not be in `file_instances`. When a file is deleted from the disk, it is also removed from the `file_instances` table.

The `file_instances` table has these columns:

- `FILE_NAME`

  The file name.

- `EVENT_NAME`

The instrument name associated with the file.

- OPEN_COUNT

  The count of open handles on the file. If a file was opened and then closed, it was opened 1 time, but OPEN_COUNT will be 0. To list all the files currently opened by the server, use WHERE OPEN_COUNT > 0.

20.9.3.3 The mutex_instances Table

The mutex_instances table lists all the mutexes seen by the Performance Schema while the server executes. A mutex is a synchronization mechanism used in the code to enforce that only one thread at a given time can have access to some common resource. The resource is said to be “protected” by the mutex.

When two threads executing in the server (for example, two user sessions executing a query simultaneously) do need to access the same resource (a file, a buffer, or some piece of data), these two threads will compete against each other, so that the first query to obtain a lock on the mutex will cause the other query to wait until the first is done and unlocks the mutex.

The work performed while holding a mutex is said to be in a “critical section,” and multiple queries do execute this critical section in a serialized way (one at a time), which is a potential bottleneck.

The mutex_instances table has these columns:

- NAME

  The instrument name associated with the mutex.

- OBJECT_INSTANCE_BEGIN

  The address in memory of the instrumented mutex.

- LOCKED_BY_THREAD_ID

  When a thread currently has a mutex locked, LOCKED_BY_THREAD_ID is the THREAD_ID of the locking thread, otherwise it is NULL.

For every mutex instrumented in the code, the Performance Schema provides the following information.

- The setup_instruments table lists the name of the instrumentation point, with the prefix wait/synch/mutex/.
- When some code creates a mutex, a row is added to the mutex_instances table. The OBJECT_INSTANCE_BEGIN column is a property that uniquely identifies the mutex.
- When a thread attempts to lock a mutex, the events_waits_current table shows a row for that thread, indicating that it is waiting on a mutex (in the EVENT_NAME column), and indicating which mutex is waited on (in the OBJECT_INSTANCE_BEGIN column).
- When a thread succeeds in locking a mutex:
  - events_waits_current shows that the wait on the mutex is completed (in the TIMER_END and TIMER_WAIT columns)
  - The completed wait event is added to the events_waits_history and events_waits_history_long tables

  - mutex_instances shows that the mutex is now owned by the thread (in the THREAD_ID column).

- When a thread unlocks a mutex, mutex_instances shows that the mutex now has no owner (the THREAD_ID column is NULL).
- When a mutex object is destroyed, the corresponding row is removed from mutex_instances.

By performing queries on both of the following tables, a monitoring application or a DBA can detect bottlenecks or deadlocks between threads that involve mutexes:

- events_waits_current, to see what mutex a thread is waiting for
- mutex_instances, to see which other thread currently owns a mutex

20.9.3.4 The rwlock_instances Table

The rwlock_instances table lists all the rwlock instances (read write locks) seen by the Performance Schema while the server executes. An rwlock is a synchronization mechanism used in the code to enforce that threads at a given time can have access to some common resource following certain rules. The resource is said to be “protected” by the rwlock. The access is either shared (many threads can have a read lock at the same time) or exclusive (only one thread can have a write lock at a given time).

Depending on how many threads are requesting a lock, and the nature of the locks requested, access can be either granted in shared mode, granted in exclusive mode, or not granted at all, waiting for other threads to finish first.

The rwlock_instances table has these columns:

- NAME

  The instrument name associated with the lock.
- OBJECT_INSTANCE_BEGIN

  The address in memory of the instrumented lock.
- WRITE_LOCKED_BY_THREAD_ID

  When a thread currently has an rwlock locked in exclusive (write) mode, WRITE_LOCKED_BY_THREAD_ID is the THREAD_ID of the locking thread, otherwise it is NULL.
- READ_LOCKED_BY_COUNT

  When a thread currently has an rwlock locked in shared (read) mode, READ_LOCKED_BY_COUNT is incremented by 1. This is a counter only, so it cannot be used directly to find which thread holds a read lock, but it can be used to see whether there is a read contention on an rwlock, and see how many readers are currently active.

By performing queries on both of the following tables, a monitoring application or a DBA may detect some bottlenecks or deadlocks between threads that involve locks:

- events_waits_current, to see what rwlock a thread is waiting for
- rwlock_instances, to see which other thread currently owns an rwlock

There is a limitation: The rwlock_instances can be used only to identify the thread holding a write lock, but not the threads holding a read lock.

20.9.3.5 The socket_instances Table

The socket_instances table provides a real-time snapshot of the active connections to the MySQL server. The table contains one row per TCP/IP or Unix socket file connection. Information available in this table provides a real-time snapshot of the active connections to the server. (Additional information is available in socket summary tables, including network activity such as socket operations and number of bytes transmitted and received; see Section 20.9.12.9, “Socket Summary Tables”).

```
mysql> SELECT * FROM socket_instances\G
*************************** 1. row ***************************
           EVENT_NAME: wait/io/socket/sql/server_unix_socket
OBJECT_INSTANCE_BEGIN: 4316619408
            THREAD_ID: 1
            SOCKET_ID: 16
                   IP:
                 PORT: 0
                STATE: ACTIVE
*************************** 2. row ***************************
           EVENT_NAME: wait/io/socket/sql/client_connection
OBJECT_INSTANCE_BEGIN: 4316644608
            THREAD_ID: 21
            SOCKET_ID: 39
                   IP: 127.0.0.1
                 PORT: 55233
                STATE: ACTIVE
*************************** 3. row ***************************
           EVENT_NAME: wait/io/socket/sql/server_tcpip_socket
OBJECT_INSTANCE_BEGIN: 4316699040
            THREAD_ID: 1
            SOCKET_ID: 14
                   IP: 0.0.0.0
                 PORT: 50603
                STATE: ACTIVE
```

Socket instruments have names of the form wait/io/socket/sql/*socket_type* and are used like this:

1. The server has a listening socket for each network protocol that it supports. The instruments associated with listening sockets for TCP/IP or Unix socket file connections have a *socket_type* value of server_tcpip_socket or server_unix_socket, respectively.

2. When a listening socket detects a connection, the server transfers the connection to a new socket managed by a separate thread. The instrument for the new connection thread has a *socket_type* value of client_connection.

3. When a connection terminates, the row in socket_instances corresponding to it is deleted.

The socket_instances table has these columns:

- EVENT_NAME

  The name of the wait/io/socket/* instrument that produced the event. This is a NAME value from the setup_instruments table. Instrument names may have multiple parts and form a hierarchy, as discussed in Section 20.4, “Performance Schema Instrument Naming Conventions”.

- OBJECT_INSTANCE_BEGIN

  This column uniquely identifies the socket. The value is the address of an object in memory.

- THREAD_ID

  The internal thread identifier assigned by the server. Each socket is managed by a single thread, so each socket can be mapped to a thread which can be mapped to a server process.

- SOCKET_ID

  The internal file handle assigned to the socket.

- IP

  The client IP address. The value may be either an IPv4 or IPv6 address, or blank to indicate a Unix socket file connection.

- PORT

  The TCP/IP port number, in the range from 0 to 65535.

- STATE

  The socket status, either IDLE or ACTIVE. Wait times for active sockets are tracked using the corresponding socket instrument. Wait times for idle sockets are tracked using the idle instrument.

  A socket is idle if it is waiting for a request from the client. When a socket becomes idle, the event row in socket_instances that is tracking the socket switches from a status of ACTIVE to IDLE. The EVENT_NAME value remains wait/io/socket/*, but timing for the instrument is suspended. Instead, an event is generated in the events_waits_current table with an EVENT_NAME value of idle.

  When the next request is received, the idle event terminates, the socket instance switches from IDLE to ACTIVE, and timing of the socket instrument resumes.

The IP:PORT column combination value identifies the connection. This combination value is used in the OBJECT_NAME column of the events_waits_*xxx* tables, to identify the connection from which socket events come:

- For the Unix domain listener socket (server_unix_socket), the port is 0, and the IP is ''.
- For client connections via the Unix domain listener (client_connection), the port is 0, and the IP is ''.
- For the TCP/IP server listener socket (server_tcpip_socket), the port is always the master port (for example, 3306), and the IP is always 0.0.0.0.
- For client connections via the TCP/IP listener (client_connection), the port is whatever the server assigns, but never 0. The IP is the IP of the originating host (127.0.0.1 or ::1 for the local host)

20.9.4 Performance Schema Wait Event Tables

These tables store wait events:

- events_waits_current: Current wait events
- events_waits_history: The most recent wait events for each thread
- events_waits_history_long: The most recent wait events overall

The following sections describe those tables. There are also summary tables that aggregate information about wait events; see Section 20.9.12.1, “Event Wait Summary Tables”.

Wait Event Configuration

The setup_instruments table contains instruments with name that begin with wait. For example:

```
mysql> SELECT * FROM setup_instruments WHERE NAME LIKE 'wait/io/file/innodb%';
+--------------------------------------+---------+-------+
| NAME                                 | ENABLED | TIMED |
+--------------------------------------+---------+-------+
| wait/io/file/innodb/innodb_data_file | YES     | YES   |
| wait/io/file/innodb/innodb_log_file  | YES     | YES   |
| wait/io/file/innodb/innodb_temp_file | YES     | YES   |
+--------------------------------------+---------+-------+
mysql> SELECT * FROM setup_instruments WHERE NAME LIKE 'wait/io/socket/%';
+----------------------------------------+---------+-------+
| NAME                                   | ENABLED | TIMED |
+----------------------------------------+---------+-------+
| wait/io/socket/sql/server_tcpip_socket | NO      | NO    |
| wait/io/socket/sql/server_unix_socket  | NO      | NO    |
| wait/io/socket/sql/client_connection   | NO      | NO    |
+----------------------------------------+---------+-------+
```

To modify collection of wait events, change the `ENABLED` and `TIMING` columns of the relevant instruments. For example:

```
mysql> UPDATE setup_instruments SET ENABLED = 'YES', TIMED = 'YES'
    -> WHERE NAME LIKE 'wait/io/socket/sql/%';
```

The `setup_consumers` table contains consumer values with names corresponding to the current and recent wait event table names. These consumers may be used to filter collection of wait events. By default, the wait consumers are disabled:

```
mysql> SELECT * FROM setup_consumers WHERE NAME LIKE '%waits%';
+---------------------------+---------+
| NAME                      | ENABLED |
+---------------------------+---------+
| events_waits_current      | NO      |
| events_waits_history      | NO      |
| events_waits_history_long | NO      |
+---------------------------+---------+
```

To enable all wait consumers, do this:

```
mysql> UPDATE setup_consumers SET ENABLED = 'YES'
    -> WHERE NAME LIKE '%waits%';
```

The `setup_timers` table contains a row with a `NAME` value of `wait` that indicates the unit for wait event timing. The default unit is `CYCLE`.

```
mysql> SELECT * FROM setup_timers WHERE NAME = 'wait';
+------+------------+
| NAME | TIMER_NAME |
+------+------------+
| wait | CYCLE      |
+------+------------+
```

To change the timing unit, modify the `TIMER_NAME` value:

```
mysql> UPDATE setup_timers SET TIMER_NAME = 'NANOSECOND'
    -> WHERE NAME = 'wait';
```

For additional information about configuring event collection, see Section 20.2, “Performance Schema Configuration”.

20.9.4.1 The `events_waits_current` Table

The events_waits_current table contains current wait events, one row per thread showing the current status of the thread's most recent monitored wait event.

The events_waits_current table can be truncated with TRUNCATE TABLE.

Of the tables that contain wait event rows, events_waits_current is the most fundamental. Other tables that contain wait event rows are logically derived from the current events. For example, the events_waits_history and events_waits_history_long tables are collections of the most recent wait events, up to a fixed number of rows.

For information about configuration of wait event collection, see Section 20.9.4, “Performance Schema Wait Event Tables”.

The events_waits_current table has these columns:

- THREAD_ID, EVENT_ID

  The thread associated with the event and the thread current event number when the event starts. The THREAD_ID and EVENT_ID values taken together form a primary key that uniquely identifies the row. No two rows will have the same pair of values.

- END_EVENT_ID

  This column is set to NULL when the event starts and updated to the thread current event number when the event ends.

- EVENT_NAME

  The name of the instrument that produced the event. This is a NAME value from the setup_instruments table. Instrument names may have multiple parts and form a hierarchy, as discussed in Section 20.4, “Performance Schema Instrument Naming Conventions”.

- SOURCE

  The name of the source file containing the instrumented code that produced the event and the line number in the file at which the instrumentation occurs. This enables you to check the source to determine exactly what code is involved. For example, if a mutex or lock is being blocked, you can check the context in which this occurs.

- TIMER_START, TIMER_END, TIMER_WAIT

  Timing information for the event. The unit for these values is picoseconds (trillionths of a second). The TIMER_START and TIMER_END values indicate when event timing started and ended. TIMER_WAIT is the event elapsed time (duration).

  If an event has not finished, TIMER_END and TIMER_WAIT are NULL.

  If an event is produced from an instrument that has TIMED = NO, timing information is not collected, and TIMER_START, TIMER_END, and TIMER_WAIT are all NULL.

  For discussion of picoseconds as the unit for event times and factors that affect time values, see Section 20.2.3.1, “Performance Schema Event Timing”.

- SPINS

  For a mutex, the number of spin rounds. If the value is NULL, the code does not use spin rounds or spinning is not instrumented.

- OBJECT_SCHEMA, OBJECT_NAME, OBJECT_TYPE, OBJECT_INSTANCE_BEGIN

  These columns identify the object "being acted on." What that means depends on the object type.

  For a synchronization object (cond, mutex, rwlock):

  - OBJECT_SCHEMA, OBJECT_NAME, and OBJECT_TYPE are NULL.
  - OBJECT_INSTANCE_BEGIN is the address of the synchronization object in memory.

  For a file I/O object:

  - OBJECT_SCHEMA is NULL.
  - OBJECT_NAME is the file name.
  - OBJECT_TYPE is FILE.
  - OBJECT_INSTANCE_BEGIN is an address in memory.

  For a socket object:

  - OBJECT_NAME is the IP:PORT value for the socket.
  - OBJECT_INSTANCE_BEGIN is an address in memory.

  For a table I/O object:

  - OBJECT_SCHEMA is the name of the schema that contains the table.
  - OBJECT_NAME is the table name.
  - OBJECT_TYPE is TABLE for a persistent base table or TEMPORARY TABLE for a temporary table.
  - OBJECT_INSTANCE_BEGIN is an address in memory.

  An OBJECT_INSTANCE_BEGIN value itself has no meaning, except that different values indicate different objects. OBJECT_INSTANCE_BEGIN can be used for debugging. For example, it can be used with GROUP BY OBJECT_INSTANCE_BEGIN to see whether the load on 1,000 mutexes (that protect, say, 1,000 pages or blocks of data) is spread evenly or just hitting a few bottlenecks. This can help you correlate with other sources of information if you see the same object address in a log file or another debugging or performance tool.

- INDEX_NAME

  The name of the index used. PRIMARY indicates the table primary index. NULL means that no index was used.

- NESTING_EVENT_ID

  The EVENT_ID value of the event within which this event is nested.

- NESTING_EVENT_TYPE

  The nesting event type. The value is TRANSACTION, STATEMENT, STAGE, or WAIT.

- OPERATION

  The type of operation performed, such as lock, read, or write.

- NUMBER_OF_BYTES

  The number of bytes read or written by the operation. For table I/O waits, NUMBER_OF_BYTES is NULL.

- FLAGS

  Reserved for future use.

20.9.4.2 The events_waits_history Table

The events_waits_history table contains the most recent 10 wait events per thread. To change the table size, modify the performance_schema_events_waits_history_size system variable at server startup. Wait events are not added to the table until they have ended. As new events are added, older events are discarded if the table is full.

The events_waits_history table has the same structure as events_waits_current. See Section 20.9.4.1, "The events_waits_current Table".

The events_waits_history table can be truncated with TRUNCATE TABLE.

For information about configuration of wait event collection, see Section 20.9.4, "Performance Schema Wait Event Tables".

20.9.4.3 The events_waits_history_long Table

The events_waits_history_long table contains the most recent 10,000 wait events. To change the table size, modify the performance_schema_events_waits_history_long_size system variable at server startup. Wait events are not added to the table until they have ended. As new events are added, older events are discarded if the table is full.

The events_waits_history_long table has the same structure as events_waits_current. See Section 20.9.4.1, "The events_waits_current Table".

The events_waits_history_long table can be truncated with TRUNCATE TABLE.

For information about configuration of wait event collection, see Section 20.9.4, "Performance Schema Wait Event Tables".

20.9.5 Performance Schema Stage Event Tables

The Performance Schema instruments stages, which are steps during the statement-execution process, such as parsing a statement, opening a table, or performing a filesort operation. Stages correspond to the thread states displayed by SHOW PROCESSLIST or that are visible in the INFORMATION_SCHEMA.PROCESSLIST table. Stages begin and end when state values change.

Within the event hierarchy, wait events nest within stage events, which nest within statement events, which nest within transaction events.

These tables store stage events:

- events_stages_current: Current stage events
- events_stages_history: The most recent stage events for each thread
- events_stages_history_long: The most recent stage events overall

The following sections describe those tables. There are also summary tables that aggregate information about stage events; see Section 20.9.12.2, "Stage Summary Tables".

Stage Event Configuration

The setup_instruments table contains instruments with name that begin with stage. These instruments are disabled by default. For example:

```
mysql> SELECT * FROM setup_instruments WHERE NAME RLIKE 'stage/sql/[a-c]';
+---------------------------------------------------+---------+-------+
| NAME                                              | ENABLED | TIMED |
+---------------------------------------------------+---------+-------+
| stage/sql/After create                            | NO      | NO    |
| stage/sql/allocating local table                  | NO      | NO    |
| stage/sql/altering table                          | NO      | NO    |
| stage/sql/committing alter table to storage engine | NO      | NO    |
| stage/sql/Changing master                         | NO      | NO    |
| stage/sql/Checking master version                 | NO      | NO    |
| stage/sql/checking permissions                    | NO      | NO    |
| stage/sql/checking privileges on cached query     | NO      | NO    |
| stage/sql/checking query cache for query          | NO      | NO    |
| stage/sql/cleaning up                             | NO      | NO    |
| stage/sql/closing tables                          | NO      | NO    |
| stage/sql/Connecting to master                    | NO      | NO    |
| stage/sql/converting HEAP to MyISAM               | NO      | NO    |
| stage/sql/Copying to group table                  | NO      | NO    |
| stage/sql/Copying to tmp table                    | NO      | NO    |
| stage/sql/copy to tmp table                       | NO      | NO    |
| stage/sql/Creating sort index                     | NO      | NO    |
| stage/sql/creating table                          | NO      | NO    |
| stage/sql/Creating tmp table                      | NO      | NO    |
+---------------------------------------------------+---------+-------+
```

To modify collection of stage events, change the ENABLED and TIMING columns of the relevant instruments. For example:

```
mysql> UPDATE setup_instruments SET ENABLED = 'YES', TIMED = 'YES'
    -> WHERE NAME = 'stage/sql/altering table';
```

The setup_consumers table contains consumer values with names corresponding to the current and recent stage event table names. These consumers may be used to filter collection of stage events. By default, the stage consumers are disabled:

```
mysql> SELECT * FROM setup_consumers WHERE NAME LIKE '%stages%';
+----------------------------+---------+
| NAME                       | ENABLED |
+----------------------------+---------+
| events_stages_current      | NO      |
| events_stages_history      | NO      |
| events_stages_history_long | NO      |
+----------------------------+---------+
```

To enable all stage consumers, do this:

```
mysql> UPDATE setup_consumers SET ENABLED = 'YES'
    -> WHERE NAME LIKE '%stages%';
```

The setup_timers table contains a row with a NAME value of stage that indicates the unit for stage event timing. The default unit is NANOSECOND.

```
mysql> SELECT * FROM setup_timers WHERE NAME = 'stage';
+-------+------------+
| NAME  | TIMER_NAME |
+-------+------------+
| stage | NANOSECOND |
+-------+------------+
```

To change the timing unit, modify the TIMER_NAME value:

```
mysql> UPDATE setup_timers SET TIMER_NAME = 'MICROSECOND'
    -> WHERE NAME = 'stage';
```

For additional information about configuring event collection, see Section 20.2, “Performance Schema Configuration”.

20.9.5.1 The events_stages_current Table

The events_stages_current table contains current stage events, one row per thread showing the current status of the thread's most recent monitored stage event.

The events_stages_current table can be truncated with TRUNCATE TABLE.

Of the tables that contain stage event rows, events_stages_current is the most fundamental. Other tables that contain stage event rows are logically derived from the current events. For example, the events_stages_history and events_stages_history_long tables are collections of the most recent stage events, up to a fixed number of rows.

For information about configuration of stage event collection, see Section 20.9.5, “Performance Schema Stage Event Tables”.

The events_stages_current table has these columns:

- THREAD_ID, EVENT_ID

  The thread associated with the event and the thread current event number when the event starts. The THREAD_ID and EVENT_ID values taken together form a primary key that uniquely identifies the row. No two rows will have the same pair of values.

- END_EVENT_ID

  This column is set to NULL when the event starts and updated to the thread current event number when the event ends.

- EVENT_NAME

  The name of the instrument that produced the event. This is a NAME value from the setup_instruments table. Instrument names may have multiple parts and form a hierarchy, as discussed in Section 20.4, “Performance Schema Instrument Naming Conventions”.

- SOURCE

  The name of the source file containing the instrumented code that produced the event and the line number in the file at which the instrumentation occurs. This enables you to check the source to determine exactly what code is involved.

- TIMER_START, TIMER_END, TIMER_WAIT

  Timing information for the event. The unit for these values is picoseconds (trillionths of a second). The TIMER_START and TIMER_END values indicate when event timing started and ended. TIMER_WAIT is the event elapsed time (duration).

If an event has not finished, TIMER_END and TIMER_WAIT are NULL.

If an event is produced from an instrument that has TIMED = NO, timing information is not collected, and TIMER_START, TIMER_END, and TIMER_WAIT are all NULL.

For discussion of picoseconds as the unit for event times and factors that affect time values, see Section 20.2.3.1, “Performance Schema Event Timing”.

- NESTING_EVENT_ID

  The EVENT_ID value of the event within which this event is nested. The nesting event for a stage event is usually a statement event.

- NESTING_EVENT_TYPE

  The nesting event type. The value is TRANSACTION, STATEMENT, STAGE, or WAIT.

20.9.5.2 The events_stages_history Table

The events_stages_history table contains the most recent 10 stage events per thread. To change the table size, modify the performance_schema_events_stages_history_size system variable at server startup. Stage events are not added to the table until they have ended. As new events are added, older events are discarded if the table is full.

The events_stages_history table has the same structure as events_stages_current. See Section 20.9.5.1, “The events_stages_current Table”.

The events_stages_history table can be truncated with TRUNCATE TABLE.

For information about configuration of stage event collection, see Section 20.9.5, “Performance Schema Stage Event Tables”.

20.9.5.3 The events_stages_history_long Table

The events_stages_history_long table contains the most recent 10,000 stage events. To change the table size, modify the performance_schema_events_stages_history_long_size system variable at server startup. Stage events are not added to the table until they have ended. As new events are added, older events are discarded if the table is full.

The events_stages_history_long table has the same structure as events_stages_current. See Section 20.9.5.1, “The events_stages_current Table”.

The events_stages_history_long table can be truncated with TRUNCATE TABLE.

For information about configuration of stage event collection, see Section 20.9.5, “Performance Schema Stage Event Tables”.

20.9.6 Performance Schema Statement Event Tables

The Performance Schema instruments statement execution. Statement events occur at a high level of the event hierarchy: Wait events nest within stage events, which nest within statement events, which nest within transaction events.

These tables store statement events:

- events_statements_current: Current statement events

- events_statements_history: The most recent statement events for each thread
- events_statements_history_long: The most recent statement events overall
- prepared_statements_instances: Prepared statement instances and statistics (added in MySQL 5.7.4)

The following sections describe those tables. There are also summary tables that aggregate information about statement events; see Section 20.9.12.3, “Statement Summary Tables”.

Statement Event Configuration

The setup_instruments table contains instruments with name that begin with statement. These instruments are enabled by default:

```
mysql> SELECT * FROM setup_instruments WHERE NAME LIKE 'statement/%';
+---------------------------------------------+---------+-------+
| NAME                                        | ENABLED | TIMED |
+---------------------------------------------+---------+-------+
| statement/sql/select                        | YES     | YES   |
| statement/sql/create_table                  | YES     | YES   |
| statement/sql/create_index                  | YES     | YES   |
...
| statement/sp/stmt                           | YES     | YES   |
| statement/sp/set                            | YES     | YES   |
| statement/sp/set_trigger_field              | YES     | YES   |
| statement/scheduler/event                   | YES     | YES   |
| statement/com/Sleep                         | YES     | YES   |
| statement/com/Quit                          | YES     | YES   |
| statement/com/Init DB                       | YES     | YES   |
...
| statement/abstract/Query                    | YES     | YES   |
| statement/abstract/new_packet               | YES     | YES   |
| statement/abstract/relay_log                | YES     | YES   |
+---------------------------------------------+---------+-------+
```

To modify collection of statement events, change the ENABLED and TIMING columns of the relevant instruments. For example:

```
mysql> UPDATE setup_instruments SET ENABLED = 'NO'
    -> WHERE NAME LIKE 'statement/com/%';
```

The setup_consumers table contains consumer values with names corresponding to the current and recent statement event table names, and the statement digest consumer. These consumers may be used to filter collection of statement events and statement digesting. By default, only events_statements_current and statements_digest are enabled:

```
mysql> SELECT * FROM setup_consumers WHERE NAME LIKE '%statements%';
+--------------------------------+---------+
| NAME                           | ENABLED |
+--------------------------------+---------+
| events_statements_current      | YES     |
| events_statements_history      | NO      |
| events_statements_history_long | NO      |
| statements_digest              | YES     |
+--------------------------------+---------+
```

To enable all statement consumers, do this:

```
mysql> UPDATE setup_consumers SET ENABLED = 'YES'
    -> WHERE NAME LIKE '%statements%';
```

The setup_timers table contains a row with a NAME value of statement that indicates the unit for statement event timing. The default unit is NANOSECOND.

```
mysql> SELECT * FROM setup_timers WHERE NAME = 'statement';
+-----------+------------+
| NAME      | TIMER_NAME |
+-----------+------------+
| statement | NANOSECOND |
+-----------+------------+
```

To change the timing unit, modify the TIMER_NAME value:

```
mysql> UPDATE setup_timers SET TIMER_NAME = 'MICROSECOND'
    -> WHERE NAME = 'statement';
```

For additional information about configuring event collection, see Section 20.2, “Performance Schema Configuration”.

Statement Monitoring

Statement monitoring begins from the moment the server sees that activity is requested on a thread, to the moment when all activity has ceased. Typically, this means from the time the server gets the first packet from the client to the time the server has finished sending the response. Before MySQL 5.7.2, monitoring occurs only for top-level statements. Statements within stored programs and subqueries are not seen separately. As of 5.7.2, statements within stored programs are monitored like other statements.

When the Performance Schema instruments a request (server command or SQL statement), it uses instrument names that proceed in stages from more general (or “abstract”) to more specific until it arrives at a final instrument name.

Final instrument names correspond to server commands and SQL statements:

- Server commands correspond to the COM_*xxx* codes defined in the mysql_com.h header file and processed in sql/sql_parse.cc. Examples are COM_PING and COM_QUIT. Instruments for commands have names that begin with statement/com, such as statement/com/Ping and statement/com/Quit.

- SQL statements are expressed as text, such as DELETE FROM t1 or SELECT * FROM t2. Instruments for SQL statements have names that begin with statement/sql, such as statement/sql/delete and statement/sql/select.

Some final instrument names are specific to error handling:

- statement/com/Error accounts for messages received by the server that are out of band. It can be used to detect commands sent by clients that the server does not understand. This may be helpful for purposes such as identifying clients that are misconfigured or using a version of MySQL more recent than that of the server, or clients that are attempting to attack the server.

- statement/sql/error accounts for SQL statements that fail to parse. It can be used to detect malformed queries sent by clients. A query that fails to parse differs from a query that parses but fails due to an error during execution. For example, SELECT * FROM is malformed, and the statement/sql/error instrument is used. By contrast, SELECT * parses but fails with a No tables used error. In this case, statement/sql/select is used and the statement event contains information to indicate the nature of the error.

A request can be obtained from any of these sources:

- As a command or statement request from a client, which sends the request as packets
- As a statement string read from the relay log on a replication slave (as of MySQL 5.7.2)
- As an event from the Event Scheduler (as of MySQL 5.7.2)

The details for a request are not initially known and the Performance Schema proceeds from abstract to specific instrument names in a sequence that depends on the source of the request.

For a request received from a client:

1. When the server detects a new packet at the socket level, a new statement is started with an abstract instrument name of `statement/abstract/new_packet`.

2. When the server reads the packet number, it knows more about the type of request received, and the Performance Schema refines the instrument name. For example, if the request is a `COM_PING` packet, the instrument name becomes `statement/com/Ping` and that is the final name. If the request is a `COM_QUERY` packet, it is known to correspond to a SQL statement but not the particular type of statement. In this case, the instrument changes from one abstract name to a more specific but still abstract name, `statement/abstract/Query`, and the request requires further classification.

3. If the request is a statement, the statement text is read and given to the parser. After parsing, the exact statement type is known. If the request is, for example, an `INSERT` statement, the Performance Schema refines the instrument name from `statement/abstract/Query` to `statement/sql/insert`, which is the final name.

For a request read as a statement from the relay log on a replication slave:

1. Statements in the relay log are stored as text and are read as such. There is no network protocol, so the `statement/abstract/new_packet` instrument is not used. Instead, the initial instrument is `statement/abstract/relay_log`.

2. When the statement is parsed, the exact statement type is known. If the request is, for example, an `INSERT` statement, the Performance Schema refines the instrument name from `statement/abstract/Query` to `statement/sql/insert`, which is the final name.

The preceding description applies only for statement-based replication. For row-based replication, table I/O done on the slave as it processes row changes can be instrumented, but row events in the relay log do not appear as discrete statements.

For a request received from the Event Scheduler:

The event execution is instrumented using the name `statement/scheduler/event`. This is the final name.

Statements executed within the event body are instrumented using `statement/sql/*` names, without use of any preceding abstract instrument. An event is a stored program, and stored programs are precompiled in memory before execution. Consequently, there is no parsing at runtime and the type of each statement is known by the time it executes.

Statements executed within the event body are child statements. For example, if an event executes an `INSERT` statement, execution of the event itself is the parent, instrumented using `statement/scheduler/event`, and the `INSERT` is the child, instrumented using `statement/sql/insert`. The parent/child relationship holds *between* separate instrumented operations. This differs from the sequence of refinement that occurs *within* a single instrumented operation, from abstract to final instrument names.

For statistics to be collected for statements, it is not sufficient to enable only the final statement/sql/* instruments used for individual statement types. The abtract statement/abstract/* instruments must be enabled as well. This should not normally be an issue because all statement instruments are enabled by default. However, an application that enables or disables statement instruments selectively must take into account that disabling abstract instruments also disables statistics collection for the individual statement instruments. For example, to collect statistics for INSERT statements, statement/sql/insert must be enabled, but also statement/abstract/new_packet and statement/abstract/Query. Similarly, for replicated statements to be instrumented, statement/abstract/relay_log must be enabled.

No statistics are aggregated for for abstract instruments such as statement/abstract/Query because no statement is ever classified with an abstract instrument as the final statement name.

The abstract instrument names in the preceding discussion are as of MySQL 5.7.3. In earlier 5.7 versions, there was some renaming before those names were settled on:

- statement/abstract/new_packet was statement/com/ before MySQL 5.7.3.
- statement/abstract/Query was statement/com/Query before MySQL 5.7.3.
- statement/abstract/relay_log was statement/rpl/relay_log in MySQL 5.7.2 and did not exist before that.

20.9.6.1 The events_statements_current Table

The events_statements_current table contains current statement events, one row per thread showing the current status of the thread's most recent monitored statement event.

The events_statements_current table can be truncated with TRUNCATE TABLE.

Of the tables that contain statement event rows, events_statements_current is the most fundamental. Other tables that contain statement event rows are logically derived from the current events. For example, the events_statements_history and events_statements_history_long tables are collections of the most recent statement events, up to a fixed number of rows.

For information about configuration of statement event collection, see Section 20.9.6, “Performance Schema Statement Event Tables”.

The events_statements_current table has these columns:

- THREAD_ID, EVENT_ID

  The thread associated with the event and the thread current event number when the event starts. The THREAD_ID and EVENT_ID values taken together form a primary key that uniquely identifies the row. No two rows will have the same pair of values.

- END_EVENT_ID

  This column is set to NULL when the event starts and updated to the thread current event number when the event ends.

- EVENT_NAME

  The name of the instrument from which the event was collected. This is a NAME value from the setup_instruments table. Instrument names may have multiple parts and form a hierarchy, as discussed in Section 20.4, “Performance Schema Instrument Naming Conventions”.

For SQL statements, the EVENT_NAME value initially is statement/com/Query until the statement is parsed, then changes to a more appropriate value, as described in Section 20.9.6, “Performance Schema Statement Event Tables”.

- SOURCE

  The name of the source file containing the instrumented code that produced the event and the line number in the file at which the instrumentation occurs. This enables you to check the source to determine exactly what code is involved.

- TIMER_START, TIMER_END, TIMER_WAIT

  Timing information for the event. The unit for these values is picoseconds (trillionths of a second). The TIMER_START and TIMER_END values indicate when event timing started and ended. TIMER_WAIT is the event elapsed time (duration).

  If an event has not finished, TIMER_END and TIMER_WAIT are NULL.

  If an event is produced from an instrument that has TIMED = NO, timing information is not collected, and TIMER_START, TIMER_END, and TIMER_WAIT are all NULL.

  For discussion of picoseconds as the unit for event times and factors that affect time values, see Section 20.2.3.1, “Performance Schema Event Timing”.

- LOCK_TIME

  The time spent waiting for table locks. This value is computed in microseconds but normalized to picoseconds for easier comparison with other Performance Schema timers.

- SQL_TEXT

  The text of the SQL statement. For a command not associated with a SQL statement, the value is NULL.

- DIGEST

  The statement digest MD5 value as a string of 32 hexadecimal characters, or NULL if the statement_digest consumer is no. For more information about statement digesting, see Section 20.7, “Performance Schema Statement Digests”.

- DIGEST_TEXT

  The normalized statement digest text, or NULL if the statement_digest consumer is no. For more information about statement digesting, see Section 20.7, “Performance Schema Statement Digests”.

- CURRENT_SCHEMA

  The default database for the statement, NULL if there is none.

- OBJECT_SCHEMA, OBJECT_NAME, OBJECT_TYPE

  For nested statements (stored programs), these columns contain information about the parent statement. Otherwise they are NULL.

- OBJECT_INSTANCE_BEGIN

  This column identifies the statement. The value is the address of an object in memory.

- MYSQL_ERRNO

The statement error number, from the statement diagnostics area.

- RETURNED_SQLSTATE

  The statement SQLSTATE value, from the statement diagnostics area.

- MESSAGE_TEXT

  The statement error message, from the statement diagnostics area.

- ERRORS

  Whether an error occurred for the statement. The value is 0 if the SQLSTATE value begins with 00 (completion) or 01 (warning). The value is 1 is the SQLSTATE value is anything else.

- WARNINGS

  The number of warnings, from the statement diagnostics area.

- ROWS_AFFECTED

  The number of rows affected by the statement. For a description of the meaning of "affected," see Section 21.8.7.1, "mysql_affected_rows()".

- ROWS_SENT

  The number of rows returned by the statement.

- ROWS_EXAMINED

  The number of rows read from storage engines during statement execution.

- CREATED_TMP_DISK_TABLES

  Like the Created_tmp_disk_tables status variable, but specific to the statement.

- CREATED_TMP_TABLES

  Like the Created_tmp_tables status variable, but specific to the statement.

- SELECT_FULL_JOIN

  Like the Select_full_join status variable, but specific to the statement.

- SELECT_FULL_RANGE_JOIN

  Like the Select_full_range_join status variable, but specific to the statement.

- SELECT_RANGE

  Like the Select_range status variable, but specific to the statement.

- SELECT_RANGE_CHECK

  Like the Select_range_check status variable, but specific to the statement.

- SELECT_SCAN

  Like the Select_scan status variable, but specific to the statement.

- SORT_MERGE_PASSES

  Like the Sort_merge_passes status variable, but specific to the statement.

- SORT_RANGE

  Like the Sort_range status variable, but specific to the statement.

- SORT_ROWS

  Like the Sort_rows status variable, but specific to the statement.

- SORT_SCAN

  Like the Sort_scan status variable, but specific to the statement.

- NO_INDEX_USED

  1 if the statement performed a table scan without using an index, 0 otherwise.

- NO_GOOD_INDEX_USED

  1 if the server found no good index to use for the statement, 0 otherwise. For additional information, see the description of the Extra column from EXPLAIN output for the Range checked for each record value in Section 8.8.2, “EXPLAIN Output Format”.

- NESTING_EVENT_ID, NESTING_EVENT_TYPE, NESTING_EVENT_LEVEL

  Before MySQL 5.7.2, only NESTING_EVENT_ID and NESTING_EVENT_TYPE exist and are always NULL.

  As of MySQL 5.7.2, all three columns exist and are used with other columns to provide information as follows for top-level (unnested) statements and nested statements (executed within a stored program).

  For top level statements:

  ```
  OBJECT_TYPE = NULL
  OBJECT_SCHEMA = NULL
  OBJECT_NAME = NULL
  NESTING_EVENT_ID = NULL
  NESTING_EVENT_TYPE = NULL
  NESTING_LEVEL = 0
  ```

  For nested statements:

  ```
  OBJECT_TYPE = the parent statement object type
  OBJECT_SCHEMA = the parent statement object schema
  OBJECT_NAME = the parent statement object name
  NESTING_EVENT_ID = the parent statement EVENT_ID
  NESTING_EVENT_TYPE = 'STATEMENT'
  NESTING_LEVEL = the parent statement NESTING_LEVEL plus one
  ```

20.9.6.2 The events_statements_history Table

The events_statements_history table contains the most recent 10 statement events per thread. To change the table size, modify the performance_schema_events_statements_history_size system variable at server startup. Statement events are not added to the table until they have ended. As new events are added, older events are discarded if the table is full.

The events_statements_history table has the same structure as events_statements_current. See Section 20.9.6.1, “The events_statements_current Table”.

The events_statements_history table can be truncated with TRUNCATE TABLE.

For information about configuration of statement event collection, see Section 20.9.6, “Performance Schema Statement Event Tables”.

20.9.6.3 The events_statements_history_long Table

The events_statements_history_long table contains the most recent 10,000 statement events. To change the table size, modify the performance_schema_events_statements_history_long_size system variable at server startup. Statement events are not added to the table until they have ended. As new events are added, older events are discarded if the table is full.

The events_statements_history_long table has the same structure as events_statements_current. See Section 20.9.6.1, “The events_statements_current Table”.

The events_statements_history_long table can be truncated with TRUNCATE TABLE.

For information about configuration of statement event collection, see Section 20.9.6, “Performance Schema Statement Event Tables”.

20.9.6.4 The prepared_statements_instances Table

As of MySQL 5.7.4, the Performance Schema provides instrumentation for prepared statements, for which there are two protocols:

- The binary protocol. This is accessed through the MySQL C API and maps onto underlying server commands as shown in the following table.

C API Function	Corresponding Server Command
mysql_stmt_prepare()	COM_STMT_PREPARE
mysql_stmt_execute()	COM_STMT_EXECUTE
mysql_stmt_close()	COM_STMT_CLOSE

- The text protocol. This is accessed using SQL statements and maps onto underlying server commands as shown in the following table.

SQL Statement	Corresponding Server Command
PREPARE	SQLCOM_PREPARE
EXECUTE	SQLCOM_EXECUTE
DEALLOCATE PREPARE, DROP PREPARE	SQLCOM_DEALLOCATE PREPARE

Performance Schema prepared statement instrumentation covers both protocols. The following discussion refers to the server commands rather than the C API functions or SQL statements.

Information about prepared statements is available in the prepared_statements_instances table. This table enables inspection of prepared statements used in the server and provides aggregated statistics about them. To control the size of this table, set the performance_schema_max_prepared_statements_instances system variable at server startup.

Collection of prepared statement information depends on the statement instruments shown in the following table. These instruments are enabled by default. To modify them, update the setup_instruments table.

Server Command	Instrument
statement/com/Prepare	COM_STMT_PREPARE
statement/com/Execute	COM_STMT_EXECUTE
statement/sql/prepare_sql	SQLCOM_PREPARE
statement/sql/execute_sql	SQLCOM_EXECUTE

The Performance Schema manages the contents of the prepared_statements_instances table as follows:

- Statement preparation

  A COM_STMT_PREPARE or SQLCOM_PREPARE command creates a prepared statement in the server. If the statement is successfully instrumented, a new row is added to the prepared_statements_instances table. If the statement cannot be instrumented, Performance_schema_prepared_statements_lost status variable is incremented.

- Prepared statement execution

  Execution of a COM_STMT_EXECUTE or SQLCOM_PREPARE command for an instrumented prepared statement instance updates the corresponding prepared_statements_instances table row.

- Prepared statement deallocation

  Execution of a COM_STMT_CLOSE or SQLCOM_DEALLOCATE_PREPARE command for an instrumented prepared statement instance removes the corresponding prepared_statements_instances table row. To avoid resource leaks, removal occurs even if the prepared statement instruments described previously are disabled.

The prepared_statements_instances table has these columns:

- OBJECT_INSTANCE_BEGIN

  The address in memory of the instrumented prepared statement.

- STATEMENT_ID

  The internal statement ID assigned by the server. The text and binary protocols both use statement IDs.

- STATEMENT_NAME

  For the binary protocol, this column is NULL. For the text protocol, this column is the external statement name assigned by the user. For example, for the following SQL statement, the name of the prepared statement is stmt:

  ```
  PREPARE stmt FROM 'SELECT 1';
  ```

- SQL_TEXT

  The prepared statement text, with ? placeholder markers.

- OWNER_THREAD_ID, OWNER_EVENT_ID

  These columns indicate the event that created the prepared statement.

- OWNER_OBJECT_TYPE, OWNER_OBJECT_SCHEMA, OWNER_OBJECT_NAME

For a prepared statement created by a client session, these columns are NULL. For a prepared statement created by a stored program, these columns point to the stored program. A typical user error is forgetting to deallocate prepared statements. These columns can be used to find stored programs that leak prepared statements:

```
SELECT OWNER_OBJECT_TYPE, OWNER_OBJECT_SCHEMA, OWNER_OBJECT_NAME,
STATEMENT_NAME, SQL_TEXT
FROM performance_schema.prepared_statements_instances
WHERE OWNER_OBJECT_TYPE IS NOT NULL;
```

- TIMER_PREPARE

  The time spent executing the statement preparation itself.

- COUNT_REPREPARE

  The number of times the statement was reprepared internally (see Section 8.9.4, “Caching of Prepared Statements and Stored Programs”). Timing statistics for repreparation are not available because it is counted as part of statement execution, not as a separate operation.

- COUNT_EXECUTE, SUM_TIMER_EXECUTE, MIN_TIMER_EXECUTE, AVG_TIMER_EXECUTE, MAX_TIMER_EXECUTE

  Aggregated statistics for executions of the prepared statement.

- SUM_*xxx*

  The remaining SUM_*xxx* columns are the same as for the statement summary tables (see Section 20.9.12.3, “Statement Summary Tables”).

TRUNCATE TABLE resets the statistics columns of the table.

20.9.7 Performance Schema Transaction Tables

As of MySQL 5.7.3, the Performance Schema instruments transactions. Within the event hierarchy, wait events nest within stage events, which nest within statement events, which nest within transaction events.

These tables store transaction events:

- events_transactions_current: Current transaction events
- events_transactions_history: The most recent transaction events for each thread
- events_transactions_history_long: The most recent transaction events overall

The following sections describe those tables. There are also summary tables that aggregate information about transaction events; see Section 20.9.12.4, “Transaction Summary Tables”.

Transaction Event Configuration

The setup_instruments table contains an instrument named transaction. This instrument is disabled by default:

```
mysql> SELECT * FROM setup_instruments WHERE NAME = 'transaction';
+-------------+---------+-------+
| NAME        | ENABLED | TIMED |
+-------------+---------+-------+
| transaction | NO      | NO    |
+-------------+---------+-------+
```

To enable collection of transaction events, including timing information, do this:

```
mysql> UPDATE setup_instruments SET ENABLED = 'YES', TIMED = 'YES'
    -> WHERE NAME = 'transaction';
```

The setup_consumers table contains consumer values with names corresponding to the current and recent transaction event table names. These consumers may be used to filter collection of transaction events. By default, only events_transactions_current is enabled:

```
mysql> SELECT * FROM setup_consumers WHERE NAME LIKE '%transactions%';
+----------------------------------+---------+
| NAME                             | ENABLED |
+----------------------------------+---------+
| events_transactions_current      | YES     |
| events_transactions_history      | NO      |
| events_transactions_history_long | NO      |
+----------------------------------+---------+
```

To enable all transaction consumers, do this:

```
mysql> UPDATE setup_consumers SET ENABLED = 'YES'
    -> WHERE NAME LIKE '%transactions%';
```

The setup_timers table contains a row with a NAME value of transaction that indicates the unit for transaction event timing. The default unit is NANOSECOND.

```
mysql> SELECT * FROM setup_timers WHERE NAME = 'transaction';
+-------------+------------+
| NAME        | TIMER_NAME |
+-------------+------------+
| transaction | NANOSECOND |
+-------------+------------+
```

To change the timing unit, modify the TIMER_NAME value:

```
mysql> UPDATE setup_timers SET TIMER_NAME = 'MICROSECOND'
    -> WHERE NAME = 'transaction';
```

For additional information about configuring event collection, see Section 20.2, “Performance Schema Configuration”.

Transaction Boundaries

In MySQL Server, transactions start explicitly with these statements:

```
START TRANSACTION | BEGIN | XA START | XA BEGIN
```

Transactions also start implicitly. For example, when the autocommit system variable is enabled, the start of each statement starts a new transaction.

When autocommit is disabled, the first statement following a committed transaction marks the start of a new transaction. Subsequent statements are part of the transaction until it is committed.

Transactions explicitly end with these statements:

```
COMMIT | ROLLBACK | XA COMMIT | XA ROLLBACK
```

Transactions also end implicitly, by execution of DDL statements, locking statements, and server administration statements.

In the following discussion, references to START TRANSACTION also apply to BEGIN, XA START, and XA BEGIN. Similarly, references to COMMIT and ROLLBACK apply to XA COMMIT and XA ROLLBACK, respectively.

The Performance Schema defines transaction boundaries similarly to that of the server. The start and end of a transaction event closely match the corresponding state transitions in the server:

- For an explicitly started transaction, the transaction event starts during processing of the START TRANSACTION statement.
- For an implicitly started transaction, the transaction event starts on the first statement that uses a transactional engine after the previous transaction has ended.
- For any transaction, whether explicitly or implicitly ended, the transaction event ends when the server transitions out of the active transaction state during the processing of COMMIT or ROLLBACK.

There are subtle implications to this approach:

- Transaction events in the Performance Schema do not fully include the statement events associated with the corresponding START TRANSACTION, COMMIT, or ROLLBACK statements. There is a trivial amount of timing overlap between the transaction event and these statements.
- Statements that work with nontransactional engines have no effect on the transaction state of the connection. For implicit transactions, the transaction event begins with the first statement that uses a transactional engine. This means that statements operating exclusively on nontransactional tables are ignored, even following START TRANSACTION.

To illustrate, consider the following scenario:

```
1. SET autocommit = OFF;
2. CREATE TABLE t1 (a INT) ENGINE = InnoDB;
3. START TRANSACTION;                       -- Transaction 1 START
4. INSERT INTO t1 VALUES (1), (2), (3);
5. CREATE TABLE t2 (a INT) ENGINE = MyISAM; -- Transaction 1 COMMIT
                                            -- (implicit; DDL forces commit)
6. INSERT INTO t2 VALUES (1), (2), (3);     -- Update nontransactional table
7. UPDATE t2 SET a = a + 1;                 -- ... and again
8. INSERT INTO t1 VALUES (4), (5), (6);     -- Write to transactional table
                                            -- Transaction 2 START (implicit)
9. COMMIT;                                  -- Transaction 2 COMMIT
```

From the perspective of the server, Transaction 1 ends when table t2 is created. Transaction 2 does not start until a transactional table is accessed, despite the intervening updates to nontransactional tables.

From the perspective of the Performance Schema, Transaction 2 starts when the server transitions into an active transaction state. Statements 6 and 7 are not included within the boundaries of Transaction 2, which is consistent with how the server writes transactions to the binary log.

Transaction Instrumentation

Three attributes define transactions:

- Access mode (read only, read write)
- Isolation level (SERIALIZABLE, REPEATABLE READ, and so forth)
- Implicit (autocommit enabled) or explicit (autocommit disabled)

To reduce complexity of the transaction instrumentation and to ensure that the collected transaction data provides complete, meaningful results, all transactions are instrumented independently of access mode, isolation level, or autocommit mode.

To selectively examine transaction history, use the attribute columns in the transaction event tables: ACCESS_MODE, ISOLATION_LEVEL, and AUTOCOMMIT.

The cost of transaction instrumentation can be reduced various ways, such as enabling or disabling transaction instrumentation according to user, account, host, or thread (client connection).

Transactions and Nested Events

The parent of a transaction event is the event that initiated the transaction. For an explicitly started transaction, this includes the START TRANSACTION and COMMIT AND CHAIN statements. For an implicitly started transaction, it is the first statement that uses a transactional engine after the previous transaction ends.

In general, a transaction is the top-level parent to all events initiated during the transaction, including statements that explicitly end the transaction such as COMMIT and ROLLBACK. Exceptions are statements that implicitly end a transaction, such as DDL statements, in which case the current transaction must be committed before the new statement is executed.

Transactions and Stored Programs

Transactions and stored program events are related as follows:

- Stored Procedures

  Stored procedures operate independently of transactions. A stored procedure can be started within a transaction, and a transaction can be started or ended from within a stored procedure. If called from within a transaction, a stored procedure can execute statements that force a commit of the parent transaction and then start a new transaction.

  If a stored procedure is started within a transaction, that transaction is the parent of the stored procedure event.

  If a transaction is started by a stored procedure, the stored procedure is the parent of the transaction event.

- Stored Functions

  Stored functions are restricted from causing an explicit or implicit commit or rollback. Stored function events can reside within a parent transaction event.

- Triggers

  Triggers activate as part of a statement that accesses the table with which it is associated, so the parent of a trigger event is always the statement that activates it.

  Triggers cannot issue statements that cause an explicit or implicit commit or rollback of a transaction.

- Scheduled Events

The execution of the statements in the body of a scheduled event takes place in a new connection. Nesting of a scheduled event within a parent transaction is not applicable.

Transactions and Savepoints

Savepoint statements are recorded as separate statement events. Transaction events include separate counters for SAVEPOINT, ROLLBACK TO SAVEPOINT, and RELEASE SAVEPOINT statements issued during the transaction.

Transactions and Errors

Errors and warnings that occur within a transaction are recorded in statement events, but not in the corresponding transaction event. This includes transaction-specific errors and warnings, such as a rollback on a nontransactional table or GTID consistency errors.

20.9.7.1 The events_transactions_current Table

The events_transactions_current table (added in MySQL 5.7.3) contains current transaction events, one row per thread showing the current status of the thread's most recent monitored transaction event. For example:

```
mysql> SELECT * FROM events_transactions_current LIMIT 1\G
*************************** 1. row ***************************
                      THREAD_ID: 26
                       EVENT_ID: 7
                   END_EVENT_ID: NULL
                     EVENT_NAME: transaction
                          STATE: ACTIVE
                         TRX_ID: NULL
                           GTID: 3E11FA47-71CA-11E1-9E33-C80AA9429562:56
                            XID: NULL
                       XA_STATE: NULL
                         SOURCE: transaction.cc:150
                    TIMER_START: 420833537900000
                      TIMER_END: NULL
                     TIMER_WAIT: NULL
                    ACCESS_MODE: READ WRITE
                ISOLATION_LEVEL: REPEATABLE READ
                     AUTOCOMMIT: NO
           NUMBER_OF_SAVEPOINTS: 0
NUMBER_OF_ROLLBACK_TO_SAVEPOINT: 0
    NUMBER_OF_RELEASE_SAVEPOINT: 0
          OBJECT_INSTANCE_BEGIN: NULL
               NESTING_EVENT_ID: 6
             NESTING_EVENT_TYPE: STATEMENT
```

The events_transactions_current table can be truncated with TRUNCATE TABLE.

Of the tables that contain transaction event rows, events_transactions_current is the most fundamental. Other tables that contain transaction event rows are logically derived from the current events. For example, the events_transactions_history and events_transactions_history_long tables are collections of the most recent transaction events, up to a fixed number of rows.

For information about configuration of transaction event collection, see Section 20.9.7, “Performance Schema Transaction Tables”.

The events_transactions_current table has these columns:

- THREAD_ID, EVENT_ID

The thread associated with the event and the thread current event number when the event starts. The THREAD_ID and EVENT_ID values taken together form a primary key that uniquely identifies the row. No two rows will have the same pair of values.

- END_EVENT_ID

  This column is set to NULL when the event starts and updated to the thread current event number when the event ends.

- EVENT_NAME

  The name of the instrument from which the event was collected. This is a NAME value from the setup_instruments table. Instrument names may have multiple parts and form a hierarchy, as discussed in Section 20.4, "Performance Schema Instrument Naming Conventions".

- STATE

  The current transaction state. The value is ACTIVE (after START TRANSACTION or BEGIN), COMMITTED (after COMMIT), or ROLLED BACK (after ROLLBACK).

- TRX_ID

  Unused.

- GTID

  If gtid_mode=OFF, the value is NULL. If gtid_mode=ON, this is the value of gtid_next when the transaction started. If gtid_next=AUTOMATIC the value is AUTOMATIC, otherwise the value is a GTID in UUID:NUMBER format.

- XID

  The XA transaction identifier. It has the format described in Section 13.3.7.1, "XA Transaction SQL Syntax".

- XA_STATE

  The state of the XA transaction. The value is ACTIVE (after XA START), IDLE (after XA END), PREPARED (after XA PREPARE), ROLLED BACK (after XA ROLLBACK), or COMMITTED (after XA COMMIT).

- SOURCE

  The name of the source file containing the instrumented code that produced the event and the line number in the file at which the instrumentation occurs. This enables you to check the source to determine exactly what code is involved.

- TIMER_START, TIMER_END, TIMER_WAIT

  Timing information for the event. The unit for these values is picoseconds (trillionths of a second). The TIMER_START and TIMER_END values indicate when event timing started and ended. TIMER_WAIT is the event elapsed time (duration).

  If an event has not finished, TIMER_END and TIMER_WAIT are NULL.

  If an event is produced from an instrument that has TIMED = NO, timing information is not collected, and TIMER_START, TIMER_END, and TIMER_WAIT are all NULL.

For discussion of picoseconds as the unit for event times and factors that affect time values, see Section 20.2.3.1, “Performance Schema Event Timing”.

- ACCESS_MODE

  The transaction access mode. The value is READ ONLY or READ WRITE.

- ISOLATION_LEVEL

  The transaction isolation level. The value is REPEATABLE READ, READ COMMITTED, READ UNCOMMITTED, or SERIALIZABLE.

- AUTOCOMMIT

  Whether autcommit mode was enabled when the transaction started.

- NUMBER_OF_SAVEPOINTS, NUMBER_OF_ROLLBACK_TO_SAVEPOINT, NUMBER_OF_RELEASE_SAVEPOINT

  The number of SAVEPOINT, ROLLBACK TO SAVEPOINT, and RELEASE SAVEPOINT statements issued during the transaction.

- OBJECT_INSTANCE_BEGIN

  Unused.

- NESTING_EVENT_ID

  The EVENT_ID value of the event within which this event is nested.

- NESTING_EVENT_TYPE

  The nesting event type. The value is TRANSACTION, STATEMENT, STAGE, or WAIT. (TRANSACTION will not appear because transactions cannot be nested.)

20.9.7.2 The events_transactions_history Table

The events_transactions_history table (added in MySQL 5.7.3) contains the most recent 10 transaction events per thread. To change the table size, modify the performance_schema_events_transactions_history_size system variable at server startup. Transaction events are not added to the table until they have ended. As new events are added, older events are discarded if the table is full.

The events_transactions_history table has the same structure as events_transactions_current. See Section 20.9.7.1, “The events_transactions_current Table”.

The events_transactions_history table can be truncated with TRUNCATE TABLE.

For information about configuration of transaction event collection, see Section 20.9.7, “Performance Schema Transaction Tables”.

20.9.7.3 The events_transactions_history_long Table

The events_transactions_history_long table (added in MySQL 5.7.3) contains the most recent 10,000 transaction events. To change the table size, modify the performance_schema_events_transactions_history_long_size system variable at server startup. Transaction events are not added to the table until they have ended. As new events are added, older events are discarded if the table is full.

The events_transactions_history_long table has the same structure as events_transactions_current. See Section 20.9.7.1, "The events_transactions_current Table".

The events_transactions_history_long table can be truncated with TRUNCATE TABLE.

For information about configuration of transaction event collection, see Section 20.9.7, "Performance Schema Transaction Tables".

20.9.8 Performance Schema Connection Tables

The Performance Schema provides statistics about connections to the server. When a client connects, it does so under a particular user name and from a particular host. The Performance Schema tracks connections per account (user name plus host name) and separately per user name and per host name, using these tables:

- accounts: Connection statistics per client account
- hosts: Connection statistics per client host name
- users: Connection statistics per client user name

There are also summary tables that aggregate information about connections. See Section 20.9.12.8, "Connection Summary Tables".

The meaning of "account" in the connection tables is similar to its meaning in the MySQL grant tables in the mysql database, in the sense that the term refers to a combination of user and host values. Where they differ is that in the grant tables, the host part of an account can be a pattern, whereas in the connection tables the host value is always a specific nonpattern host name.

The connection tables all have CURRENT_CONNECTIONS and TOTAL_CONNECTIONS columns to track the current and total number of connections per "tracking value" on which statistics are based. The tables differ in what they use for the tracking value. The accounts table has USER and HOST columns to track connections per user name plus host name combination. The users and hosts tables have a USER and HOST column, respectively, to track connections per user name and per host name.

Suppose that clients named user1 and user2 each connect one time from hosta and hostb. The Performance Schema tracks the connections as follows:

- The accounts table will have four rows, for the user1/hosta, user1/hostb, user2/hosta, and user2/hostb account values, each row counting one connection per account.
- The users table will have two rows, for user1 and user2, each row counting two connections per user name.
- The hosts table will have two rows, for hosta and hostb, each row counting two connections per host name.

When a client connects, the Performance Schema determines which row in each connection table applies to the connection, using the tracking value appropriate to each table. If there is no such row, one is added. Then the Performance Schema increments by one the CURRENT_CONNECTIONS and TOTAL_CONNECTIONS columns in that row.

When a client disconnects, the Performance Schema decrements by one the CURRENT_CONNECTIONS column in the row and leaves the TOTAL_CONNECTIONS column unchanged.

The Performance Schema also counts threads for internal threads and user sessions that failed to authenticate. These are counted in rows with `USER` and `HOST` column values of `NULL`.

Each connection table can be truncated with `TRUNCATE TABLE`, which has this effect:

- Rows with `CURRENT_CONNECTIONS = 0` are deleted.
- For rows with `CURRENT_CONNECTIONS > 0`, `TOTAL_CONNECTIONS` is reset to `CURRENT_CONNECTIONS`.
- Connection summary tables that depend on the connection table are truncated implicitly (summary values are set to 0). For more information about implicit truncation, see Section 20.9.12.8, “Connection Summary Tables”.

20.9.8.1 The `accounts` Table

The `accounts` table contains a row for each account that has connected to the MySQL server. For each account, the table counts the current and total number of connections. To change the table size, modify the `performance_schema_accounts_size` system variable at server startup. To disable account statistics, set this variable to 0.

The `accounts` table has the following columns. For a description of how the Performance Schema maintains rows in this table, including the effect of `TRUNCATE TABLE`, see Section 20.9.8, “Performance Schema Connection Tables”.

- `USER`

  The client user name for the connection, or `NULL` for an internal thread or user session that failed to authenticate.

- `HOST`

  The host name from which the client connected, or `NULL` for an internal thread or user session that failed to authenticate.

- `CURRENT_CONNECTIONS`

  The current number of connections for the account.

- `TOTAL_CONNECTIONS`

  The total number of connections for the account.

20.9.8.2 The `hosts` Table

The `hosts` table contains a row for each host from which clients have connected to the MySQL server. For each host name, the table counts the current and total number of connections. To change the table size, modify the `performance_schema_hosts_size` system variable at server startup. To disable host statistics, set this variable to 0.

The `hosts` table has the following columns. For a description of how the Performance Schema maintains rows in this table, including the effect of `TRUNCATE TABLE`, see Section 20.9.8, “Performance Schema Connection Tables”.

- `HOST`

  The host name from which the client connected, or `NULL` for an internal thread or user session that failed to authenticate.

- CURRENT_CONNECTIONS

  The current number of connections for the host.

- TOTAL_CONNECTIONS

  The total number of connections for the host.

20.9.8.3 The users Table

The users table contains a row for each user who has connected to the MySQL server. For each user name, the table counts the current and total number of connections. To change the table size, modify the performance_schema_users_size system variable at server startup. To disable user statistics, set this variable to 0.

The users table has the following columns. For a description of how the Performance Schema maintains rows in this table, including the effect of TRUNCATE TABLE, see Section 20.9.8, “Performance Schema Connection Tables”.

- USER

  The client user name for the connection, or NULL for an internal thread or user session that failed to authenticate.

- CURRENT_CONNECTIONS

  The current number of connections for the user.

- TOTAL_CONNECTIONS

  The total number of connections for the user.

20.9.9 Performance Schema Connection Attribute Tables

Application programs can provide key/value connection attributes to be passed to the server at connect time, using the mysql_options() and mysql_options4() C API functions. The Performance Schema provides tables that expose this information through SQL statements:

- session_account_connect_attrs: Connection attributes per for the current session
- session_connect_attrs: Connection attributes for all sessions

Both tables have the same columns. The difference between them is that session_connect_attrs displays connection attributes for all sessions, whereas session_account_connect_attrs displays connections only for your own account.

- PROCESSLIST_ID

  The connection identifier for the session.

- ATTR_NAME

  The attribute name.

- ATTR_VALUE

  The attribute value.

- ORDINAL_POSITION

The order in which the attribute was added to the set of connection attributes.

20.9.10 Performance Schema Replication Tables

As of MySQL 5.7.2, the Performance Schema provides tables that expose replication information. This is similar to the information available from the `SHOW SLAVE STATUS` statement, but representation in table form is more accessible and has usability benefits:

- `SHOW SLAVE STATUS` output is useful for visual inspection, but not so much for programmatic use. By contrast, using the Performance Schema tables, information about slave status can be searched using general `SELECT` queries, including complex `WHERE` conditions, joins, and so forth.
- Query results can be saved in tables for further analysis, or assigned to variables and thus used in stored procedures.
- The replication tables provide better diagnostic information. For multi-threaded slave operation, `SHOW SLAVE STATUS` reports all coordinator and worker thread errors using the `Last_SQL_Errno` and `Last_SQL_Error` fields, so only the most recent of those errors is visible and information can be lost. The replication tables store errors on a per-thread basis without loss of information.
- The last seen transaction is visible in the replication tables on a per-worker basis. This is information not avilable from `SHOW SLAVE STATUS`.
- Developers familiar with the Performance Schema interface can extend the replication tables to provide additional information by adding rows to the tables.

Replication Table Descriptions

The Performance Schema provides several replication-related tables:

- Tables that contain information about the connection of the slave server to the master server:
  - `replication_connection_configuration`: Configuration parameters for connecting to the master
  - `replication_connection_status`: Current status of the connection to the master
- Tables that contain general (not thread-specific) information about execution of transactions received from the master:
  - `replication_execute_configuration`: Configuration parameters for transaction execution on the slave
  - `replication_execute_status`: Current transaction execution status on the slave
- Tables that contain information about specific threads responsible for execution of transactions received from the master:
  - `replication_execute_status_by_coordinator`: SQL thread or coordinator thread execution status
  - `replication_execute_status_by_worker`: Worker thread execution status (empty unless slave is multi-threaded)

The following sections describe each replication table in more detail, including the correspondence between the columns produced by `SHOW SLAVE STATUS` and the replication table columns in which the same information appears.

The remainder of this introduction to the replication tables describes how the Performance Schema populates them and which fields from `SHOW SLAVE STATUS` are not represented in the tables.

Replication Table Life Cycle

The Performance Schema populates the replication tables as follows:

- Prior to execution of `CHANGE MASTER TO`, the tables are empty.
- After `CHANGE MASTER TO`, the configuration parameters can be seen in the tables. At this time, there are no active slave threads, so the `THREAD_ID` columns are `NULL` and the `SERVICE_STATE` columns have a value of `OFF`.
- After `START SLAVE`, non-`NULL` `THREAD_ID` values can be seen. Threads that are idle or active have a `SERVICE_STATE` value of `ON`. The thread that connects to the master server has a value of `CONNECTING` while it establishes the connection, and `ON` thereafter as long as the connection lasts.
- After `STOP SLAVE`, the `THREAD_ID` columns become `NULL` and the `SERVICE_STATE` columns for threads that no longer exist have a value of `OFF`.
- The tables are preserved after `STOP SLAVE` or threads dying due to an error.
- The `replication_execute_status_by_worker` table is nonempty only when the slave is operating in multi-threaded mode. That is, if the `slave_parallel_workers` system variable is greater than 0, this table is populated when `START SLAVE` is executed, and the number of rows shows the number of workers.

`SHOW SLAVE STATUS` Information Not In the Replication Tables

The information in the Performance Schema replication tables differs somewhat from the information available from `SHOW SLAVE STATUS` because the tables are oriented toward use of global transaction identifiers (GTIDs), not file names and positions, and they represent server UUID values, not server ID values. Due to these differences, several `SHOW SLAVE STATUS` columns are not preserved in the Performance Schema replication tables, or are represented a different way:

- The following fields refer to file names and positions and are not preserved:

  ```
  Master_Log_File
  Read_Master_Log_Pos
  Relay_Log_File
  Relay_Log_Pos
  Relay_Master_Log_File
  Exec_Master_Log_Pos
  Until_Condition
  Until_Log_File
  Until_Log_Pos
  ```

- The `Master_Info_File` field is not preserved. It refers to the `master.info` file, which has been superseded by crash-safe slave tables.
- The following fields are based on `server_id`, not `server_uuid` [2162], and are not preserved:

  ```
  Master_Server_Id
  Replicate_Ignore_Server_Ids
  ```

- The `Skip_Counter` field is based on event counts, not GTIDs, and is not preserved.
- These error fields are aliases for `Last_SQL_Errno` and `Last_SQL_Error`, so they are not preserved:

```
Last_Errno
Last_Error
```

In the Performance Schema, this error information is available in the LAST_ERROR_NUMBER and LAST_ERROR_MESSAGE columns of the replication_execute_status_by_coordinator table (and replication_execute_status_by_worker if the slave is multi-threaded). Those tables provide more specific per-thread error information than is available from Last_Errno and Last_Error.

- Fields that provide information about command-line filtering options is not preserved:

```
Replicate_Do_DB
Replicate_Ignore_DB
Replicate_Do_Table
Replicate_Ignore_Table
Replicate_Wild_Do_Table
Replicate_Wild_Ignore_Table
```

- The Slave_IO_State and Slave_SQL_Running_State fields are not preserved. If needed, these values can be obtained from the process list by using the THREAD_ID column of the appropriate replication table and joining it with the ID column in the INFORMATION_SCHEMA PROCESSLIST table to select the STATE column of the latter table.

- The Executed_Gtid_Set field can show a large set with a great deal of text. Instead, the Performance Schema tables show GTIDs of transactions that are currently being applied by the slave. Alternatively, the set of executed GTIDs can be obtained from the value of the gtid_executed system variable.

- The Seconds_Behind_Master and Relay_Log_Space fields are in to-be-decided status and are not preserved.

20.9.10.1 The replication_connection_configuration Table

This table shows the configuration parameters used by the slave server for connecting to the master server. Parameters stored in the table can be changed at runtime with the CHANGE MASTER TO statement, as indicated in the column descriptions. This table was added in MySQL 5.7.2.

Compared to the replication_connection_status table, replication_connection_configuration changes less frequently. It contains values that define how the slave connects to the master and that remain constant during the connection, whereas replication_connection_status contains values that change during the connection.

The replication_connection_configuration table has these columns:

- HOST

  The master host that the slave is connected to. (CHANGE MASTER TO option: MASTER_HOST)

- PORT

  The port used to connect to the master. (CHANGE MASTER TO option: MASTER_PORT)

- USER

  The user name of the account used to connect to the master. (CHANGE MASTER TO option: MASTER_USER)

- NETWORK_INTERFACE

The network interface that the slave is bound to, if any. (CHANGE MASTER TO option: MASTER_BIND)

- AUTO_POSITION

  1 if autopositioning is in use; otherwise 0. (CHANGE MASTER TO option: MASTER_AUTO_POSITION)

- SSL_ALLOWED, SSL_CA_FILE, SSL_CA_PATH, SSL_CERTIFICATE, SSL_CIPHER, SSL_KEY, SSL_VERIFY_SERVER_CERTIFICATE, SSL_CRL_FILE, SSL_CRL_PATH

  These columns show the SSL parameters used by the slave to connect to the master, if any.

  SSL_ALLOWED has these values:

  - Yes if an SSL connection to the master is permitted
  - No if an SSL connection to the master is not permitted
  - Ignored if an SSL connection is permitted but the slave server does not have SSL support enabled

  CHANGE MASTER TO options for the other SSL columns: MASTER_SSL_CA, MASTER_SSL_CAPATH, MASTER_SSL_CERT, MASTER_SSL_CIPHER, MASTER_SSL_CRL, MASTER_SSL_CRLPATH, MASTER_SSL_KEY, MASTER_SSL_VERIFY_SERVER_CERT.

  Prior to MySQL 5.7.4, the value of SSL_CRL_PATH was not displayed correctly. (Bug #18174719)

- CONNECTION_RETRY_INTERVAL

  The number of seconds between connect retries. (CHANGE MASTER TO option: MASTER_CONNECT_RETRY)

- CONNECTION_RETRY_COUNT

  The number of times the slave can attempt to reconnect to the master in the event of a lost connection. (CHANGE MASTER TO option: MASTER_RETRY_COUNT)

The following table shows the correspondence between replication_connection_configuration columns and SHOW SLAVE STATUS columns.

replication_connection_configuration Column	**SHOW SLAVE STATUS Column**
HOST	Master_Host
PORT	Master_Port
USER	Master_User
NETWORK_INTERFACE	Master_Bind
AUTO_POSITION	Auto_Position
SSL_ALLOWED	Master_SSL_Allowed
SSL_CA_FILE	Master_SSL_CA_File
SSL_CA_PATH	Master_SSL_CA_Path
SSL_CERTIFICATE	Master_SSL_Cert
SSL_CIPHER	Master_SSL_Cipher
SSL_KEY	Master_SSL_Key
SSL_VERIFY_SERVER_CERTIFICATE	Master_SSL_Verify_Server_Cert

replication_connection_configuration Column	SHOW SLAVE STATUS Column
SSL_CRL_FILE	Master_SSL_Crl
SSL_CRL_PATH	Master_SSL_Crlpath
CONNECTION_RETRY_INTERVAL	Connect_Retry
CONNECTION_RETRY_COUNT	Master_Retry_Count

20.9.10.2 The replication_connection_status Table

This table shows the current status of the I/O thread that handles the slave server connection to the master server. This table was added in MySQL 5.7.2.

Compared to the replication_connection_configuration table, replication_connection_status changes more frequently. It contains values that change during the connection, whereas replication_connection_configuration contains values define how the slave connects to the master and that remain constant during the connection.

The replication_connection_status table has these columns:

- SOURCE_UUID

  The server_uuid [2162] value from the master.

- THREAD_ID

  The I/O thread ID.

- SERVICE_STATE

  ON (thread exists and is active or idle), OFF (thread no longer exists), or CONNECTING (thread exists and is connecting to the master).

- RECEIVED_TRANSACTION_SET

  The set of global transaction IDs (GTIDs) corresponding to all transactions received by this slave. Empty if GTIDs are not in use.

- LAST_ERROR_NUMBER, LAST_ERROR_MESSAGE

  The error number and error message of the most recent error that caused the I/O thread to stop. An error number of 0 and message of the empty string mean “no error.” If the LAST_ERROR_MESSAGE value is not empty, the error values also appear in the slave's error log.

  Issuing RESET MASTER or RESET SLAVE resets the values shown in these columns.

- LAST_ERROR_TIMESTAMP

  A timestamp in YYMMDD HH:MM:SS format that shows when the most recent I/O error took place.

The following table shows the correspondence between replication_connection_status columns and SHOW SLAVE STATUS columns.

replication_connection_status Column	SHOW SLAVE STATUS Column
SOURCE_UUID	Master_UUID
THREAD_ID	None

replication_connection_status Column	SHOW SLAVE STATUS Column
SERVICE_STATE	Slave_IO_Running
RECEIVED_TRANSACTION_SET	Retrieved_Gtid_Set
LAST_ERROR_NUMBER	Last_IO_Errno
LAST_ERROR_MESSAGE	Last_IO_Error
LAST_ERROR_TIMESTAMP	Last_IO_Error_Timestamp

20.9.10.3 The replication_execute_configuration Table

This table shows the configuration parameters that affect execution of transactions by the slave server. Parameters stored in the table can be changed at runtime with the CHANGE MASTER TO statement, as indicated in the column descriptions. This table was added in MySQL 5.7.2.

The replication_execute_configuration table has these columns:

- DESIRED_DELAY

  The number of seconds that the slave must lag the master. (CHANGE MASTER TO option: MASTER_DELAY)

The following table shows the correspondence between replication_execute_configuration columns and SHOW SLAVE STATUS columns.

replication_execute_configuration Column	SHOW SLAVE STATUS Column
DESIRED_DELAY	SQL_Delay

20.9.10.4 The replication_execute_status Table

This table shows the current general transaction execution status on the slave server. This table was added in MySQL 5.7.2.

Information in this table pertains to general aspects of transaction execution status that are not specific to any thread involved. Thread-specific status information is available in the replication_execute_status_by_coordinator table (and replication_execute_status_by_worker if the slave is multi-threaded).

The replication_execute_status table has these columns:

- SERVICE_STATE

  Reserved for future use.

- REMAINING_DELAY

  If the slave is waiting for DESIRED_DELAY seconds to pass since the master executed an event, this field contains the number of delay seconds remaining. At other times, this field is NULL. (The DESIRED_DELAY value is stored in the replication_execute_configuration table.)

The following table shows the correspondence between replication_execute_status columns and SHOW SLAVE STATUS columns.

replication_execute_status Column	SHOW SLAVE STATUS Column
SERVICE_STATE	None
REMAINING_DELAY	SQL_Remaining_Delay

20.9.10.5 The replication_execute_status_by_coordinator Table

If the slave is not multi-threaded, this table shows the status of the SQL thread. Otherwise, the slave uses multiple worker threads and a coordinator thread to manage them, and this table shows the status of the coordinator thread. (For a multi-threaded slave, the replication_execute_status_by_worker table shows the status of the worker threads.) This table was added in MySQL 5.7.2.

The replication_execute_status_by_coordinator table has these columns:

- THREAD_ID

  The SQL/coordinator thread ID.

- SERVICE_STATE

  ON (thread exists and is active or idle) or OFF (thread no longer exists).

- LAST_ERROR_NUMBER, LAST_ERROR_MESSAGE

  The error number and error message of the most recent error that caused the SQL/coordinator thread to stop. An error number of 0 and message of the empty string mean “no error.” If the LAST_ERROR_MESSAGE value is not empty, the error values also appear in the slave's error log.

  Issuing RESET MASTER or RESET SLAVE resets the values shown in these columns.

  All error codes and messages displayed in the LAST_ERROR_NUMBER and LAST_ERROR_MESSAGE columns correspond to error values listed in Section C.3, “Server Error Codes and Messages”.

- LAST_ERROR_TIMESTAMP

  A timestamp in YYMMDD HH:MM:SS format that shows when the most recent SQL/coordinator error occurred.

The following table shows the correspondence between replication_execute_status_by_coordinator columns and SHOW SLAVE STATUS columns.

replication_execute_status_by_coordinator Column	**SHOW SLAVE STATUS Column**
THREAD_ID	None
SERVICE_STATE	Slave_SQL_Running
LAST_ERROR_NUMBER	Last_SQL_Errno
LAST_ERROR_MESSAGE	Last_SQL_Error
LAST_ERROR_TIMESTAMP	Last_SQL_Error_Timestamp

20.9.10.6 The replication_execute_status_by_worker Table

If the slave is not multi-threaded, this table is empty. Otherwise, the slave uses multiple worker threads and a coordinator thread to manage them, and this table shows the status of the worker threads. (For a multi-threaded slave, the replication_execute_status_by_coordinator table shows the status of the coordinator thread.) This table was added in MySQL 5.7.2.

The replication_execute_status_by_worker table has these columns:

- WORKER_ID

The worker identifier (same value as the id column in the mysql.slave_worker_info table). After STOP SLAVE, the THREAD_ID column becomes NULL, but the WORKER_ID value is preserved.

- THREAD_ID

  The worker thread ID.

- SERVICE_STATE

  ON (thread exists and is active or idle) or OFF (thread no longer exists).

- LAST_SEEN_TRANSACTION

  The transaction that the worker has last seen. The worker has not necessarily executed this transaction because it could still be in the process of doing so.

  If the gtid_mode system variable value is OFF, this column is ANONYMOUS, indicating that transactions do not have global transaction identifiers (GTIDs) and are identified by file and position only.

  If gtid_mode is ON, the column value is defined as follows:

  - If no transaction has executed, the column is empty.
  - When a transaction has executed, the column is set from gtid_next as soon as gtid_next is set. From this moment, the column always shows a GTID.
  - The GTID is preserved until the next transaction is executed. If an error occurs, the column value is the GTID of the transaction being executed by the worker when the error occurred.
  - When the next GTID log event is picked up by this worker thread, this column is updated from gtid_next soon after gtid_next is set.

- LAST_ERROR_NUMBER, LAST_ERROR_MESSAGE

  The error number and error message of the most recent error that caused the worker thread to stop. An error number of 0 and message of the empty string mean “no error.” If the LAST_ERROR_MESSAGE value is not empty, the error values also appear in the slave's error log.

  Issuing RESET MASTER or RESET SLAVE resets the values shown in these columns.

  All error codes and messages displayed in the LAST_ERROR_NUMBER and LAST_ERROR_MESSAGE columns correspond to error values listed in Section C.3, “Server Error Codes and Messages”.

- LAST_ERROR_TIMESTAMP

  A timestamp in YYMMDD HH:MM:SS format that shows when the most recent worker error occurred.

The following table shows the correspondence between replication_execute_status_by_worker columns and SHOW SLAVE STATUS columns.

replication_execute_status_by_worker Column	**SHOW SLAVE STATUS Column**
WORKER_ID	None
THREAD_ID	None
SERVICE_STATE	None
LAST_SEEN_TRANSACTION	None

replication_execute_status_by_worker Column	SHOW SLAVE STATUS Column
LAST_ERROR_NUMBER	Last_SQL_Errno
LAST_ERROR_MESSAGE	Last_SQL_Error
LAST_ERROR_TIMESTAMP	Last_SQL_Error_Timestamp

20.9.11 Performance Schema Lock Tables

The Performance Schema exposes lock information through these tables:

- metadata_locks: Metadata locks held and requested
- table_handles: Table locks held and requested

The following sections describe these tables in more detail.

20.9.11.1 The metadata_locks Table

As of MySQL 5.7.3, the Performance Schema exposes metadata lock information through the metadata_locks table:

- Locks that have been granted (shows which sessions own which current metadata locks)
- Locks that have been requested but not yet granted (shows which sessions are waiting for which metadata locks).
- Lock requests that have been killed by the deadlock detector or timed out and are waiting for the requesting session's lock request to be discarded

This information enables you to understand metadata lock dependencies between sessions. You can see not only which lock a session is waiting for, but which session currently holds that lock.

The metadata_locks table is read only and cannot be updated. It is autosized by default; to configure the table size, set the performance_schema_max_metadata_locks system variable at server startup.

Metadata lock instrumentation is disabled by default. To enable it, enable the wait/lock/metadata/sql/mdl instrument in the setup_instruments table.

The Performance Schema maintains metadata_locks table content as follows, using the LOCK_STATUS column to indicate the status of each lock:

- When a metadata lock is requested and obtained immediately, a row with a status of GRANTED is inserted.
- When a metadata lock is requested and not obtained immediately, a row with a status of PENDING is inserted.
- When a metadata lock previously requested is granted, its row status is updated to GRANTED.
- When a metadata lock is released, its row is deleted.
- When a pending lock request is canceled by the deadlock detector to break a deadlock (ER_LOCK_DEADLOCK), its row status is updated from PENDING to VICTIM.
- When a pending lock request times out (ER_LOCK_WAIT_TIMEOUT), its row status is updated from PENDING to TIMEOUT.
- When granted lock or pending lock request is killed, its row status is updated from GRANTED or PENDING to KILLED.

- The VICTIM, TIMEOUT, and KILLED status values are brief and signify that the lock row is about to be deleted.

The metadata_locks table has these columns:

- OBJECT_TYPE

  The type of lock used in the metadata lock subsystem: The value is one of GLOBAL, SCHEMA, TABLE, FUNCTION, PROCEDURE, TRIGGER (currently unused), EVENT, or COMMIT.

- OBJECT_SCHEMA

  The schema that contains the object.

- OBJECT_NAME

  The name of the instrumented object.

- OBJECT_INSTANCE_BEGIN

  The address in memory of the instrumented object.

- LOCK_TYPE

  The lock type from the metadata lock subsystem. The value is one of INTENTION_EXCLUSIVE, SHARED, SHARED_HIGH_PRIO, SHARED_READ, SHARED_WRITE, SHARED_UPGRADABLE, SHARED_NO_WRITE, SHARED_NO_READ_WRITE, or EXCLUSIVE.

- LOCK_DURATION

  The lock duration from the metadata lock subsystem. The value is one of STATEMENT, TRANSACTION, or EXPLICIT. The STATEMENT and TRANSACTION values are for locks that are released at statement or transaction end, respectively. The EXPLICIT value is for locks that survive statement or transaction end and are released explicitly, such as global locks acquired with FLUSH TABLES WITH READ LOCK.

- LOCK_STATUS

  The lock status from the metadata lock subsystem. The value is one of PENDING, GRANTED, VICTIM, TIMEOUT, or KILLED. The Performance Schema assigns these values as described earlier in this section.

- SOURCE

  The name of the source file containing the instrumented code that produced the event and the line number in the file at which the instrumentation occurs. This enables you to check the source to determine exactly what code is involved.

- OWNER_THREAD_ID

  The thread requesting a metadata lock.

- OWNER_EVENT_ID

  The event requesting a metadata lock.

20.9.11.2 The table_handles Table

As of MySQL 5.7.3, the Performance Schema exposes table lock information through the table_handles table to show the table locks currently in effect for each opened table handle.

table_handles reports what is recorded by the table lock instrumentation. This information shows which table handles the server has open, how they are locked, and by which sessions.

The table_handles table is read only and cannot be updated. It is autosized by default; to configure the table size, set the performance_schema_max_table_handles system variable at server startup.

The table_handles table has these columns:

- OBJECT_TYPE

  The table opened by a table handle.

- OBJECT_SCHEMA

  The schema that contains the object.

- OBJECT_NAME

  The name of the instrumented object.

- OBJECT_INSTANCE_BEGIN

  The table handle address in memory.

- OWNER_THREAD_ID

  The thread owning the table handle.

- OWNER_EVENT_ID

  The event which caused the table handle to be opened.

- INTERNAL_LOCK

  The table lock used at the SQL level. The value is one of READ, READ WITH SHARED LOCKS, READ HIGH PRIORITY, READ NO INSERT, WRITE ALLOW WRITE, WRITE CONCURRENT INSERT, WRITE LOW PRIORITY, or WRITE. For information about these lock types, see the include/thr_lock.h source file.

- EXTERNAL_LOCK

  The table lock used at the storage engine level. The value is one of READ EXTERNAL or WRITE EXTERNAL.

20.9.12 Performance Schema Summary Tables

Summary tables provide aggregated information for terminated events over time. The tables in this group summarize event data in different ways.

Event Wait Summaries:

- events_waits_summary_global_by_event_name: Wait events summarized per event name
- events_waits_summary_by_instance: Wait events summarized per instance
- events_waits_summary_by_thread_by_event_name: Wait events summarized per thread and event name

Stage Summaries:

- `events_stages_summary_by_thread_by_event_name`: Stage waits summarized per thread and event name
- `events_stages_summary_global_by_event_name`: Stage waits summarized per event name

Statement Summaries:

- `events_statements_summary_by_digest`: Statement events summarized per schema and digest value
- `events_statements_summary_by_thread_by_event_name`: Statement events summarized per thread and event name
- `events_statements_summary_global_by_event_name`: Statement events summarized per event name
- `events_statements_summary_by_program`: Statement events summarized per stored program (stored procedures and functions, triggers, and events) (added in MySQL 5.7.2)
- `prepared_statements_instances`: Prepared statement instances and statistics (added in MySQL 5.7.4)

Transaction Summaries:

- `events_transactions_summary_by_account_by_event_name`: Transaction events per account and event name (added in MySQL 5.7.3)
- `events_transactions_summary_by_host_by_event_name`: Transaction events per host name and event name (added in MySQL 5.7.3)
- `events_transactions_summary_by_thread_by_event_name`: Transaction events per thread and event name (added in MySQL 5.7.3)
- `events_transactions_summary_by_user_by_event_name`: Transaction events per user name and event name (added in MySQL 5.7.3)
- `events_transactions_summary_global_by_event_name`: Transaction events per event name (added in MySQL 5.7.3)

Object Wait Summaries:

- `objects_summary_global_by_type`: Object summaries

File I/O Summaries:

- `file_summary_by_event_name`: File events summarized per event name
- `file_summary_by_instance`: File events summarized per file instance

Table I/O and Lock Wait Summaries:

- `table_io_waits_summary_by_index_usage`: Table I/O waits per index
- `table_io_waits_summary_by_table`: Table I/O waits per table
- `table_lock_waits_summary_by_table`: Table lock waits per table

Connection Summaries:

- `events_waits_summary_by_account_by_event_name`: Wait events summarized per account and event name

- `events_waits_summary_by_user_by_event_name`: Wait events summarized per user name and event name
- `events_waits_summary_by_host_by_event_name`: Wait events summarized per host name and event name
- `events_stages_summary_by_account_by_event_name`: Stage events summarized per account and event name
- `events_stages_summary_by_user_by_event_name`: Stage events summarized per user name and event name
- `events_stages_summary_by_host_by_event_name`: Stage events summarized per host name and event name
- `events_statements_summary_by_digest`: Statement events summarized per schema and digest value
- `events_statements_summary_by_account_by_event_name`: Statement events summarized per account and event name
- `events_statements_summary_by_user_by_event_name`: Statement events summarized per user name and event name
- `events_statements_summary_by_host_by_event_name`: Statement events summarized per host name and event name

Socket Summaries:

- `socket_summary_by_instance`: Socket waits and I/O summarized per instance
- `socket_summary_by_event_name`: Socket waits and I/O summarized per event name

Memory Summaries:

- `memory_summary_global_by_event_name`: Memory operations summarized globally per event name (added in MySQL 5.7.2)
- `memory_summary_by_thread_by_event_name`: Memory operations summarized per thread and event name (added in MySQL 5.7.2)
- `memory_summary_by_account_by_event_name`: Memory operations summarized per account and event name (added in MySQL 5.7.2)
- `memory_summary_by_user_by_event_name`: Memory operations summarized per user and event name (added in MySQL 5.7.2)
- `memory_summary_by_host_by_event_name`: Memory operations summarized per host and event name (added in MySQL 5.7.2)

Each summary table has grouping columns that determine how to group the data to be aggregated, and summary columns that contain the aggregated values. Tables that summarize events in similar ways often have similar sets of summary columns and differ only in the grouping columns used to determine how events are aggregated.

Summary tables can be truncated with `TRUNCATE TABLE`. Except for `events_statements_summary_by_digest` and the memory summary tables, the effect is to reset the summary columns to 0 or `NULL`, not to remove rows. This enables you to clear collected values and restart aggregation. That might be useful, for example, after you have made a runtime configuration change.

20.9.12.1 Event Wait Summary Tables

The Performance Schema maintains tables for collecting current and recent wait events, and aggregates that information in summary tables. Section 20.9.4, “Performance Schema Wait Event Tables” describes the events on which wait summaries are based. See that discussion for information about the content of wait events, the current and recent wait event tables, and how to control wait event collection.

Each event waits summary table has one or more grouping columns to indicate how the table aggregates events. Event names refer to names of event instruments in the `setup_instruments` table.

- `events_waits_summary_global_by_event_name` has an `EVENT_NAME` column. Each row summarizes events for a given event name. An instrument might be used to create multiple instances of the instrumented object. For example, if there is an instrument for a mutex that is created for each connection, there are as many instances as there are connections. The summary row for the instrument summarizes over all these instances.

- `events_waits_summary_by_instance` has `EVENT_NAME` and `OBJECT_INSTANCE_BEGIN` columns. Each row summarizes events for a given event name and object. If an instrument is used to create multiple instances, each instance has a unique `OBJECT_INSTANCE_BEGIN` value, so these instances are summarized separately in this table.

- `events_waits_summary_by_thread_by_event_name` has `THREAD_ID` and `EVENT_NAME` columns. Each row summarizes events for a given thread and event name.

All event waits summary tables have these summary columns containing aggregated values:

- `COUNT_STAR`

  The number of summarized events. This value includes all events, whether timed or nontimed.

- `SUM_TIMER_WAIT`

  The total wait time of the summarized timed events. This value is calculated only for timed events because nontimed events have a wait time of `NULL`. The same is true for the other *`xxx`*`_TIMER_WAIT` values.

- `MIN_TIMER_WAIT`

  The minimum wait time of the summarized timed events.

- `AVG_TIMER_WAIT`

  The average wait time of the summarized timed events.

- `MAX_TIMER_WAIT`

  The maximum wait time of the summarized timed events.

Example wait event summary information:

```
mysql> SELECT * FROM events_waits_summary_global_by_event_name\G
...
*************************** 6. row ***************************
    EVENT_NAME: wait/synch/mutex/sql/BINARY_LOG::LOCK_index
    COUNT_STAR: 8
SUM_TIMER_WAIT: 2119302
MIN_TIMER_WAIT: 196092
AVG_TIMER_WAIT: 264912
```

```
MAX_TIMER_WAIT: 569421
...
*************************** 9. row ***************************
    EVENT_NAME: wait/synch/mutex/sql/hash_filo::lock
    COUNT_STAR: 69
SUM_TIMER_WAIT: 16848828
MIN_TIMER_WAIT: 0
AVG_TIMER_WAIT: 244185
MAX_TIMER_WAIT: 735345
...
```

TRUNCATE TABLE is permitted for wait summary tables. It resets the summary columns to zero rather than removing rows.

20.9.12.2 Stage Summary Tables

The Performance Schema maintains tables for collecting current and recent stage events, and aggregates that information in summary tables. Section 20.9.5, "Performance Schema Stage Event Tables" describes the events on which stage summaries are based. See that discussion for information about the content of stage events, the current and recent stage event tables, and how to control stage event collection.

Each stage summary table has one or more grouping columns to indicate how the table aggregates events. Event names refer to names of event instruments in the setup_instruments table.

- events_stages_summary_by_thread_by_event_name has THREAD_ID and EVENT_NAME columns. Each row summarizes events for a given thread and event name.

- events_stages_summary_global_by_event_name has an EVENT_NAME column. Each row summarizes events for a given event name.

All stage summary tables have these summary columns containing aggregated values: COUNT_STAR, SUM_TIMER_WAIT, MIN_TIMER_WAIT, AVG_TIMER_WAIT, and MAX_TIMER_WAIT. These columns are analogous to the columns of the same names in the event wait summary tables (see Section 20.9.12.1, "Event Wait Summary Tables"), except that the stage summary tables aggregate events from events_stages_current rather than events_waits_current.

Example stage event summary information:

```
mysql> SELECT * FROM events_stages_summary_global_by_event_name\G
...
*************************** 5. row ***************************
    EVENT_NAME: stage/sql/checking permissions
    COUNT_STAR: 57
SUM_TIMER_WAIT: 26501888880
MIN_TIMER_WAIT: 7317456
AVG_TIMER_WAIT: 464945295
MAX_TIMER_WAIT: 12858936792
...
*************************** 9. row ***************************
    EVENT_NAME: stage/sql/closing tables
    COUNT_STAR: 37
SUM_TIMER_WAIT: 662606568
MIN_TIMER_WAIT: 1593864
AVG_TIMER_WAIT: 17907891
MAX_TIMER_WAIT: 437977248
...
```

TRUNCATE TABLE is permitted for stage summary tables. It resets the summary columns to zero rather than removing rows.

20.9.12.3 Statement Summary Tables

The Performance Schema maintains tables for collecting current and recent statement events, and aggregates that information in summary tables. Section 20.9.6, “Performance Schema Statement Event Tables” describes the events on which statement summaries are based. See that discussion for information about the content of statement events, the current and recent statement event tables, and how to control statement event collection.

Each statement summary table has one or more grouping columns to indicate how the table aggregates events. Event names refer to names of event instruments in the setup_instruments table.

- events_statements_summary_by_digest has SCHEMA_NAME and DIGEST columns. Each row summarizes events for given schema/digest values. (The DIGEST_TEXT column contains the corresponding normalized statement digest text, but is neither a grouping nor summary column.)
- events_statements_summary_by_program has OBJECT_TYPE, OBJECT_SCHEMA, and OBJECT_NAME columns. Each row summarizes events for a given stored program (stored procedure or function, trigger, or event).
- events_statements_summary_by_thread_by_event_name has THREAD_ID and EVENT_NAME columns. Each row summarizes events for a given thread and event name.
- events_statements_summary_global_by_event_name has an EVENT_NAME column. Each row summarizes events for a given event name.
- prepared_statements_instances has an OBJECT_INSTANCE_BEGIN column. Each row summarizes events for a given prepared statement.

Statement summary tables have these summary columns containing aggregated values:

- COUNT_STAR, SUM_TIMER_WAIT, MIN_TIMER_WAIT, AVG_TIMER_WAIT, MAX_TIMER_WAIT

  These columns are analogous to the columns of the same names in the event wait summary tables (see Section 20.9.12.1, “Event Wait Summary Tables”), except that the statement summary tables aggregate events from events_statements_current rather than events_waits_current.

  The prepared_statements_instances table does not have these columns.

- SUM_*xxx*

  The aggregate of the corresponding *xxx* column in the events_statements_current table. For example, the SUM_LOCK_TIME and SUM_ERRORS columns in statement summary tables are the aggregates of the LOCK_TIME and ERRORS columns in events_statements_current table.

The events_statements_summary_by_digest table has these additional summary columns:

- FIRST_SEEN_TIMESTAMP, LAST_SEEN_TIMESTAMP

  The times at which a statement with the given digest value were first seen and most recently seen.

The events_statements_summary_by_program table has these additional summary columns:

- COUNT_STATEMENTS, SUM_STATEMENTS_WAIT, MIN_STATEMENTS_WAIT, AVG_STATEMENTS_WAIT, MAX_STATEMENTS_WAIT

  Statistics about nested statements invoked during stored program execution.

The prepared_statements_instances table has these additional summary columns:

- COUNT_EXECUTE, SUM_TIMER_EXECUTE, MIN_TIMER_EXECUTE, AVG_TIMER_EXECUTE, MAX_TIMER_EXECUTE

Aggregated statistics for executions of the prepared statement.

Example statement event summary information:

```
mysql> SELECT * FROM events_statements_summary_global_by_event_name\G
*************************** 1. row ***************************
                 EVENT_NAME: statement/sql/select
                 COUNT_STAR: 25
             SUM_TIMER_WAIT: 1535983999000
             MIN_TIMER_WAIT: 209823000
             AVG_TIMER_WAIT: 61439359000
             MAX_TIMER_WAIT: 1363397650000
              SUM_LOCK_TIME: 20186000000
                 SUM_ERRORS: 0
               SUM_WARNINGS: 0
          SUM_ROWS_AFFECTED: 0
              SUM_ROWS_SENT: 388
          SUM_ROWS_EXAMINED: 370
SUM_CREATED_TMP_DISK_TABLES: 0
     SUM_CREATED_TMP_TABLES: 0
       SUM_SELECT_FULL_JOIN: 0
 SUM_SELECT_FULL_RANGE_JOIN: 0
           SUM_SELECT_RANGE: 0
     SUM_SELECT_RANGE_CHECK: 0
            SUM_SELECT_SCAN: 6
      SUM_SORT_MERGE_PASSES: 0
             SUM_SORT_RANGE: 0
              SUM_SORT_ROWS: 0
              SUM_SORT_SCAN: 0
          SUM_NO_INDEX_USED: 6
     SUM_NO_GOOD_INDEX_USED: 0
...
```

TRUNCATE TABLE is permitted for statement summary tables. For events_statements_summary_by_digest, it empties the table. For the other statement summary tables, it resets the summary columns to zero rather than removing rows.

Statement Digest Aggregation Rules

If the statement_digest consumer is enabled, aggregation into events_statements_summary_by_digest occurs as follows when a statement completes. Aggregation is based on the DIGEST value computed for the statement.

- If a events_statements_summary_by_digest row already exists with the digest value for the statement that just completed, statistics for the statement are aggregated to that row. The LAST_SEEN column is updated to the current time.
- If no row has the digest value for the statement that just completed, and the table is not full, a new row is created for the statement. The FIRST_SEEN and LAST_SEEN columns are initialized with the current time.
- If no row has the statement digest value for the statement that just completed, and the table is full, the statistics for the statement that just completed are added to a special “catch-all” row with DIGEST = NULL, which is created if necessary. If the row is created, the FIRST_SEEN and LAST_SEEN columns are initialized with the current time. Otherwise, the LAST_SEEN column is updated with the current time.

The row with DIGEST = NULL is maintained because Performance Schema tables have a maximum size due to memory constraints. The DIGEST = NULL row permits digests that do not match other rows to be counted even if the summary table is full, using a common “other” bucket. This row helps you estimate whether the digest summary is representative:

- A DIGEST = NULL row that has a COUNT_STAR value that represents 5% of all digests shows that the digest summary table is very representative; the other rows cover 95% of the statements seen.

- A DIGEST = NULL row that has a COUNT_STAR value that represents 50% of all digests shows that the digest summary table is not very representative; the other rows cover only half the statements seen. Most likely the DBA should increase the maximum table size so that more of the rows counted in the DIGEST = NULL row would be counted using more specific rows instead. To do this, set the performance_schema_digests_size system variable to a larger value at server startup. The default size is 200.

Stored Program Instrumentation Behavior

For stored program types for which instrumentation is enabled in the setup_objects table, events_statements_summary_by_program maintains statistics for stored programs as follows:

- A row is added for an object when it is first used in the server.

- The row for an object is removed when the object is dropped.

- Statistics are aggregated in the row for an object as it executes.

See also Section 20.2.3.3, “Event Pre-Filtering”.

20.9.12.4 Transaction Summary Tables

As of MySQL 5.7.3, the Performance Schema maintains tables for collecting current and recent transaction events, and aggregates that information in summary tables. Section 20.9.7, “Performance Schema Transaction Tables” describes the events on which transaction summaries are based. See that discussion for information about the content of transaction events, the current and recent transaction event tables, and how to control transaction event collection, which is disabled by default.

Each transaction summary table has one or more grouping columns to indicate how the table aggregates events. Event names refer to names of event instruments in the setup_instruments table.

- events_transactions_summary_by_account_by_event_name has USER, HOST, and EVENT_NAME columns. Each row summarizes events for a given account and event name.

- events_transactions_summary_by_host_by_event_name has HOST and EVENT_NAME columns. Each row summarizes events for a given host and event name.

- events_transactions_summary_by_thread_by_event_name has THREAD_ID and EVENT_NAME columns. Each row summarizes events for a given thread and event name.

- events_transactions_summary_by_user_by_event_name has USER and EVENT_NAME columns. Each row summarizes events for a given user and event name.

- events_transactions_summary_global_by_event_name has an EVENT_NAME column. Each row summarizes events for a given event name.

All transaction summary tables have these summary columns containing aggregated values:

- COUNT_STAR, SUM_TIMER_WAIT, MIN_TIMER_WAIT, AVG_TIMER_WAIT, MAX_TIMER_WAIT

  These columns are analogous to the columns of the same names in the event wait summary tables (see Section 20.9.12.1, “Event Wait Summary Tables”), except that the transaction summary tables aggregate events from events_transactions_current rather than events_waits_current. These columns summarize read-write and read-only transactions.

- COUNT_READ_WRITE, SUM_TIMER_READ_WRITE, MIN_TIMER_READ_WRITE, AVG_TIMER_READ_WRITE, MAX_TIMER_READ_WRITE

  These are similar to the COUNT_STAR and *xxx*_TIMER_WAIT columns, but summarize read-write transactions only.

- COUNT_READ_ONLY, SUM_TIMER_READ_ONLY, MIN_TIMER_READ_ONLY, AVG_TIMER_READ_ONLY, MAX_TIMER_READ_ONLY

  These are similar to the COUNT_STAR and *xxx*_TIMER_WAIT columns, but summarize read-only transactions only.

Example transaction event summary information:

```
mysql> SELECT * FROM events_transactions_summary_global_by_event_name LIMIT 1\G
*************************** 1. row ***************************
          EVENT_NAME: transaction
          COUNT_STAR: 5
      SUM_TIMER_WAIT: 19550092000
      MIN_TIMER_WAIT: 2954148000
      AVG_TIMER_WAIT: 3910018000
      MAX_TIMER_WAIT: 5486275000
    COUNT_READ_WRITE: 5
SUM_TIMER_READ_WRITE: 19550092000
MIN_TIMER_READ_WRITE: 2954148000
AVG_TIMER_READ_WRITE: 3910018000
MAX_TIMER_READ_WRITE: 5486275000
     COUNT_READ_ONLY: 0
 SUM_TIMER_READ_ONLY: 0
 MIN_TIMER_READ_ONLY: 0
 AVG_TIMER_READ_ONLY: 0
 MAX_TIMER_READ_ONLY: 0
```

TRUNCATE TABLE is permitted for transaction summary tables. It resets the summary columns to zero rather than removing rows.

Transaction Aggregation Rules

Transaction events are collected regardless of isolation level, access mode, or autocommit mode.

Read-write transactions are generally more resource intensive than read-only transactions, therefore transaction summary tables include separate aggregate columns for read-write and read-only transactions.

Resource requirements may also vary with transaction isolation level. However, presuming that only one isolation level would be used per server, aggregation by isolation level is not provided.

20.9.12.5 Object Wait Summary Table

The objects_summary_global_by_type table aggregates object wait events. It has these grouping columns to indicate how the table aggregates events: OBJECT_TYPE, OBJECT_SCHEMA, and OBJECT_NAME. Each row summarizes events for the given object.

objects_summary_global_by_type has the same summary columns as the events_waits_summary_by_*xxx* tables. See Section 20.9.12.1, “Event Wait Summary Tables”.

Example object wait event summary information:

```
mysql> SELECT * FROM objects_summary_global_by_type\G
...
*************************** 3. row ***************************
```

```
   OBJECT_TYPE: TABLE
 OBJECT_SCHEMA: test
   OBJECT_NAME: t
    COUNT_STAR: 3
SUM_TIMER_WAIT: 263126976
MIN_TIMER_WAIT: 1522272
AVG_TIMER_WAIT: 87708678
MAX_TIMER_WAIT: 258428280
...
*************************** 10. row ***************************
   OBJECT_TYPE: TABLE
 OBJECT_SCHEMA: mysql
   OBJECT_NAME: user
    COUNT_STAR: 14
SUM_TIMER_WAIT: 365567592
MIN_TIMER_WAIT: 1141704
AVG_TIMER_WAIT: 26111769
MAX_TIMER_WAIT: 334783032
...
```

TRUNCATE TABLE is permitted for the object summary table. It resets the summary columns to zero rather than removing rows.

20.9.12.6 File I/O Summary Tables

The file I/O summary tables aggregate information about I/O operations.

Each file I/O summary table has one or more grouping columns to indicate how the table aggregates events. Event names refer to names of event instruments in the setup_instruments table.

- file_summary_by_event_name has an EVENT_NAME column. Each row summarizes events for a given event name.

- file_summary_by_instance has FILE_NAME, EVENT_NAME, and OBJECT_INSTANCE_BEGIN columns. Each row summarizes events for a given file and event name.

All file I/O summary tables have the following summary columns containing aggregated values. Some columns are more general and have values that are the same as the sum of the values of more fine-grained columns. In this way, aggregations at higher levels are available directly without the need for user-defined views that sum lower-level columns.

- COUNT_STAR, SUM_TIMER_WAIT, MIN_TIMER_WAIT, AVG_TIMER_WAIT, MAX_TIMER_WAIT

  These columns aggregate all I/O operations.

- COUNT_READ, SUM_TIMER_READ, MIN_TIMER_READ, AVG_TIMER_READ, MAX_TIMER_READ, SUM_NUMBER_OF_BYTES_READ

  These columns aggregate all read operations, including FGETS, FGETC, FREAD, and READ.

- COUNT_WRITE, SUM_TIMER_WRITE, MIN_TIMER_WRITE, AVG_TIMER_WRITE, MAX_TIMER_WRITE, SUM_NUMBER_OF_BYTES_WRITE

  These columns aggregate all write operations, including FPUTS, FPUTC, FPRINTF, VFPRINTF, FWRITE, and PWRITE.

- COUNT_MISC, SUM_TIMER_MISC, MIN_TIMER_MISC, AVG_TIMER_MISC, MAX_TIMER_MISC

  These columns aggregate all other I/O operations, including CREATE, DELETE, OPEN, CLOSE, STREAM_OPEN, STREAM_CLOSE, SEEK, TELL, FLUSH, STAT, FSTAT, CHSIZE, RENAME, and SYNC. There are no byte counts for these operations.

Example file I/O event summary information:

```
mysql> SELECT * FROM file_summary_by_event_name\G
...
*************************** 2. row ***************************
               EVENT_NAME: wait/io/file/sql/binlog
               COUNT_STAR: 31
           SUM_TIMER_WAIT: 8243784888
           MIN_TIMER_WAIT: 0
           AVG_TIMER_WAIT: 265928484
           MAX_TIMER_WAIT: 6490658832
...
mysql> SELECT * FROM file_summary_by_instance\G
...
*************************** 2. row ***************************
                FILE_NAME: /var/mysql/share/english/errmsg.sys
               EVENT_NAME: wait/io/file/sql/ERRMSG
               EVENT_NAME: wait/io/file/sql/ERRMSG
    OBJECT_INSTANCE_BEGIN: 4686193384
               COUNT_STAR: 5
           SUM_TIMER_WAIT: 13990154448
           MIN_TIMER_WAIT: 26349624
           AVG_TIMER_WAIT: 2798030607
           MAX_TIMER_WAIT: 8150662536
...
```

TRUNCATE TABLE is permitted for file I/O summary tables. It resets the summary columns to zero rather than removing rows.

The MySQL server uses several techniques to avoid I/O operations by caching information read from files, so it is possible that statements you might expect to result in I/O events will not. You may be able to ensure that I/O does occur by flushing caches or restarting the server to reset its state.

20.9.12.7 Table I/O and Lock Wait Summary Tables

The following sections describe the table I/O and lock wait summary tables:

- table_io_waits_summary_by_index_usage: Table I/O waits per index
- table_io_waits_summary_by_table: Table I/O waits per table
- table_lock_waits_summary_by_table: Table lock waits per table

The table_io_waits_summary_by_table Table

The table_io_waits_summary_by_table table aggregates all table I/O wait events, as generated by the wait/io/table/sql/handler instrument. The grouping is by table.

The table_io_waits_summary_by_table table has these grouping columns to indicate how the table aggregates events: OBJECT_TYPE, OBJECT_SCHEMA, and OBJECT_NAME. These columns have the same meaning as in the events_waits_current table. They identify the table to which the row applies.

table_io_waits_summary_by_table has the following summary columns containing aggregated values. As indicated in the column descriptions, some columns are more general and have values that are the same as the sum of the values of more fine-grained columns. For example, columns that aggregate all writes hold the sum of the corresponding columns that aggregate inserts, updates, and deletes. In this way, aggregations at higher levels are available directly without the need for user-defined views that sum lower-level columns.

- COUNT_STAR, SUM_TIMER_WAIT, MIN_TIMER_WAIT, AVG_TIMER_WAIT, MAX_TIMER_WAIT

These columns aggregate all I/O operations. They are the same as the sum of the corresponding *xxx*_READ and *xxx*_WRITE columns.

- COUNT_READ, SUM_TIMER_READ, MIN_TIMER_READ, AVG_TIMER_READ, MAX_TIMER_READ

  These columns aggregate all read operations. They are the same as the sum of the corresponding *xxx*_FETCH columns.

- COUNT_WRITE, SUM_TIMER_WRITE, MIN_TIMER_WRITE, AVG_TIMER_WRITE, MAX_TIMER_WRITE

  These columns aggregate all write operations. They are the same as the sum of the corresponding *xxx*_INSERT, *xxx*_UPDATE, and *xxx*_DELETE columns.

- COUNT_FETCH, SUM_TIMER_FETCH, MIN_TIMER_FETCH, AVG_TIMER_FETCH, MAX_TIMER_FETCH

  These columns aggregate all fetch operations.

- COUNT_INSERT, SUM_TIMER_INSERT, MIN_TIMER_INSERT, AVG_TIMER_INSERT, MAX_TIMER_INSERT

  These columns aggregate all insert operations.

- COUNT_UPDATE, SUM_TIMER_UPDATE, MIN_TIMER_UPDATE, AVG_TIMER_UPDATE, MAX_TIMER_UPDATE

  These columns aggregate all update operations.

- COUNT_DELETE, SUM_TIMER_DELETE, MIN_TIMER_DELETE, AVG_TIMER_DELETE, MAX_TIMER_DELETE

  These columns aggregate all delete operations.

TRUNCATE TABLE is permitted for table I/O summary tables. It resets the summary columns to zero rather than removing rows. Truncating this table also truncates the table_io_waits_summary_by_index_usage table.

The table_io_waits_summary_by_index_usage Table

The table_io_waits_summary_by_index_usage table aggregates all table index I/O wait events, as generated by the wait/io/table/sql/handler instrument. The grouping is by table index.

The structure of table_io_waits_summary_by_index_usage is nearly identical to table_io_waits_summary_by_table. The only difference is the additional group column, INDEX_NAME, which corresponds to the name of the index that was used when the table I/O wait event was recorded:

- A value of PRIMARY indicates that table I/O used the primary index.
- A value of NULL means that table I/O used no index.
- Inserts are counted against INDEX_NAME = NULL.

TRUNCATE TABLE is permitted for table I/O summary tables. It resets the summary columns to zero rather than removing rows. This table is also truncated by truncation of the table_io_waits_summary_by_table table. A DDL operation that changes the index structure of a table may cause the per-index statistics to be reset.

The table_lock_waits_summary_by_table Table

The table_lock_waits_summary_by_table table aggregates all table lock wait events, as generated by the wait/lock/table/sql/handler instrument. The grouping is by table.

This table contains information about internal and external locks:

- An internal lock corresponds to a lock in the SQL layer. This is currently implemented by a call to thr_lock(). In event rows, these locks are distinguished by the OPERATION column, which will have one of these values:

```
read normal
read with shared locks
read high priority
read no insert
write allow write
write concurrent insert
write delayed
write low priority
write normal
```

- An external lock corresponds to a lock in the storage engine layer. This is currently implemented by a call to handler::external_lock(). In event rows, these locks are distinguished by the OPERATION column, which will have one of these values:

```
read external
write external
```

The table_lock_waits_summary_by_table table has these grouping columns to indicate how the table aggregates events: OBJECT_TYPE, OBJECT_SCHEMA, and OBJECT_NAME. These columns have the same meaning as in the events_waits_current table. They identify the table to which the row applies.

table_lock_waits_summary_by_table has the following summary columns containing aggregated values. As indicated in the column descriptions, some columns are more general and have values that are the same as the sum of the values of more fine-grained columns. For example, columns that aggregate all locks hold the sum of the corresponding columns that aggregate read and write locks. In this way, aggregations at higher levels are available directly without the need for user-defined views that sum lower-level columns.

- COUNT_STAR, SUM_TIMER_WAIT, MIN_TIMER_WAIT, AVG_TIMER_WAIT, MAX_TIMER_WAIT

  These columns aggregate all lock operations. They are the same as the sum of the corresponding *xxx*_READ and *xxx*_WRITE columns.

- COUNT_READ, SUM_TIMER_READ, MIN_TIMER_READ, AVG_TIMER_READ, MAX_TIMER_READ

  These columns aggregate all read-lock operations. They are the same as the sum of the corresponding *xxx*_READ_NORMAL, *xxx*_READ_WITH_SHARED_LOCKS, *xxx*_READ_HIGH_PRIORITY, and *xxx*_READ_NO_INSERT columns.

- COUNT_WRITE, SUM_TIMER_WRITE, MIN_TIMER_WRITE, AVG_TIMER_WRITE, MAX_TIMER_WRITE

  These columns aggregate all write-lock operations. They are the same as the sum of the corresponding *xxx*_WRITE_ALLOW_WRITE, *xxx*_WRITE_CONCURRENT_INSERT, *xxx*_WRITE_LOW_PRIORITY, and *xxx*_WRITE_NORMAL columns.

- COUNT_READ_NORMAL, SUM_TIMER_READ_NORMAL, MIN_TIMER_READ_NORMAL, AVG_TIMER_READ_NORMAL, MAX_TIMER_READ_NORMAL

  These columns aggregate internal read locks.

- COUNT_READ_WITH_SHARED_LOCKS, SUM_TIMER_READ_WITH_SHARED_LOCKS, MIN_TIMER_READ_WITH_SHARED_LOCKS, AVG_TIMER_READ_WITH_SHARED_LOCKS, MAX_TIMER_READ_WITH_SHARED_LOCKS

  These columns aggregate internal read locks.

- COUNT_READ_HIGH_PRIORITY, SUM_TIMER_READ_HIGH_PRIORITY, MIN_TIMER_READ_HIGH_PRIORITY, AVG_TIMER_READ_HIGH_PRIORITY, MAX_TIMER_READ_HIGH_PRIORITY

  These columns aggregate internal read locks.

- COUNT_READ_NO_INSERT, SUM_TIMER_READ_NO_INSERT, MIN_TIMER_READ_NO_INSERT, AVG_TIMER_READ_NO_INSERT, MAX_TIMER_READ_NO_INSERT

  These columns aggregate internal read locks.

- COUNT_READ_EXTERNAL, SUM_TIMER_READ_EXTERNAL, MIN_TIMER_READ_EXTERNAL, AVG_TIMER_READ_EXTERNAL, MAX_TIMER_READ_EXTERNAL

  These columns aggregate external read locks.

- COUNT_WRITE_ALLOW_WRITE, SUM_TIMER_WRITE_ALLOW_WRITE, MIN_TIMER_WRITE_ALLOW_WRITE, AVG_TIMER_WRITE_ALLOW_WRITE, MAX_TIMER_WRITE_ALLOW_WRITE

  These columns aggregate internal write locks.

- COUNT_WRITE_CONCURRENT_INSERT, SUM_TIMER_WRITE_CONCURRENT_INSERT, MIN_TIMER_WRITE_CONCURRENT_INSERT, AVG_TIMER_WRITE_CONCURRENT_INSERT, MAX_TIMER_WRITE_CONCURRENT_INSERT

  These columns aggregate internal write locks.

- COUNT_WRITE_LOW_PRIORITY, SUM_TIMER_WRITE_LOW_PRIORITY, MIN_TIMER_WRITE_LOW_PRIORITY, AVG_TIMER_WRITE_LOW_PRIORITY, MAX_TIMER_WRITE_LOW_PRIORITY

  These columns aggregate internal write locks.

- COUNT_WRITE_NORMAL, SUM_TIMER_WRITE_NORMAL, MIN_TIMER_WRITE_NORMAL, AVG_TIMER_WRITE_NORMAL, MAX_TIMER_WRITE_NORMAL

  These columns aggregate internal write locks.

- COUNT_WRITE_EXTERNAL, SUM_TIMER_WRITE_EXTERNAL, MIN_TIMER_WRITE_EXTERNAL, AVG_TIMER_WRITE_EXTERNAL, MAX_TIMER_WRITE_EXTERNAL

  These columns aggregate external write locks.

TRUNCATE TABLE is permitted for table lock summary tables. It resets the summary columns to zero rather than removing rows.

20.9.12.8 Connection Summary Tables

The connection summary tables are similar to the corresponding events_*xxx*_summary_by_thread_by_event_name tables, except that aggregation occurs per account, user, or host, rather than by thread.

The Performance Schema maintains summary tables that aggregate connection statistics by event name and account, user, or host. Separate groups of tables are available that aggregate wait, stage, and statement events, which results in this set of connection summary tables:

- `events_waits_summary_by_account_by_event_name`: Wait events summarized per account and event name
- `events_waits_summary_by_user_by_event_name`: Wait events summarized per user name and event name
- `events_waits_summary_by_host_by_event_name`: Wait events summarized per host name and event name
- `events_stages_summary_by_account_by_event_name`: Stage events summarized per account and event name
- `events_stages_summary_by_user_by_event_name`: Stage events summarized per user name and event name
- `events_stages_summary_by_host_by_event_name`: Stage events summarized per host name and event name
- `events_statements_summary_by_account_by_event_name`: Statement events summarized per account and event name
- `events_statements_summary_by_user_by_event_name`: Statement events summarized per user name and event name
- `events_statements_summary_by_host_by_event_name`: Statement events summarized per host name and event name

In other words, the connection summary tables have names of the form `events_`*`xxx`*`_summary_`*`yyy`*`_by_event_name`, where *`xxx`* is `waits`, `stages`, or `statements`, and *`yyy`* is `account`, `user`, or `host`.

The connection summary tables provide an intermediate aggregation level:

- *`xxx`*`_summary_by_thread_by_event_name` tables are more detailed than connection summary tables
- *`xxx`*`_summary_global_by_event_name` tables are less detailed than connection summary tables

Each connection summary table has one or more grouping columns to indicate how the table aggregates events. Event names refer to names of event instruments in the `setup_instruments` table.

- For tables with `_by_account` in the name, the `USER`, `HOST`, and `EVENT_NAME` columns group events per account and event name.
- For tables with `_by_host` in the name, the `HOST` and `EVENT_NAME` columns group events per host name and event name.
- For tables with `_by_user` in the name, the `USER` and `EVENT_NAME` columns group events per user name and event name.

All connection summary tables have these summary columns containing aggregated values: `COUNT_STAR`, `SUM_TIMER_WAIT`, `MIN_TIMER_WAIT`, `AVG_TIMER_WAIT`, and `MAX_TIMER_WAIT`. These are similar to the columns of the same names in the `events_waits_summary_by_instance` table. Connection summary tables for statements have additional `SUM_`*`xxx`* columns that aggregate statement types.

TRUNCATE TABLE is permitted for connection summary tables. It resets the summary columns to zero rather than removing rows. In addition, connection summary tables are implicitly truncated if a connection table on which they depend is truncated. Table 20.2, “Effect of Implicit Table Truncation”, describes the relationship between connection table truncation and implicitly truncated tables.

Table 20.2 Effect of Implicit Table Truncation

Truncated Table	Implicitly Truncated Summary Tables
accounts	Tables with names matching %_by_account%, %_by_thread%
hosts	Tables with names matching %_by_account%, %_by_host%, %_by_thread%
users	Tables with names matching %_by_account%, %_by_user%, %_by_thread%

20.9.12.9 Socket Summary Tables

These socket summary tables aggregate timer and byte count information for socket operations:

- socket_summary_by_instance: Aggregate timer and byte count statistics generated by the wait/io/socket/* instruments for all socket I/O operations, per socket instance. When a connection terminates, the row in socket_summary_by_instance corresponding to it is deleted.

- socket_summary_by_event_name: Aggregate timer and byte count statistics generated by the wait/io/socket/* instruments for all socket I/O operations, per socket instrument.

The socket summary tables do not aggregate waits generated by idle events while sockets are waiting for the next request from the client. For idle event aggregations, use the wait-event summary tables; see Section 20.9.12.1, “Event Wait Summary Tables”.

Each socket summary table has one or more grouping columns to indicate how the table aggregates events. Event names refer to names of event instruments in the setup_instruments table.

- socket_summary_by_instance has an OBJECT_INSTANCE_BEGIN column. Each row summarizes events for a given object.

- socket_summary_by_event_name has an EVENT_NAME column. Each row summarizes events for a given event name.

All socket summary tables have these summary columns containing aggregated values:

- COUNT_STAR, SUM_TIMER_WAIT, MIN_TIMER_WAIT, AVG_TIMER_WAIT, MAX_TIMER_WAIT

  These columns aggregate all operations.

- COUNT_READ, SUM_TIMER_READ, MIN_TIMER_READ, AVG_TIMER_READ, MAX_TIMER_READ, SUM_NUMBER_OF_BYTES_READ

  These columns aggregate all receive operations (RECV, RECVFROM, and RECVMSG).

- COUNT_WRITE, SUM_TIMER_WRITE, MIN_TIMER_WRITE, AVG_TIMER_WRITE, MAX_TIMER_WRITE, SUM_NUMBER_OF_BYTES_WRITE

  These columns aggregate all send operations (SEND, SENDTO, and SENDMSG).

- COUNT_MISC, SUM_TIMER_MISC, MIN_TIMER_MISC, AVG_TIMER_MISC, MAX_TIMER_MISC

These columns aggregate all other socket operations, such as CONNECT, LISTEN, ACCEPT, CLOSE, and SHUTDOWN. There are no byte counts for these operations.

The socket_summary_by_instance table also has an EVENT_NAME column that indicates the class of the socket: client_connection, server_tcpip_socket, server_unix_socket. This column can be grouped on to isolate, for example, client activity from that of the server listening sockets.

TRUNCATE TABLE is permitted for socket summary tables. Except for events_statements_summary_by_digest, tt resets the summary columns to zero rather than removing rows.

20.9.12.10 Memory Summary Tables

The Performance Schema instruments memory usage and aggregates memory usage statistics, detailed by these factors:

- Type of memory used (various caches, internal buffers, and so forth)

- Thread, account, user, host indirectly performing the memory operation

The Performance Schema instruments the following aspects of memory use

- Memory sizes used

- Operation counts

- Low and high water marks

Memory sizes help to understand or tune the memory consumption of a server.

Operation counts help to understand or tune the overall pressure the server is putting on the memory allocator, which has an impact on performance. Allocating a single byte one million times is not the same as allocating one million bytes a single time; tracking both sizes and counts can expose the difference.

Low and high water marks are critical to detect workload spikes, overall workload stability, and possible memory leaks.

Each memory summary table has one or more grouping columns to indicate how the table aggregates events. Event names refer to names of event instruments in the setup_instruments table.

- memory_summary_by_account_by_event_name has USER, HOST, and EVENT_NAME columns. Each row summarizes events for a given account.

- memory_summary_by_host_by_event_name has HOST and EVENT_NAME columns. Each row summarizes events for a given host.

- memory_summary_by_thread_by_event_name has THREAD_ID and EVENT_NAME columns. Each row summarizes events for a given thread and event name.

- memory_summary_by_user_by_event_name has USER and EVENT_NAME columns. Each row summarizes events for a given user.

- memory_summary_global_by_event_name has an EVENT_NAME column. Each row summarizes events for a given event name.

All memory summary tables have these summary columns containing aggregated values:

- COUNT_ALLOC, COUNT_FREE

  These columns aggregate the number of calls to malloc-like and free-like functions.

- SUM_NUMBER_OF_BYTES_ALLOC, SUM_NUMBER_OF_BYTES_FREE

  These columns indicate the aggregate size of allocated and freed memory blocks.

- CURRENT_COUNT_USED

  This column is the aggregate number of currently allocated blocks that have not been freed yet. This is a convenience column, equal to COUNT_ALLOC – COUNT_FREE.

- CURRENT_NUMBER_OF_BYTES_USED

  This column is the aggregate size of currently allocated memory blocks that have not been freed yet. This is a convenience column, equal to SUM_NUMBER_OF_BYTES_ALLOC – SUM_NUMBER_OF_BYTES_FREE.

- LOW_COUNT_USED, HIGH_COUNT_USED

  These columns are the low and high water marks corresponding to the CURRENT_COUNT_USED column.

- LOW_NUMBER_OF_BYTES_USED, HIGH_NUMBER_OF_BYTES_USED

  These columns are the low and high water marks corresponding to the CURRENT_NUMBER_OF_BYTES_USED column.

Example memory event summary information:

```
mysql> SELECT * FROM memory_summary_global_by_event_name
    -> WHERE EVENT_NAME = 'memory/sql/TABLE'\G
*************************** 1. row ***************************
                  EVENT_NAME: memory/sql/TABLE
                 COUNT_ALLOC: 1381
                  COUNT_FREE: 924
   SUM_NUMBER_OF_BYTES_ALLOC: 2059873
    SUM_NUMBER_OF_BYTES_FREE: 1407432
              LOW_COUNT_USED: 0
          CURRENT_COUNT_USED: 457
             HIGH_COUNT_USED: 461
    LOW_NUMBER_OF_BYTES_USED: 0
CURRENT_NUMBER_OF_BYTES_USED: 652441
   HIGH_NUMBER_OF_BYTES_USED: 669269
```

TRUNCATE TABLE is permitted for memory summary tables. It has these effects:

- In general, truncation resets the baseline for statistics, but does not change the server state. That is, truncating a memory table does not free memory.
- COUNT_ALLOC and COUNT_FREE are reset to a new baseline, by reducing each counter by the same value.
- Likewise, SUM_NUMBER_OF_BYTES_ALLOC and SUM_NUMBER_OF_BYTES_FREE are reset to a new baseline.
- LOW_COUNT_USED and HIGH_COUNT_USED are reset to CURRENT_COUNT_USED.
- LOW_NUMBER_OF_BYTES_USED and HIGH_NUMBER_OF_BYTES_USED are reset to CURRENT_NUMBER_OF_BYTES_USED.

Memory Instrumentation Behavior

Memory instrumentation is disabled by default, and can be enabled or disabled dynamically by updating the `ENABLED` column of the relevant instruments in the `setup_instruments` table. Memory instruments have names of the form `memory/code_area/instrument_name`.

For memory instruments, the `TIMED` column in `setup_instruments` is ignored because memory operations are not timed.

When a thread in the server executes a memory allocation that has been instrumented, these rules apply:

- If the thread is not instrumented or the memory instrument is not enabled, the memory block allocated is not instrumented.
- Otherwise (that is, both the thread and the instrument are enabled), the memory block allocated is instrumented.

For deallocation, these rules apply:

- If a thread is instrumented, and a memory block is not instrumented, the free operation is not instrumented; no statistics are changed.
- If a thread is not instrumented, and a memory block is instrumented, the free operation is instrumented, and statistics are changed.

For the per-thread statistics, the following rules apply.

When an instrumented memory block of size *`N`* is allocated, the Performance Schema makes these updates to memory summary table columns:

- `COUNT_ALLOC`: Incremented by 1
- `CURRENT_COUNT_USED`: Incremented by 1
- `HIGH_COUNT_USED`: Increased if `CURRENT_COUNT_USED` is a new maximum
- `SUM_NUMBER_OF_BYTES_ALLOC`: Increased by *`N`*
- `CURRENT_NUMBER_OF_BYTES_USED`: Increased by *`N`*
- `HIGH_NUMBER_OF_BYTES_USED`: Increased if `CURRENT_NUMBER_OF_BYTES_USED` is a new maximum

When an instrumented memory block is deallocated, the Performance Schema makes these updates to memory summary table columns:

- `COUNT_FREE`: Incremented by 1
- `CURRENT_COUNT_USED`: Iecremented by 1
- `LOW_COUNT_USED`: Decreased if `CURRENT_COUNT_USED` is a new minimum
- `SUM_NUMBER_OF_BYTES_FREE`: Increased by *`N`*
- `CURRENT_NUMBER_OF_BYTES_USED`: Decreased by *`N`*
- `LOW_NUMBER_OF_BYTES_USED`: Decreased if `CURRENT_NUMBER_OF_BYTES_USED` is a new minimum

For higher-level aggregates (global, by account, by user, by host), the same rules apply as expected for low and high water marks.

- LOW_COUNT_USED and LOW_NUMBER_OF_BYTES_USED are lower estimates
- HIGH_COUNT_USED and HIGH_NUMBER_OF_BYTES_USED are higher estimates

“Lower estimates” means that the value reported by the Performance Schema is guaranteed to be less than or equal to the lowest count or size of memory effectively used at runtime.

“Higher estimates” means that the value reported by the Performance Schema is guaranteed to be greater than or equal to the highest count or size of memory effectively used at runtime.

For lower estimates in summary tables other than memory_summary_global_by_event_name, it is possible for values to go negative if memory ownership is transferred between threads.

Here is an example of estimate computation; but note that estimate implementation is subject to change:

Thread 1 uses memory in the range from 1MB to 2MB during execution, as reported by the LOW_NUMBER_OF_BYTES_USED and HIGH_NUMBER_OF_BYTES_USED columns of the memory_summary_by_thread_by_event_name table.

Thread 2 uses memory in the range from 10MB to 12MB during execution, as reported likewise.

When these two threads belong to the same user account, the per-account summary estimates that this account used memory in the range from 11MB to 14MB. That is, the LOW_NUMBER_OF_BYTES_USED for the higher level aggregate is the sum of each LOW_NUMBER_OF_BYTES_USED (assuming the worst case). Likewise, the HIGH_NUMBER_OF_BYTES_USED for the higher level aggregate is the sum of each HIGH_NUMBER_OF_BYTES_USED (assuming the worst case).

11MB is a lower estimate that can occur only if both threads hit the low usage mark at the same time.

14MB is a higher estimate that can occur only if both threads hit the high usage mark at the same time.

The real memory usage for this account could have been in the range from 11.5MB to 13.5MB.

For capacity planning, reporting the worst case is actually the desired behavior, as it shows what can potentially happen when sessions are uncorrelated, which is typically the case.

20.9.13 Performance Schema Miscellaneous Tables

The following sections describe tables that do not fall into the table categories discussed in the preceding sections:

- host_cache: Information from the internal host cache
- performance_timers: Which event timers are available
- threads: Information about server threads

20.9.13.1 The host_cache Table

The host_cache table provides access to the contents of the host cache, which contains client host name and IP address information and is used to avoid DNS lookups. (See Section 8.11.5.2, “DNS Lookup Optimization and the Host Cache”.) The host_cache table exposes the contents of the host cache so that it can be examined using SELECT statements. The Performance Schema must be enabled or this table is empty.

FLUSH HOSTS and TRUNCATE TABLE host_cache have the same effect: They clear the host cache. This also empties the host_cache table (because it is the visible representation of the cache) and unblocks any blocked hosts (see Section C.5.2.6, “Host '*host_name*' is blocked”.) FLUSH HOSTS requires the RELOAD privilege. TRUNCATE TABLE requires the DROP privilege for the host_cache table.

The host_cache table has these columns:

- IP

  The IP address of the client that connected to the server, expressed as a string.

- HOST

  The resolved DNS host name for that client IP, or NULL if the name is unknown.

- HOST_VALIDATED

  Whether the IP-to-host name-to-IP DNS resolution was performed successfully for the client IP. If HOST_VALIDATED is YES, the HOST column is used as the host name corresponding to the IP so that calls to DNS can be avoided. While HOST_VALIDATED is NO, DNS resolution is attempted again for each connect, until it eventually completes with either a valid result or a permanent error. This information enables the server to avoid caching bad or missing host names during temporary DNS failures, which would affect clients forever.

- SUM_CONNECT_ERRORS

  The number of connection errors that are deemed "blocking" (assessed against the max_connect_errors system variable). Currently, only protocol handshake errors are counted, and only for hosts that passed validation (HOST_VALIDATED = YES).

- COUNT_HOST_BLOCKED_ERRORS

  The number of connections that were blocked because SUM_CONNECT_ERRORS exceeded the value of the max_connect_errors system variable.

- COUNT_NAMEINFO_TRANSIENT_ERRORS

  The number of transient errors during IP-to-host name DNS resolution.

- COUNT_NAMEINFO_PERMANENT_ERRORS

  The number of permanent errors during IP-to-host name DNS resolution.

- COUNT_FORMAT_ERRORS

  The number of host name format errors. MySQL does not perform matching of Host column values in the mysql.user table against host names for which one or more of the initial components of the name are entirely numeric, such as 1.2.example.com. The client IP address is used instead. For the rationale why this type of matching does not occur, see Section 6.2.3, "Specifying Account Names".

- COUNT_ADDRINFO_TRANSIENT_ERRORS

  The number of transient errors during host name-to-IP reverse DNS resolution.

- COUNT_ADDRINFO_PERMANENT_ERRORS

  The number of permanent errors during host name-to-IP reverse DNS resolution.

- COUNT_FCRDNS_ERRORS

  The number of forward-confirmed reverse DNS errors. These errors occur when IP-to-host name-to-IP DNS resolution produces an IP address that does not match the client originating IP address.

- COUNT_HOST_ACL_ERRORS

The number of errors that occur because no user from the client host can possibly log in. In such cases, the server returns ER_HOST_NOT_PRIVILEGED and does not even ask for a user name or password.

- COUNT_NO_AUTH_PLUGIN_ERRORS

  The number of errors due to requests for an unavailable authentication plugin. A plugin can be unavailable if, for example, it was never loaded or a load attempt failed.

- COUNT_AUTH_PLUGIN_ERRORS

  The number of errors reported by authentication plugins.

  An authentication plugin can report different error codes to indicate the root cause of a failure. Depending on the type of error, one of these columns is incremented: COUNT_AUTHENTICATION_ERRORS, COUNT_AUTH_PLUGIN_ERRORS, COUNT_HANDSHAKE_ERRORS. New return codes are an optional extension to the existing plugin API. Unknown or unexpected plugin errors are counted in the COUNT_AUTH_PLUGIN_ERRORS column.

- COUNT_HANDSHAKE_ERRORS

  The number of errors detected at the wire protocol level.

- COUNT_PROXY_USER_ERRORS

  The number of errors detected when a proxy user A is proxied to another user B who does not exist.

- COUNT_PROXY_USER_ACL_ERRORS

  The number of errors detected when a proxy user A is proxied to another user B who does exist but for whom A does not have the PROXY privilege.

- COUNT_AUTHENTICATION_ERRORS

  The number of errors caused by failed authentication.

- COUNT_SSL_ERRORS

  The number of errors due to SSL problems.

- COUNT_MAX_USER_CONNECTIONS_ERRORS

  The number of errors caused by exceeding per-user connection quotas. See Section 6.3.4, “Setting Account Resource Limits”.

- COUNT_MAX_USER_CONNECTIONS_PER_HOUR_ERRORS

  The number of errors caused by exceeding per-user connections-per-hour quotas. See Section 6.3.4, “Setting Account Resource Limits”.

- COUNT_DEFAULT_DATABASE_ERRORS

  The number of errors related to the default database. For example, the database did not exist or the user had no privileges for accessing it.

- COUNT_INIT_CONNECT_ERRORS

  The number of errors caused by execution failures of statements in the init_connect system variable value.

- COUNT_LOCAL_ERRORS

  The number of errors local to the server implementation and not related to the network, authentication, or authorization. For example, out-of-memory conditions fall into this category.

- COUNT_UNKNOWN_ERRORS

  The number of other, unknown errors not accounted for by other columns in this table. This column is reserved for future use, in case new error conditions must be reported, and if preserving the backward compatibility and table structure of the host_cache table is required.

- FIRST_SEEN

  The timestamp of the first connection attempt seen from the client in the IP column.

- LAST_SEEN

  The timestamp of the last connection attempt seen from the client in the IP column.

- FIRST_ERROR_SEEN

  The timestamp of the first error seen from the client in the IP column.

- LAST_ERROR_SEEN

  The timestamp of the last error seen from the client in the IP column.

20.9.13.2 The performance_timers Table

The performance_timers table shows which event timers are available:

```
mysql> SELECT * FROM performance_timers;
+-------------+-----------------+------------------+----------------+
| TIMER_NAME  | TIMER_FREQUENCY | TIMER_RESOLUTION | TIMER_OVERHEAD |
+-------------+-----------------+------------------+----------------+
| CYCLE       |      2389029850 |                1 |             72 |
| NANOSECOND  |            NULL |             NULL |           NULL |
| MICROSECOND |         1000000 |                1 |            585 |
| MILLISECOND |            1035 |                1 |            738 |
| TICK        |             101 |                1 |            630 |
+-------------+-----------------+------------------+----------------+
```

If the values associated with a given timer name are NULL, that timer is not supported on your platform. The rows that do not contain NULL indicate which timers you can use in setup_timers.

The performance_timers table has these columns:

- TIMER_NAME

  The name by which to refer to the timer when configuring the setup_timers table.

- TIMER_FREQUENCY

  The number of timer units per second. For a cycle timer, the frequency is generally related to the CPU speed. For example, on a system with a 2.4GHz processor, the CYCLE may be close to 2400000000.

- TIMER_RESOLUTION

  Indicates the number of timer units by which timer values increase. If a timer has a resolution of 10, its value increases by 10 each time.

- TIMER_OVERHEAD

  The minimal number of cycles of overhead to obtain one timing with the given timer. The Performance Schema determines this value by invoking the timer 20 times during initialization and picking the smallest value. The total overhead really is twice this amount because the instrumentation invokes the timer at the start and end of each event. The timer code is called only for timed events, so this overhead does not apply for nontimed events.

20.9.13.3 The threads Table

The threads table contains a row for each server thread. Each row contains information about a thread and indicates whether monitoring is enabled for it:

```
mysql> SELECT * FROM threads\G
*************************** 1. row ***************************
          THREAD_ID: 1
               NAME: thread/sql/main
               TYPE: BACKGROUND
     PROCESSLIST_ID: NULL
   PROCESSLIST_USER: NULL
   PROCESSLIST_HOST: NULL
     PROCESSLIST_DB: NULL
PROCESSLIST_COMMAND: NULL
   PROCESSLIST_TIME: 80284
  PROCESSLIST_STATE: NULL
   PROCESSLIST_INFO: NULL
   PARENT_THREAD_ID: NULL
               ROLE: NULL
       INSTRUMENTED: YES
...
*************************** 4. row ***************************
          THREAD_ID: 51
               NAME: thread/sql/one_connection
               TYPE: FOREGROUND
     PROCESSLIST_ID: 34
   PROCESSLIST_USER: paul
   PROCESSLIST_HOST: localhost
     PROCESSLIST_DB: performance_schema
PROCESSLIST_COMMAND: Query
   PROCESSLIST_TIME: 0
  PROCESSLIST_STATE: Sending data
   PROCESSLIST_INFO: SELECT * FROM threads
   PARENT_THREAD_ID: 1
               ROLE: NULL
       INSTRUMENTED: YES
...
```

The initial contents of the threads table are based on the threads in existence when Performance Schema initialization occurs. Thereafter, a new row is added each time the server creates a thread.

Removal of rows from the threads table occurs when threads end. For a thread associated with a client session, removal occurs when the session ends. If a client has auto-reconnect enabled and the session reconnects after a disconnect, the session will be associated with a new row in the threads table that has a different PROCESSLIST_ID value. The initial INSTRUMENTED value for the new thread may be different from that of the original thread: The setup_actors table may have changed in the meantime, and if the INSTRUMENTED value for the original thread was changed after it was initialized, that change does not carry over to the new thread.

The threads table columns with names having a prefix of PROCESSLIST_ provide information similar to that available from the INFORMATION_SCHEMA.PROCESSLIST table or the SHOW PROCESSLIST statement. Thus, all three sources provide thread-monitoring information. Use of threads differs from use of the other two sources in these ways:

- Access to threads does not require a mutex and has minimal impact on server performance. INFORMATION_SCHEMA.PROCESSLIST and SHOW PROCESSLIST have negative performance consequences because they require a mutex.
- threads provides additional information for each thread, such as whether it is a foreground or background thread, and the location within the server associated with the thread.
- threads provides information about background threads, so it can be used to monitor activity the other thread information sources cannot.
- You can enable or disable thread monitoring (that is, whether events executed by the thread are instrumented). To control monitoring of existing threads, set the INSTRUMENTED column of the threads table. To control the initial INSTRUMENTED value for new foreground threads, use the setup_actors table. (For more information about the conditions under which thread monitoring occurs, see the description of the INSTRUMENTED column.)

For these reasons, DBAs who perform server monitoring using INFORMATION_SCHEMA.PROCESSLIST or SHOW PROCESSLIST may wish to monitor using threads instead.

Note

For INFORMATION_SCHEMA.PROCESSLIST and SHOW PROCESSLIST, information about threads for other users is shown only if the current user has the PROCESS privilege. That is not true of the threads table; all rows are shown to any user who has the SELECT privilege for the table. Users who should not be able to see threads for other users should not be given that privilege.

The threads table has these columns:

- THREAD_ID

  A unique thread identifier.

- NAME

  The name associated with the thread instrumentation code in the server. For example, thread/sql/one_connection corresponds to the thread function in the code responsible for handling a user connection, and thread/sql/main stands for the main() function of the server.

- TYPE

  The thread type, either FOREGROUND or BACKGROUND. User connection threads are foreground threads. Threads associated with internal server activity are background threads. Examples are internal InnoDB threads, “binlog dump” threads sending information to slaves, and slave I/O and SQL threads.

- PROCESSLIST_ID

  For threads that are displayed in the INFORMATION_SCHEMA.PROCESSLIST table, this is the PROCESSLIST.ID value, which is also the value that CONNECTION_ID() would return within that thread. For background threads (threads not associated with a user connection), PROCESSLIST_ID is NULL, so the values are not unique.

- PROCESSLIST_USER

  The user associated with a foreground thread, NULL for a background thread.

- PROCESSLIST_HOST

  The host name of the client associated with a foreground thread, NULL for a background thread.

- PROCESSLIST_DB

  The default database for the thread, or NULL if there is none.

- PROCESSLIST_COMMAND

  The type of command the thread is executing. For descriptions of thread commands, see Section 8.12.5, "Examining Thread Information". The value of this column corresponds to the COM_*xxx* commands of the client/server protocol and Com_*xxx* status variables. See Section 5.1.6, "Server Status Variables"

- PROCESSLIST_TIME

  The time in seconds that the thread has been in its current state.

- PROCESSLIST_STATE

  An action, event, or state that indicates what the thread is doing. For descriptions of PROCESSLIST_STATE values, see Section 8.12.5, "Examining Thread Information".

  Most states correspond to very quick operations. If a thread stays in a given state for many seconds, there might be a problem that bears investigation.

- PROCESSLIST_INFO

  The statement the thread is executing, or NULL if it is not executing any statement. The statement might be the one sent to the server, or an innermost statement if the statement executes other statements. For example, if a CALL statement executes a stored procedure that is executing a SELECT statement, the PROCESSLIST_INFO value shows the SELECT statement.

- PARENT_THREAD_ID

  If this thread is a subthread (spawned by another thread), this is the THREAD_ID value of the spawning thread.

- ROLE

  Unused.

- INSTRUMENTED

  Whether the thread is instrumented. This does not affect the threads table row for the thread, it affects whether events executed by the thread are instrumented.

  - For foreground threads, the initial INSTRUMENTED value is determined by whether the user account associated with the thread matches any row in the setup_actors table. Matching is based on the values of the PROCESSLIST_USER and PROCESSLIST_HOST columns.

    If the thread spawns a subthread, matching occurs again for the subthread.

  - For background threads, INSTRUMENTED is YES by default. setup_actors is not consulted because there is no associated user for background threads.

  - For any thread, its INSTRUMENTED value can be changed during the lifetime of the thread. This is the only threads table column that can be modified.

For monitoring of events executed by the thread to occur, these things must be true:

- The `thread_instrumentation` consumer in the `setup_consumers` table must be `YES`.
- The `thread.INSTRUMENTED` column must be `YES`.
- Monitoring occurs only for those thread events produced from instruments that are enabled in the `setup_instruments` table.

20.10 Performance Schema Option and Variable Reference

Table 20.3 Performance Schema Variable Reference

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
performance_schema	Yes	Yes	Yes		Global	No
Performance_schema_accounts_lost				Yes	Global	No
performance_schema_accounts_size	Yes	Yes	Yes		Global	No
Performance_schema_cond_classes_lost				Yes	Global	No
Performance_schema_cond_instances_lost				Yes	Global	No
performance-schema-consumer-events-stages-current	Yes	Yes				
performance-schema-consumer-events-stages-history	Yes	Yes				
performance-schema-consumer-events-stages-history-long	Yes	Yes				
performance-schema-consumer-events-statements-current	Yes	Yes				
performance-schema-consumer-events-statements-history	Yes	Yes				
performance-schema-consumer-events-statements-history-long	Yes	Yes				
performance-schema-	Yes	Yes				

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
consumer-events-transactions-current						
performance-schema-consumer-events-transactions-history	Yes	Yes				
performance-schema-consumer-events-transactions-history-long	Yes	Yes				
performance-schema-consumer-events-waits-current	Yes	Yes				
performance-schema-consumer-events-waits-history	Yes	Yes				
performance-schema-consumer-events-waits-history-long	Yes	Yes				
performance-schema-consumer-global-instrumentation	Yes	Yes				
performance-schema-consumer-statements-digest	Yes	Yes				
performance-schema-consumer-thread-instrumentation	Yes	Yes				
Performance_schema_digest_lost				Yes	Global	No
performance_schema_digests_size	Yes	Yes	Yes		Global	No
performance_schema_events_stages_history_long_size	Yes	Yes	Yes		Global	No
performance_schema_events_stages_history_size	Yes	Yes	Yes		Global	No
performance_schema_events_statements_history_long_size	Yes	Yes	Yes		Global	No
performance_schema_events_statements_history_size	Yes	Yes	Yes		Global	No
performance_schema_events_transactions_history_long_size	Yes	Yes	Yes		Global	No
performance_schema_events_transactions_history_size	Yes	Yes	Yes		Global	No
performance_schema_events_waits_history_long_size	Yes	Yes	Yes		Global	No

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
performance_schema_events_waits_history_size	Yes	Yes	Yes		Global	No
Performance_schema_file_classes_lost				Yes	Global	No
Performance_schema_file_handles_lost				Yes	Global	No
Performance_schema_file_instances_lost				Yes	Global	No
Performance_schema_hosts_lost				Yes	Global	No
performance_schema_hosts_size	Yes	Yes	Yes		Global	No
performance-schema-instrument	Yes	Yes				
Performance_schema_locker_lost				Yes	Global	No
performance_schema_max_cond_classes	Yes	Yes	Yes		Global	No
performance_schema_max_cond_instances	Yes	Yes	Yes		Global	No
performance_schema_max_file_classes	Yes	Yes	Yes		Global	No
performance_schema_max_file_handles	Yes	Yes	Yes		Global	No
performance_schema_max_file_instances	Yes	Yes	Yes		Global	No
performance_schema_max_memory_classes	Yes	Yes	Yes		Global	No
performance_schema_max_metadata_locks	Yes	Yes	Yes		Global	No
performance_schema_max_mutex_classes	Yes	Yes	Yes		Global	No
performance_schema_max_mutex_instances	Yes	Yes	Yes		Global	No
performance_schema_max_prepared_statements_instances	Yes	Yes	Yes		Global	No
performance_schema_max_program_instances	Yes	Yes	Yes		Global	No
performance_schema_max_rwlock_classes	Yes	Yes	Yes		Global	No
performance_schema_max_rwlock_instances	Yes	Yes	Yes		Global	No
performance_schema_max_socket_classes	Yes	Yes	Yes		Global	No
performance_schema_max_socket_instances	Yes	Yes	Yes		Global	No
performance_schema_max_stage_classes	Yes	Yes	Yes		Global	No
performance_schema_max_statement_classes	Yes	Yes	Yes		Global	No
performance_schema_max_statement_stack	Yes	Yes	Yes		Global	No
performance_schema_max_table_handles	Yes	Yes	Yes		Global	No
performance_schema_max_table_instances	Yes	Yes	Yes		Global	No
performance_schema_max_thread_classes	Yes	Yes	Yes		Global	No
performance_schema_max_thread_instances	Yes	Yes	Yes		Global	No
Performance_schema_memory_classes_lost				Yes	Global	No
Performance_schema_metadata_lock_lost				Yes	Global	No
Performance_schema_mutex_classes_lost				Yes	Global	No
Performance_schema_mutex_instances_lost				Yes	Global	No
Performance_schema_nested_statement_lost				Yes	Global	No
Performance_schema_prepared_statements_lost				Yes	Global	No

Name	Cmd-Line	Option file	System Var	Status Var	Var Scope	Dynamic
Performance_schema_program_lost				Yes	Global	No
Performance_schema_rwlock_classes_lost				Yes	Global	No
Performance_schema_rwlock_instances_lost				Yes	Global	No
Performance_schema_session_connect_attrs_lost				Yes	Global	No
performance_schema_session_connect_attrs_size	Yes	Yes	Yes		Global	No
performance_schema_setup_actors_size	Yes	Yes	Yes		Global	No
performance_schema_setup_objects_size	Yes	Yes	Yes		Global	No
Performance_schema_socket_classes_lost				Yes	Global	No
Performance_schema_socket_instances_lost				Yes	Global	No
Performance_schema_stage_classes_lost				Yes	Global	No
Performance_schema_statement_classes_lost				Yes	Global	No
Performance_schema_table_handles_lost				Yes	Global	No
Performance_schema_table_instances_lost				Yes	Global	No
Performance_schema_thread_classes_lost				Yes	Global	No
Performance_schema_thread_instances_lost				Yes	Global	No
Performance_schema_users_lost				Yes	Global	No
performance_schema_users_size	Yes	Yes	Yes		Global	No

20.11 Performance Schema Command Options

Performance Schema parameters can be specified at server startup on the command line or in option files to configure Performance Schema instruments and consumers. Runtime configuration is also possible in many cases (see Section 20.2.3, “Performance Schema Runtime Configuration”), but startup configuration must be used when runtime configuration is too late to affect instruments that have already been initialized during the startup process.

Performance Schema consumers and instruments can be configured at startup using the following syntax. For additional details, see Section 20.2.2, “Performance Schema Startup Configuration”.

- `--performance-schema-consumer-*consumer_name*=value`

  Configure a Performance Schema consumer. Consumer names in the `setup_consumers` table use underscores, but for consumers set at startup, dashes and underscores within the name are equivalent. Options for configuring individual consumers are detailed later in this section.

- `--performance-schema-instrument=*instrument_name*=value`

  Configure a Performance Schema instrument. The name may be given as a pattern to configure instruments that match the pattern.

The following items configure individual consumers:

- `--performance-schema-consumer-events-stages-current=value`

  Configure the `events-stages-current` consumer.

- `--performance-schema-consumer-events-stages-history=value`

Configure the `events-stages-history` consumer.

- `--performance-schema-consumer-events-stages-history-long=value`

  Configure the `events-stages-history-long` consumer.

- `--performance-schema-consumer-events-statements-current=value`

  Configure the `events-statements-current` consumer.

- `--performance-schema-consumer-events-statements-history=value`

  Configure the `events-statements-history` consumer.

- `--performance-schema-consumer-events-statements-history-long=value`

  Configure the `events-statements-history-long` consumer.

- `--performance-schema-consumer-events-transactions-current=value`

  Configure the Performance Schema `events-transactions-current` consumer. This option was added in MySQL 5.7.3.

- `--performance-schema-consumer-events-transactions-history=value`

  Configure the Performance Schema `events-transactions-history` consumer. This option was added in MySQL 5.7.3.

- `--performance-schema-consumer-events-transactions-history-long=value`

  Configure the Performance Schema `events-transactions-history-long` consumer. This option was added in MySQL 5.7.3.

- `--performance-schema-consumer-events-waits-current=value`

  Configure the `events-waits-current` consumer.

- `--performance-schema-consumer-events-waits-history=value`

  Configure the `events-waits-history` consumer.

- `--performance-schema-consumer-events-waits-history-long=value`

  Configure the `events-waits-history-long` consumer.

- `--performance-schema-consumer-global-instrumentation=value`

  Configure the `global-instrumentation` consumer.

- `--performance-schema-consumer-statements-digest=value`

  Configure the `statements-digest` consumer.

- `--performance-schema-consumer-thread-instrumentation=value`

  Configure the `thread-instrumentation` consumer.

20.12 Performance Schema System Variables

The Performance Schema implements several system variables that provide configuration information:

```
mysql> SHOW VARIABLES LIKE 'perf%';
+--------------------------------------------------------+---------+
| Variable_name                                          | Value   |
+--------------------------------------------------------+---------+
| performance_schema                                     | ON      |
| performance_schema_accounts_size                       | 100     |
| performance_schema_digests_size                        | 200     |
| performance_schema_events_stages_history_long_size     | 10000   |
| performance_schema_events_stages_history_size          | 10      |
| performance_schema_events_statements_history_long_size | 10000   |
| performance_schema_events_statements_history_size      | 10      |
| performance_schema_events_waits_history_long_size      | 10000   |
| performance_schema_events_waits_history_size           | 10      |
| performance_schema_hosts_size                          | 100     |
| performance_schema_max_cond_classes                    | 80      |
| performance_schema_max_cond_instances                  | 1000    |
| performance_schema_max_file_classes                    | 50      |
| performance_schema_max_file_handles                    | 32768   |
| performance_schema_max_file_instances                  | 10000   |
| performance_schema_max_mutex_classes                   | 200     |
| performance_schema_max_mutex_instances                 | 1000000 |
| performance_schema_max_rwlock_classes                  | 30      |
| performance_schema_max_rwlock_instances                | 1000000 |
| performance_schema_max_socket_classes                  | 10      |
| performance_schema_max_socket_instances                | 1000    |
| performance_schema_max_stage_classes                   | 150     |
| performance_schema_max_statement_classes               | 165     |
| performance_schema_max_table_handles                   | 10000   |
| performance_schema_max_table_instances                 | 1000    |
| performance_schema_max_thread_classes                  | 50      |
| performance_schema_max_thread_instances                | 1000    |
| performance_schema_session_connect_attrs_size          | 512     |
| performance_schema_setup_actors_size                   | 100     |
| performance_schema_setup_objects_size                  | 100     |
| performance_schema_users_size                          | 100     |
+--------------------------------------------------------+---------+
```

Performance Schema system variables can be set at server startup on the command line or in option files, and many can be set at runtime. See Section 20.10, “Performance Schema Option and Variable Reference”.

The Performance Schema automatically sizes the values of several of its parameters at server startup if they are not set explicitly. For more information, see Section 20.2.2, “Performance Schema Startup Configuration”.

Performance Schema system variables have the following meanings:

- `performance_schema`

Command-Line Format	`--performance_schema=#`
Option-File Format	`performance_schema`
System Variable Name	`performance_schema`
Variable Scope	Global
Dynamic Variable	No
	Permitted Values

	Type	boolean
	Default	ON

The value of this variable is ON or OFF to indicate whether the Performance Schema is enabled. By default, the value is ON by default. At server startup, you can specify this variable with no value or a value of ON or 1 to enable it, or with a value of OFF or 0 to disable it.

- performance_schema_accounts_size

Command-Line Format	--performance_schema_accounts_size=#	
Option-File Format	performance_schema_accounts_size	
System Variable Name	performance_schema_accounts_size	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	numeric
	Default	-1 (autosized)
	Range	-1 .. 1048576

The number of rows in the accounts table. If this variable is 0, the Performance Schema does not maintain connection statistics in the accounts table.

- performance_schema_digests_size

Command-Line Format	--performance_schema_digests_size=#	
Option-File Format	performance_schema_digests_size	
System Variable Name	performance_schema_digests_size	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	numeric
	Default	-1 (autosized)
	Range	-1 .. 1048576

The maximum number of rows in the events_statements_summary_by_digest table. If this maximum is exceeded such that a digest cannot be instrumented, the Performance Schema increments the Performance_schema_digest_lost status variable.

- performance_schema_events_stages_history_long_size

Command-Line Format	--performance_schema_events_stages_history_long_size=#
Option-File Format	performance_schema_events_stages_history_long_size
System Variable Name	performance_schema_events_stages_history_long_size
Variable Scope	Global
Dynamic Variable	No
	Permitted Values

	Type	numeric
	Default	-1 (autosized)

The number of rows in the events_stages_history_long table.

- performance_schema_events_stages_history_size

Command-Line Format	--performance_schema_events_stages_history_size=#	
Option-File Format	performance_schema_events_stages_history_size	
System Variable Name	performance_schema_events_stages_history_size	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	numeric
	Default	-1 (autosized)

The number of rows per thread in the events_stages_history table.

- performance_schema_events_statements_history_long_size

Command-Line Format	--performance_schema_events_statements_history_long_size=#	
Option-File Format	performance_schema_events_statements_history_long_size	
System Variable Name	performance_schema_events_statements_history_long_size	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	numeric
	Default	-1 (autosized)

The number of rows in the events_statements_history_long table.

- performance_schema_events_statements_history_size

Command-Line Format	--performance_schema_events_statements_history_size=#	
Option-File Format	performance_schema_events_statements_history_size	
System Variable Name	performance_schema_events_statements_history_size	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	numeric
	Default	-1 (autosized)

The number of rows per thread in the events_statements_history table.

- performance_schema_events_transactions_history_long_size

Introduced	5.7.3	
Command-Line Format	--performance_schema_events_transactions_history_long_size=#	
Option-File Format	performance_schema_events_transactions_history_long_size	
System Variable Name	performance_schema_events_transactions_history_long_size	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	numeric
	Default	10000

The number of rows in the events_transactions_history_long table. This variable was added in MySQL 5.7.3.

- performance_schema_events_transactions_history_size

Introduced	5.7.3	
Command-Line Format	--performance_schema_events_transactions_history_size=#	
Option-File Format	performance_schema_events_transactions_history_size	
System Variable Name	performance_schema_events_transactions_history_size	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	numeric
	Default	10

The number of rows per thread in the events_transactions_history table. This variable was added in MySQL 5.7.3.

- performance_schema_events_waits_history_long_size

Command-Line Format	--performance_schema_events_waits_history_long_size=#	
Option-File Format	performance_schema_events_waits_history_long_size	
System Variable Name	performance_schema_events_waits_history_long_size	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	numeric
	Default	-1 (autosized)

The number of rows in the events_waits_history_long table.

- performance_schema_events_waits_history_size

Command-Line Format	--performance_schema_events_waits_history_size=#

Option-File Format	performance_schema_events_waits_history_size	
System Variable Name	performance_schema_events_waits_history_size	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	numeric
	Default	-1 (autosized)

The number of rows per thread in the events_waits_history table.

- performance_schema_hosts_size

Command-Line Format	--performance_schema_hosts_size=#	
Option-File Format	performance_schema_hosts_size	
System Variable Name	performance_schema_hosts_size	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	numeric
	Default	-1 (autosized)
	Range	-1 .. 1048576

The number of rows in the hosts table. If this variable is 0, the Performance Schema does not maintain connection statistics in the hosts table.

- performance_schema_max_cond_classes

Command-Line Format	--performance_schema_max_cond_classes=#	
Option-File Format	performance_schema_max_cond_classes	
System Variable Name	performance_schema_max_cond_classes	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	numeric
	Default	80

The maximum number of condition instruments.

- performance_schema_max_cond_instances

Command-Line Format	--performance_schema_max_cond_instances=#
Option-File Format	performance_schema_max_cond_instances
System Variable Name	performance_schema_max_cond_instances
Variable Scope	Global
Dynamic Variable	No

	Permitted Values	
	Type	`numeric`
	Default	`-1 (autosized)`

The maximum number of instrumented condition objects.

- `performance_schema_max_file_classes`

Command-Line Format	`--performance_schema_max_file_classes=#`	
Option-File Format	`performance_schema_max_file_classes`	
System Variable Name	`performance_schema_max_file_classes`	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	`numeric`
	Default	`50`

The maximum number of file instruments.

- `performance_schema_max_file_handles`

Command-Line Format	`--performance_schema_max_file_handles=#`	
Option-File Format	`performance_schema_max_file_handles`	
System Variable Name	`performance_schema_max_file_handles`	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	`numeric`
	Default	`32768`

The maximum number of opened file objects.

The value of `performance_schema_max_file_handles` should be greater than the value of `open_files_limit`: `open_files_limit` affects the maximum number of open file handles the server can support and `performance_schema_max_file_handles` affects how many of these file handles can be instrumented.

- `performance_schema_max_file_instances`

Command-Line Format	`--performance_schema_max_file_instances=#`	
Option-File Format	`performance_schema_max_file_instances`	
System Variable Name	`performance_schema_max_file_instances`	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	`numeric`

	Default	`-1 (autosized)`

The maximum number of instrumented file objects.

- `performance_schema_max_memory_classes`

Introduced	5.7.2	
Command-Line Format	`--performance_schema_max_memory_classes=#`	
Option-File Format	`performance_schema_max_memory_classes`	
System Variable Name	`performance_schema_max_memory_classes`	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	`numeric`
	Default	`250`

The maximum number of memory instruments. This variable was added in MySQL 5.7.2.

- `performance_schema_max_metadata_locks`

Introduced	5.7.3	
Command-Line Format	`--performance_schema_max_metadata_locks=#`	
Option-File Format	`performance_schema_max_metadata_locks`	
System Variable Name	`performance_schema_max_metadata_locks`	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	`numeric`
	Default	`-1 (autosized)`

The maximum number of metadata lock instruments. This value controls the size of the `metadata_locks` table. If this maximum is exceeded such that a metadata lock cannot be instrumented, the Performance Schema increments the `Performance_schema_metadata_lock_lost` status variable.

This variable was added in MySQL 5.7.3.

- `performance_schema_max_mutex_classes`

Command-Line Format	`--performance_schema_max_mutex_classes=#`	
Option-File Format	`performance_schema_max_mutex_classes`	
System Variable Name	`performance_schema_max_mutex_classes`	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	`numeric`

	Default	200

The maximum number of mutex instruments.

- `performance_schema_max_mutex_instances`

Command-Line Format	`--performance_schema_max_mutex_instances=#`	
Option-File Format	`performance_schema_max_mutex_instances`	
System Variable Name	`performance_schema_max_mutex_instances`	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	`numeric`
	Default	`-1 (autosized)`

The maximum number of instrumented mutex objects.

- `performance_schema_max_prepared_statements_instances`

Introduced	5.7.4	
Command-Line Format	`--performance_schema_max_prepared_statements_instances=#`	
Option-File Format	`performance_schema_max_prepared_statements_instances=#`	
System Variable Name	`performance_schema_max_prepared_statements_instances`	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	`numeric`
	Default	`(autosized)`

The maximum number of rows in the `prepared_statements_instances` table. If this maximum is exceeded such that a prepared statement cannot be instrumented, the Performance Schema increments the `Performance_schema_prepared_statements_lost` status variable. The default value of this variable is autosized based on the value of the `max_prepared_stmt_count` system variable.

This variable was added in MySQL 5.7.4.

- `performance_schema_max_rwlock_classes`

Command-Line Format	`--performance_schema_max_rwlock_classes=#`	
Option-File Format	`performance_schema_max_rwlock_classes`	
System Variable Name	`performance_schema_max_rwlock_classes`	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values (<= 5.7.2)	
	Type	`numeric`

	Default	30
	Permitted Values (>= 5.7.3)	
	Type	numeric
	Default	40

The maximum number of rwlock instruments.

- performance_schema_max_program_instances

Introduced	5.7.2	
Command-Line Format	--performance_schema_max_program_instances=#	
Option-File Format	performance_schema_max_program_instances	
System Variable Name	performance_schema_max_program_instances	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	numeric
	Default	5000

The maximum number of stored programs for which the Performance Schema maintains statistics. If this maximum is exceeded, the Performance Schema increments the Performance_schema_program_lost status variable.

This variable was added in MySQL 5.7.2.

- performance_schema_max_rwlock_instances

Command-Line Format	--performance_schema_max_rwlock_instances=#	
Option-File Format	performance_schema_max_rwlock_instances	
System Variable Name	performance_schema_max_rwlock_instances	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	numeric
	Default	-1 (autosized)

The maximum number of instrumented rwlock objects.

- performance_schema_max_socket_classes

Command-Line Format	--performance_schema_max_socket_classes=#	
Option-File Format	performance_schema_max_socket_classes	
System Variable Name	performance_schema_max_socket_classes	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	

	Type	numeric
	Default	10

The maximum number of socket instruments.

- performance_schema_max_socket_instances

Command-Line Format	--performance_schema_max_socket_instances=#	
Option-File Format	performance_schema_max_socket_instances	
System Variable Name	performance_schema_max_socket_instances	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	numeric
	Default	-1 (autosized)

The maximum number of instrumented socket objects.

- performance_schema_max_stage_classes

Command-Line Format	--performance_schema_max_stage_classes=#	
Option-File Format	performance_schema_max_stage_classes	
System Variable Name	performance_schema_max_stage_classes	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	numeric
	Default	150

The maximum number of stage instruments.

- performance_schema_max_statement_classes

Command-Line Format	--performance_schema_max_statement_classes=#	
Option-File Format	performance_schema_max_statement_classes	
System Variable Name	performance_schema_max_statement_classes	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	numeric
	Default	autosized

The maximum number of statement instruments. The default value is calculated at server build time based on the number of commands in the client/server protocol and the number of SQL statement types supported by the server.

This variable should not be changed, unless to set it to 0 to disable all statement instrumentation and save all memory associated with it. Setting the variable to nonzero values other than the default has no benefit; in particular, values larger than the default cause more memory to be allocated then is needed.

- performance_schema_max_statement_stack

Introduced	5.7.2	
Command-Line Format	--performance_schema_max_statement_stack=#	
Option-File Format	performance_schema_max_statement_stack	
System Variable Name	performance_schema_max_statement_stack	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	numeric
	Default	10

The maximum depth of nested stored program calls for which the Performance Schema maintains statistics. When this maximum is exceeded, the Performance Schema increments the Performance_schema_nested_statement_lost status variable for each stored program statement executed.

This variable was added in MySQL 5.7.2.

- performance_schema_max_table_handles

Command-Line Format	--performance_schema_max_table_handles=#	
Option-File Format	performance_schema_max_table_handles	
System Variable Name	performance_schema_max_table_handles	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	numeric
	Default	-1 (autosized)

The maximum number of opened table objects. This value controls the size of the table_handles table. If this maximum is exceeded such that a table handle cannot be instrumented, the Performance Schema increments the Performance_schema_table_handles_lost status variable.

- performance_schema_max_table_instances

Command-Line Format	--performance_schema_max_table_instances=#
Option-File Format	performance_schema_max_table_instances
System Variable Name	performance_schema_max_table_instances
Variable Scope	Global
Dynamic Variable	No
	Permitted Values

	Type	numeric
	Default	-1 (autosized)

The maximum number of instrumented table objects.

- performance_schema_max_thread_classes

Command-Line Format	--performance_schema_max_thread_classes=#	
Option-File Format	performance_schema_max_thread_classes	
System Variable Name	performance_schema_max_thread_classes	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	numeric
	Default	50

The maximum number of thread instruments.

- performance_schema_max_thread_instances

Command-Line Format	--performance_schema_max_thread_instances=#	
Option-File Format	performance_schema_max_thread_instances	
System Variable Name	performance_schema_max_thread_instances	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	numeric
	Default	-1 (autosized)

The maximum number of instrumented thread objects. The value controls the size of the threads table. If this maximum is exceeded such that a thread cannot be instrumented, the Performance Schema increments the Performance_schema_thread_instances_lost status variable.

The max_connections system variable affects how many threads are run in the server. performance_schema_max_thread_instances affects how many of these running threads can be instrumented. The default value of performance_schema_max_thread_instances is autosized based on the value of max_connections.

- performance_schema_session_connect_attrs_size

Command-Line Format	--performance_schema_session_connect_attrs_size=#
Option-File Format	performance_schema_session_connect_attrs_size
System Variable Name	performance_schema_session_connect_attrs_size
Variable Scope	Global
Dynamic Variable	No
	Permitted Values

	Type	numeric
	Default	-1 (autosized)
	Range	-1 .. 1048576

The amount of preallocated memory per thread used to hold connection attribute strings. If the connection attribute strings are larger than the reserved storage, the `Performance_schema_session_connect_attrs_lost` status variable is incremented.

- `performance_schema_setup_actors_size`

Command-Line Format	`--performance_schema_setup_actors_size=#`	
Option-File Format	`performance_schema_setup_actors_size`	
System Variable Name	`performance_schema_setup_actors_size`	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	numeric
	Default	100

The number of rows in the `setup_actors` table.

- `performance_schema_setup_objects_size`

Command-Line Format	`--performance_schema_setup_objects_size=#`	
Option-File Format	`performance_schema_setup_objects_size`	
System Variable Name	`performance_schema_setup_objects_size`	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	numeric
	Default	100

The number of rows in the `setup_objects` table.

- `performance_schema_users_size`

Command-Line Format	`--performance_schema_users_size=#`	
Option-File Format	`performance_schema_users_size`	
System Variable Name	`performance_schema_users_size`	
Variable Scope	Global	
Dynamic Variable	No	
	Permitted Values	
	Type	numeric
	Default	-1 (autosized)
	Range	-1 .. 1048576

The number of rows in the users table. If this variable is 0, the Performance Schema does not maintain connection statistics in the users table.

20.13 Performance Schema Status Variables

The Performance Schema implements several status variables that provide information about instrumentation that could not be loaded or created due to memory constraints:

```
mysql> SHOW STATUS LIKE 'perf%';
+------------------------------------------+-------+
| Variable_name                            | Value |
+------------------------------------------+-------+
| Performance_schema_accounts_lost         | 0     |
| Performance_schema_cond_classes_lost     | 0     |
| Performance_schema_cond_instances_lost   | 0     |
| Performance_schema_file_classes_lost     | 0     |
| Performance_schema_file_handles_lost     | 0     |
| Performance_schema_file_instances_lost   | 0     |
| Performance_schema_hosts_lost            | 0     |
| Performance_schema_locker_lost           | 0     |
| Performance_schema_mutex_classes_lost    | 0     |
| Performance_schema_mutex_instances_lost  | 0     |
| Performance_schema_rwlock_classes_lost   | 0     |
| Performance_schema_rwlock_instances_lost | 0     |
| Performance_schema_socket_classes_lost   | 0     |
| Performance_schema_socket_instances_lost | 0     |
| Performance_schema_stage_classes_lost    | 0     |
| Performance_schema_statement_classes_lost | 0     |
| Performance_schema_table_handles_lost    | 0     |
| Performance_schema_table_instances_lost  | 0     |
| Performance_schema_thread_classes_lost   | 0     |
| Performance_schema_thread_instances_lost | 0     |
| Performance_schema_users_lost            | 0     |
+------------------------------------------+-------+
```

Performance Schema status variables have the following meanings:

- Performance_schema_accounts_lost

  The number of times a row could not be added to the accounts table because it was full.

- Performance_schema_cond_classes_lost

  How many condition instruments could not be loaded.

- Performance_schema_cond_instances_lost

  How many condition instrument instances could not be created.

- Performance_schema_digest_lost

  The number of digest instances that could not be instrumented in the events_statements_summary_by_digest table. This can be nonzero if the value of performance_schema_digests_size is too small.

- Performance_schema_file_classes_lost

  How many file instruments could not be loaded.

- Performance_schema_file_handles_lost

How many file instrument instances could not be opened.

- Performance_schema_file_instances_lost

  How many file instrument instances could not be created.

- Performance_schema_hosts_lost

  The number of times a row could not be added to the hosts table because it was full.

- Performance_schema_locker_lost

  How many events are "lost" or not recorded, due to the following conditions:

  - Events are recursive (for example, waiting for A caused a wait on B, which caused a wait on C).
  - The depth of the nested events stack is greater than the limit imposed by the implementation.

  Currently, events recorded by the Performance Schema are not recursive, so this variable should always be 0.

- Performance_schema_memory_classes_lost

  The number of times a memory instrument could not be loaded. This variable was added in MySQL 5.7.2.

- Performance_schema_metadata_lock_lost

  The number of metadata locks that could not be instrumented in the metadata_locks table. This can be nonzero if the value of performance_schema_max_metadata_locks is too small.

  This variable was added in MySQL 5.7.3.

- Performance_schema_mutex_classes_lost

  How many mutex instruments could not be loaded.

- Performance_schema_mutex_instances_lost

  How many mutex instrument instances could not be created.

- Performance_schema_nested_statement_lost

  The number of stored program statements for which statistics were lost. This can be nonzero if the value of performance_schema_max_statement_stack is too small.

  This variable was added in MySQL 5.7.2.

- Performance_schema_prepared_statements_lost

  The number of prepared statements that could not be instrumented in the prepared_statements_instances table. This can be nonzero if the value of performance_schema_max_prepared_statements_instances is too small.

  This variable was added in MySQL 5.7.4.

- Performance_schema_program_lost

The number of stored programs for which statistics were lost. This can be nonzero if the value of performance_schema_max_program_instances is too small.

This variable was added in MySQL 5.7.2.

- Performance_schema_rwlock_classes_lost

  How many rwlock instruments could not be loaded.

- Performance_schema_rwlock_instances_lost

  How many rwlock instrument instances could not be created.

- Performance_schema_session_connect_attrs_lost

  The number of times a connection attribute string was larger than the reserved storage.

- Performance_schema_socket_classes_lost

  How many socket instruments could not be loaded.

- Performance_schema_socket_instances_lost

  How many socket instrument instances could not be created.

- Performance_schema_stage_classes_lost

  How many stage instruments could not be loaded.

- Performance_schema_statement_classes_lost

  How many statement instruments could not be loaded.

- Performance_schema_table_handles_lost

  How many table instrument instances could not be opened. This can be nonzero if the value of performance_schema_max_table_handles is too small.

- Performance_schema_table_instances_lost

  How many table instrument instances could not be created.

- Performance_schema_thread_classes_lost

  How many thread instruments could not be loaded.

- Performance_schema_thread_instances_lost

  The number of thread instances that could not be instrumented in the threads table. This can be nonzero if the value of performance_schema_max_thread_instances is too small.

- Performance_schema_users_lost

  The number of times a row could not be added to the users table because it was full.

For information on using these variables to check Performance Schema status, see Section 20.5, “Performance Schema Status Monitoring”.

20.14 Performance Schema and Plugins

Removing a plugin with `UNINSTALL PLUGIN` does not affect information already collected for code in that plugin. Time spent executing the code while the plugin was loaded was still spent even if the plugin is unloaded later. The associated event information, including aggregate information, remains readable in `performance_schema` database tables. For additional information about the effect of plugin installation and removal, see Section 20.5, “Performance Schema Status Monitoring”.

A plugin implementor who instruments plugin code should document its instrumentation characteristics to enable those who load the plugin to account for its requirements. For example, a third-party storage engine should include in its documentation how much memory the engine needs for mutex and other instruments.

20.15 Using the Performance Schema to Diagnose Problems

The Performance Schema is a tool to help a DBA do performance tuning by taking real measurements instead of “wild guesses.” This section demonstrates some ways to use the Performance Schema for this purpose. The discussion here relies on the use of event filtering, which is described in Section 20.2.3.2, “Performance Schema Event Filtering”.

The following example provides one methodology that you can use to analyze a repeatable problem, such as investigating a performance bottleneck. To begin, you should have a repeatable use case where performance is deemed “too slow” and needs optimization, and you should enable all instrumentation (no pre-filtering at all).

1. Run the use case.

2. Using the Performance Schema tables, analyze the root cause of the performance problem. This analysis will rely heavily on post-filtering.

3. For problem areas that are ruled out, disable the corresponding instruments. For example, if analysis shows that the issue is not related to file I/O in a particular storage engine, disable the file I/O instruments for that engine. Then truncate the history and summary tables to remove previously collected events.

4. Repeat the process at step 1.

   At each iteration, the Performance Schema output, particularly the `events_waits_history_long` table, will contain less and less “noise” caused by nonsignificant instruments, and given that this table has a fixed size, will contain more and more data relevant to the analysis of the problem at hand.

   At each iteration, investigation should lead closer and closer to the root cause of the problem, as the “signal/noise” ratio will improve, making analysis easier.

5. Once a root cause of performance bottleneck is identified, take the appropriate corrective action, such as:

   - Tune the server parameters (cache sizes, memory, and so forth).

   - Tune a query by writing it differently,

   - Tune the database schema (tables, indexes, and so forth).

   - Tune the code (this applies to storage engine or server developers only).

6. Start again at step 1, to see the effects of the changes on performance.

The mutex_instances.LOCKED_BY_THREAD_ID and rwlock_instances.WRITE_LOCKED_BY_THREAD_ID columns are extremely important for investigating performance bottlenecks or deadlocks. This is made possible by Performance Schema instrumentation as follows:

1. Suppose that thread 1 is stuck waiting for a mutex.
2. You can determine what the thread is waiting for:

   ```
   SELECT * FROM events_waits_current WHERE THREAD_ID = thread_1;
   ```

   Say the query result identifies that the thread is waiting for mutex A, found in events_waits_current.OBJECT_INSTANCE_BEGIN.
3. You can determine which thread is holding mutex A:

   ```
   SELECT * FROM mutex_instances WHERE OBJECT_INSTANCE_BEGIN = mutex_A;
   ```

   Say the query result identifies that it is thread 2 holding mutex A, as found in mutex_instances.LOCKED_BY_THREAD_ID.
4. You can see what thread 2 is doing:

   ```
   SELECT * FROM events_waits_current WHERE THREAD_ID = thread_2;
   ```

Chapter 21 Connectors and APIs

Table of Contents

MySQL Connectors provide connectivity to the MySQL server for client programs. APIs provide low-level access to the MySQL protocol and MySQL resources. Both Connectors and the APIs enable you to connect and execute MySQL statements from another language or environment, including ODBC, Java (JDBC), Perl, Python, PHP, Ruby, and native C and embedded MySQL instances.

Note

Connector version numbers do not correlate with MySQL Server version numbers. See Table 21.2, “MySQL Connector Versions and MySQL Server Versions”.

MySQL Connectors

Oracle develops a number of connectors:

- Connector/ODBC provides driver support for connecting to MySQL using the Open Database Connectivity (ODBC) API. Support is available for ODBC connectivity from Windows, Unix, and Mac OS X platforms.
- Connector/Net enables developers to create .NET applications that connect to MySQL. Connector/Net implements a fully functional ADO.NET interface and provides support for use with ADO.NET aware tools. Applications that use Connector/Net can be written in any supported .NET language.

  The MySQL Visual Studio Plugin works with Connector/Net and Visual Studio 2005. The plugin is a MySQL DDEX Provider, which means that you can use the schema and data manipulation tools available in Visual Studio to create and edit objects within a MySQL database.
- Connector/J provides driver support for connecting to MySQL from Java applications using the standard Java Database Connectivity (JDBC) API.
- Connector/Python provides driver support for connecting to MySQL from Python applications using an API that is compliant with the Python DB API version 2.0. No additional Python modules or MySQL client libraries are required.
- Connector/C++ enables C++ applications to connect to MySQL.
- Connector/C is a standalone replacement for the MySQL Client Library (`libmysqlclient`), to be used for C applications.

The MySQL C API

For direct access to using MySQL natively within a C application, there are two methods:

- The C API provides low-level access to the MySQL client/server protocol through the `libmysqlclient` client library. This is the primary method used to connect to an instance of the MySQL server, and is used both by MySQL command-line clients and many of the MySQL Connectors and third-party APIs detailed here.

  `libmysqlclient` is included in MySQL distributions and in MySQL Connector/C distributions.
- `libmysqld` is an embedded MySQL server library that enables you to embed an instance of the MySQL server into your C applications.

  `libmysqld` is included in MySQL distributions, but not in MySQL Connector/C distributions.

See also Section 21.8.1, “MySQL C API Implementations”.

To access MySQL from a C application, or to build an interface to MySQL for a language not supported by the Connectors or APIs in this chapter, the C API is where to start. A number of programmer's utilities are available to help with the process; see Section 4.7, “MySQL Program Development Utilities”.

Third-Party MySQL APIs

The remaining APIs described in this chapter provide an interface to MySQL from specific application languages. These third-party solutions are not developed or supported by Oracle. Basic information on their usage and abilities is provided here for reference purposes only.

All the third-party language APIs are developed using one of two methods, using `libmysqlclient` or by implementing a *native driver*. The two solutions offer different benefits:

- Using *`libmysqlclient`* offers complete compatibility with MySQL because it uses the same libraries as the MySQL client applications. However, the feature set is limited to the implementation and interfaces exposed through `libmysqlclient` and the performance may be lower as data is copied between the native language, and the MySQL API components.
- *Native drivers* are an implementation of the MySQL network protocol entirely within the host language or environment. Native drivers are fast, as there is less copying of data between components, and they can offer advanced functionality not available through the standard MySQL API. Native drivers are also easier for end users to build and deploy because no copy of the MySQL client libraries is needed to build the native driver components.

Table 21.1, “MySQL APIs and Interfaces” lists many of the libraries and interfaces available for MySQL. Table 21.2, “MySQL Connector Versions and MySQL Server Versions” shows which MySQL Server versions each connector supports.

Table 21.1 MySQL APIs and Interfaces

Environment	API	Type	Notes
Ada	GNU Ada MySQL Bindings	`libmysqlclient`	See MySQL Bindings for GNU Ada
C	C API	`libmysqlclient`	See Section 21.8, “MySQL C API”.
C	Connector/C	Replacement for `libmysqlclient`	See MySQL Connector/C Developer Guide.
C++	Connector/C++	`libmysqlclient`	See MySQL Connector/C++ Developer Guide.
	MySQL++	`libmysqlclient`	See MySQL++ Web site.
	MySQL wrapped	`libmysqlclient`	See MySQL wrapped.
Cocoa	MySQL-Cocoa	`libmysqlclient`	Compatible with the Objective-C Cocoa environment. See http://mysql-cocoa.sourceforge.net/
D	MySQL for D	`libmysqlclient`	See MySQL for D.
Eiffel	Eiffel MySQL	`libmysqlclient`	See Section 21.14, “MySQL Eiffel Wrapper”.
Erlang	`erlang-mysql-driver`	`libmysqlclient`	See `erlang-mysql-driver`.
Haskell	Haskell MySQL Bindings	Native Driver	See Brian O'Sullivan's pure Haskell MySQL bindings.
	`hsql-mysql`	`libmysqlclient`	See MySQL driver for Haskell .
Java/ JDBC	Connector/J	Native Driver	See MySQL Connector/J Developer Guide.
Kaya	MyDB	`libmysqlclient`	See MyDB.

Environment	API	Type	Notes
Lua	LuaSQL	libmysqlclient	See LuaSQL.
.NET/ Mono	Connector/Net	Native Driver	See MySQL Connector/Net Developer Guide.
Objective Caml	OBjective Caml MySQL Bindings	libmysqlclient	See MySQL Bindings for Objective Caml.
Octave	Database bindings for GNU Octave	libmysqlclient	See Database bindings for GNU Octave.
ODBC	Connector/ODBC	libmysqlclient	See MySQL Connector/ODBC Developer Guide.
Perl	DBI/DBD::mysql	libmysqlclient	See Section 21.10, “MySQL Perl API”.
	Net::MySQL	Native Driver	See Net::MySQL at CPAN
PHP	mysql, ext/mysql interface (deprecated)	libmysqlclient	See Original MySQL API (Mysql).
	mysqli, ext/mysqli interface	libmysqlclient	See MySQL Improved Extension (Mysqli).
	PDO_MYSQL	libmysqlclient	See MySQL Functions (PDO_MYSQL) (MySQL (PDO)).
	PDO mysqlnd	Native Driver	
Python	Connector/Python	Native Driver	See MySQL Connector/Python Developer Guide.
	MySQLdb	libmysqlclient	See Section 21.11, “MySQL Python API”.
Ruby	MySQL/Ruby	libmysqlclient	Uses libmysqlclient. See Section 21.12.1, “The MySQL/Ruby API”.
	Ruby/MySQL	Native Driver	See Section 21.12.2, “The Ruby/MySQL API”.
Scheme	Myscsh	libmysqlclient	See Myscsh.
SPL	sql_mysql	libmysqlclient	See sql_mysql for SPL.
Tcl	MySQLtcl	libmysqlclient	See Section 21.13, “MySQL Tcl API”.

Table 21.2 MySQL Connector Versions and MySQL Server Versions

Connector	Connector version	MySQL Server version
Connector/C	6.1.0 GA	5.6, 5.5, 5.1, 5.0, 4.1
Connector/C++	1.0.5 GA	5.6, 5.5, 5.1
Connector/J	5.1.8	5.6, 5.5, 5.1, 5.0, 4.1
Connector/Net	6.5	5.6, 5.5, 5.1, 5.0
Connector/Net	6.4	5.6, 5.5, 5.1, 5.0
Connector/Net	6.3	5.6, 5.5, 5.1, 5.0
Connector/Net	6.2 (No longer supported)	5.6, 5.5, 5.1, 5.0
Connector/Net	6.1 (No longer supported)	5.6, 5.5, 5.1, 5.0
Connector/Net	6.0 (No longer supported)	5.6, 5.5, 5.1, 5.0
Connector/Net	5.2 (No longer supported)	5.6, 5.5, 5.1, 5.0
Connector/Net	1.0 (No longer supported)	5.0, 4.0

Connector	Connector version	MySQL Server version
Connector/ODBC	5.1	5.6, 5.5, 5.1, 5.0, 4.1.1+
Connector/ODBC	3.51 (Unicode not supported)	5.6, 5.5, 5.1, 5.0, 4.1

21.1 MySQL Connector/ODBC

The MySQL Connector/ODBC manual is now published in standalone form, not as part of the MySQL Reference Manual. For information, see these documents:

- Main manual: MySQL Connector/ODBC Developer Guide
- Release notes: MySQL Connector/ODBC Release Notes

21.2 MySQL Connector/Net

The MySQL Connector/Net manual is now published in standalone form, not as part of the MySQL Reference Manual. For information, see these documents:

- Main manual: MySQL Connector/Net Developer Guide
- Release notes: MySQL Connector/Net Release Notes

21.3 MySQL Connector/J

The MySQL Connector/J manual is now published in standalone form, not as part of the MySQL Reference Manual. For information, see these documents:

- Main manual: MySQL Connector/J Developer Guide
- Release notes: MySQL Connector/J Release Notes

21.4 MySQL Connector/C++

The MySQL Connector/C++ manual is now published in standalone form, not as part of the MySQL Reference Manual. For information, see these documents:

- Main manual: MySQL Connector/C++ Developer Guide
- Release notes: MySQL Connector/C++ Release Notes

21.5 MySQL Connector/C

The MySQL Connector/C manual is now published in standalone form, not as part of the MySQL Reference Manual. For information, see these documents:

- Main manual: MySQL Connector/C Developer Guide
- Release notes: MySQL Connector/C Release Notes

21.6 MySQL Connector/Python

The MySQL Connector/Python manual is now published in standalone form, not as part of the MySQL Reference Manual. For information, see these documents:

- Main manual: MySQL Connector/Python Developer Guide
- Release notes: MySQL Connector/Python Release Notes

21.7 libmysqld, the Embedded MySQL Server Library

The embedded MySQL server library makes it possible to run a full-featured MySQL server inside a client application. The main benefits are increased speed and more simple management for embedded applications.

The embedded server library is based on the client/server version of MySQL, which is written in C/C++. Consequently, the embedded server also is written in C/C++. There is no embedded server available in other languages.

The API is identical for the embedded MySQL version and the client/server version. To change a threaded application to use the embedded library, you normally only have to add calls to the following functions.

Table 21.3 MySQL Embedded Server Library Functions

Function	When to Call
`mysql_library_init()`	Call it before any other MySQL function is called, preferably early in the `main()` function.
`mysql_library_end()`	Call it before your program exits.
`mysql_thread_init()`	Call it in each thread you create that accesses MySQL.
`mysql_thread_end()`	Call it before calling `pthread_exit()`.

Then, link your code with `libmysqld.a` instead of `libmysqlclient.a`. To ensure binary compatibility between your application and the server library, always compile your application against headers for the same series of MySQL that was used to compile the server library. For example, if `libmysqld` was compiled against MySQL 5.1 headers, do not compile your application against MySQL 5.5 headers, or vice versa.

Because the `mysql_library_`*`xxx`*`()` functions are also included in `libmysqlclient.a`, you can change between the embedded and the client/server version by just linking your application with the right library. See Section 21.8.7.41, "`mysql_library_init()`".

One difference between the embedded server and the standalone server is that for the embedded server, authentication for connections is disabled by default.

21.7.1 Compiling Programs with `libmysqld`

In precompiled binary MySQL distributions that include `libmysqld`, the embedded server library, MySQL builds the library using the appropriate vendor compiler if there is one.

To get a `libmysqld` library if you build MySQL from source yourself, you should configure MySQL with the `-DWITH_EMBEDDED_SERVER=1` option. See Section 2.8.4, "MySQL Source-Configuration Options".

When you link your program with `libmysqld`, you must also include the system-specific `pthread` libraries and some libraries that the MySQL server uses. You can get the full list of libraries by executing `mysql_config --libmysqld-libs`.

The correct flags for compiling and linking a threaded program must be used, even if you do not directly call any thread functions in your code.

To compile a C program to include the necessary files to embed the MySQL server library into an executable version of a program, the compiler will need to know where to find various files and need instructions on how to compile the program. The following example shows how a program could be compiled from the command line, assuming that you are using `gcc`, use the GNU C compiler:

```
gcc mysql_test.c -o mysql_test \
`/usr/local/mysql/bin/mysql_config --include --libmysqld-libs`
```

Immediately following the `gcc` command is the name of the C program source file. After it, the `-o` option is given to indicate that the file name that follows is the name that the compiler is to give to the output file, the compiled program. The next line of code tells the compiler to obtain the location of the include files and libraries and other settings for the system on which it is compiled. The `mysql_config` command is contained in backticks, not single quotation marks.

On some non-`gcc` platforms, the embedded library depends on C++ runtime libraries and linking against the embedded library might result in missing-symbol errors. To solve this, link using a C++ compiler or explicitly list the required libraries on the link command line.

21.7.2 Restrictions When Using the Embedded MySQL Server

The embedded server has the following limitations:

- No user-defined functions (UDFs).
- No stack trace on core dump.
- You cannot set this up as a master or a slave (no replication).
- Very large result sets may be unusable on low memory systems.
- You cannot connect to an embedded server from an outside process with sockets or TCP/IP. However, you can connect to an intermediate application, which in turn can connect to an embedded server on the behalf of a remote client or outside process.
- `InnoDB` is not reentrant in the embedded server and cannot be used for multiple connections, either successively or simultaneously.
- The Event Scheduler is not available. Because of this, the `event_scheduler` system variable is disabled.

Some of these limitations can be changed by editing the `mysql_embed.h` include file and recompiling MySQL.

21.7.3 Options with the Embedded Server

Any options that may be given with the `mysqld` server daemon, may be used with an embedded server library. Server options may be given in an array as an argument to the `mysql_library_init()`, which initializes the server. They also may be given in an option file like `my.cnf`. To specify an option file for a C program, use the `--defaults-file` option as one of the elements of the second argument of the `mysql_library_init()` function. See Section 21.8.7.41, “`mysql_library_init()`”, for more information on the `mysql_library_init()` function.

Using option files can make it easier to switch between a client/server application and one where MySQL is embedded. Put common options under the `[server]` group. These are read by both MySQL versions.

Client/server-specific options should go under the [mysqld] section. Put options specific to the embedded MySQL server library in the [embedded] section. Options specific to applications go under section labeled [ApplicationName_SERVER]. See Section 4.2.3.3, "Using Option Files".

21.7.4 Embedded Server Examples

These two example programs should work without any changes on a Linux or FreeBSD system. For other operating systems, minor changes are needed, mostly with file paths. These examples are designed to give enough details for you to understand the problem, without the clutter that is a necessary part of a real application. The first example is very straightforward. The second example is a little more advanced with some error checking. The first is followed by a command-line entry for compiling the program. The second is followed by a GNUmake file that may be used for compiling instead.

Example 1

test1_libmysqld.c

```
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include "mysql.h"

MYSQL *mysql;
MYSQL_RES *results;
MYSQL_ROW record;

static char *server_options[] = \
       { "mysql_test", "--defaults-file=my.cnf", NULL };
int num_elements = (sizeof(server_options) / sizeof(char *)) - 1;

static char *server_groups[] = { "libmysqld_server",
                                 "libmysqld_client", NULL };

int main(void)
{
   mysql_library_init(num_elements, server_options, server_groups);
   mysql = mysql_init(NULL);
   mysql_options(mysql, MYSQL_READ_DEFAULT_GROUP, "libmysqld_client");
   mysql_options(mysql, MYSQL_OPT_USE_EMBEDDED_CONNECTION, NULL);

   mysql_real_connect(mysql, NULL,NULL,NULL, "database1", 0,NULL,0);

   mysql_query(mysql, "SELECT column1, column2 FROM table1");

   results = mysql_store_result(mysql);

   while((record = mysql_fetch_row(results))) {
      printf("%s - %s \n", record[0], record[1]);
   }

   mysql_free_result(results);
   mysql_close(mysql);
   mysql_library_end();

   return 0;
}
```

Here is the command line for compiling the above program:

```
gcc test1_libmysqld.c -o test1_libmysqld \
 `/usr/local/mysql/bin/mysql_config --include --libmysqld-libs`
```

Example 2

To try the example, create an `test2_libmysqld` directory at the same level as the MySQL source directory. Save the `test2_libmysqld.c` source and the `GNUmakefile` in the directory, and run GNU `make` from inside the `test2_libmysqld` directory.

`test2_libmysqld.c`

```
/*
 * A simple example client, using the embedded MySQL server library
*/

#include <mysql.h>
#include <stdarg.h>
#include <stdio.h>
#include <stdlib.h>

MYSQL *db_connect(const char *dbname);
void db_disconnect(MYSQL *db);
void db_do_query(MYSQL *db, const char *query);

const char *server_groups[] = {
  "test2_libmysqld_SERVER", "embedded", "server", NULL
};

int
main(int argc, char **argv)
{
  MYSQL *one, *two;

  /* mysql_library_init() must be called before any other mysql
   * functions.
   *
   * You can use mysql_library_init(0, NULL, NULL), and it
   * initializes the server using groups = {
   *   "server", "embedded", NULL
   *  }.
   *
   * In your $HOME/.my.cnf file, you probably want to put:

[test2_libmysqld_SERVER]
language = /path/to/source/of/mysql/sql/share/english

   * You could, of course, modify argc and argv before passing
   * them to this function.  Or you could create new ones in any
   * way you like.  But all of the arguments in argv (except for
   * argv[0], which is the program name) should be valid options
   * for the MySQL server.
   *
   * If you link this client against the normal mysqlclient
   * library, this function is just a stub that does nothing.
   */
  mysql_library_init(argc, argv, (char **)server_groups);

  one = db_connect("test");
  two = db_connect(NULL);

  db_do_query(one, "SHOW TABLE STATUS");
  db_do_query(two, "SHOW DATABASES");

  mysql_close(two);
  mysql_close(one);

  /* This must be called after all other mysql functions */
  mysql_library_end();
  exit(EXIT_SUCCESS);
}

static void
die(MYSQL *db, char *fmt, ...)
{
  va_list ap;
  va_start(ap, fmt);
  vfprintf(stderr, fmt, ap);
  va_end(ap);
  (void)putc('\n', stderr);
  if (db)
    db_disconnect(db);
  exit(EXIT_FAILURE);
}

MYSQL *
db_connect(const char *dbname)
{
  MYSQL *db = mysql_init(NULL);
  if (!db)
    die(db, "mysql_init failed: no memory");
  /*
   * Notice that the client and server use separate group names.
   * This is critical, because the server does not accept the
   * client's options, and vice versa.
   */
  mysql_options(db, MYSQL_READ_DEFAULT_GROUP, "test2_libmysqld_CLIENT");
  if (!mysql_real_connect(db, NULL, NULL, NULL, dbname, 0, NULL, 0))
    die(db, "mysql_real_connect failed: %s", mysql_error(db));

  return db;
}

void
db_disconnect(MYSQL *db)
{
  mysql_close(db);
}

void
db_do_query(MYSQL *db, const char *query)
{
  if (mysql_query(db, query) != 0)
    goto err;

  if (mysql_field_count(db) > 0)
  {
    MYSQL_RES   *res;
    MYSQL_ROW    row, end_row;
    int num_fields;

    if (!(res = mysql_store_result(db)))
      goto err;
    num_fields = mysql_num_fields(res);
    while ((row = mysql_fetch_row(res)))
    {
      (void)fputs(">> ", stdout);
      for (end_row = row + num_fields; row < end_row; ++row)
        (void)printf("%s\t", row ? (char*)*row : "NULL");
      (void)fputc('\n', stdout);
    }
    (void)fputc('\n', stdout);
    mysql_free_result(res);
  }
  else
```

```
    (void)printf("Affected rows: %lld\n", mysql_affected_rows(db));

  return;

err:
  die(db, "db_do_query failed: %s [%s]", mysql_error(db), query);
}
```

`GNUmakefile`

```
# This assumes the MySQL software is installed in /usr/local/mysql
inc      := /usr/local/mysql/include/mysql
lib      := /usr/local/mysql/lib

# If you have not installed the MySQL software yet, try this instead
#inc      := $(HOME)/mysql-5.7/include
#lib      := $(HOME)/mysql-5.7/libmysqld

CC       := gcc
CPPFLAGS := -I$(inc) -D_THREAD_SAFE -D_REENTRANT
CFLAGS   := -g -W -Wall
LDFLAGS  := -static
# You can change -lmysqld to -lmysqlclient to use the
# client/server library
LDLIBS    = -L$(lib) -lmysqld -lm -ldl -lcrypt

ifneq (,$(shell grep FreeBSD /COPYRIGHT 2>/dev/null))
# FreeBSD
LDFLAGS += -pthread
else
# Assume Linux
LDLIBS += -lpthread
endif

# This works for simple one-file test programs
sources := $(wildcard *.c)
objects := $(patsubst %c,%o,$(sources))
targets := $(basename $(sources))

all: $(targets)

clean:
        rm -f $(targets) $(objects) *.core
```

21.8 MySQL C API

The C API provides low-level access to the MySQL client/server protocol and enables C programs to access database contents. The C API code is distributed with MySQL and implemented in the `libmysqlclient` library. See Section 21.8.1, “MySQL C API Implementations”.

Most other client APIs use the `libmysqlclient` library to communicate with the MySQL server. (Exceptions are except Connector/J and Connector/Net.) This means that, for example, you can take advantage of many of the same environment variables that are used by other client programs because they are referenced from the library. For a list of these variables, see Section 4.1, “Overview of MySQL Programs”.

For instructions on building client programs using the C API, see Section 21.8.4.1, “Building C API Client Programs”. For programming with threads, see Section 21.8.4.2, “Writing C API Threaded Client Programs”. To create a standalone application which includes the "server" and "client" in the same program (and does not communicate with an external MySQL server), see Section 21.7, “libmysqld, the Embedded MySQL Server Library”.

> **Note**
>
> If, after an upgrade, you experience problems with compiled client programs, such as `Commands out of sync` or unexpected core dumps, the programs were probably compiled using old header or library files. In this case, check the date of the `mysql.h` file and `libmysqlclient.a` library used for compilation to verify that they are from the new MySQL distribution. If not, recompile the programs with the new headers and libraries. Recompilation might also be necessary for programs compiled against the shared client library if the library major version number has changed (for example, from `libmysqlclient.so.17` to `libmysqlclient.so.18`). For additional compatibility information, see Section 21.8.4.3, “Running C API Client Programs”.

Clients have a maximum communication buffer size. The size of the buffer that is allocated initially (16KB) is automatically increased up to the maximum size (16MB by default). Because buffer sizes are increased only as demand warrants, simply increasing the maximum limit does not in itself cause more resources to be used. This size check is mostly a precaution against erroneous statements and communication packets.

The communication buffer must be large enough to contain a single SQL statement (for client-to-server traffic) and one row of returned data (for server-to-client traffic). Each session's communication buffer is dynamically enlarged to handle any query or row up to the maximum limit. For example, if you have `BLOB` values that contain up to 16MB of data, you must have a communication buffer limit of at least 16MB (in both server and client). The default maximum built into the client library is 1GB, but the default maximum in the server is 1MB. You can increase this by changing the value of the `max_allowed_packet` parameter at server startup. See Section 8.11.2, “Tuning Server Parameters”.

The MySQL server shrinks each communication buffer to `net_buffer_length` bytes after each query. For clients, the size of the buffer associated with a connection is not decreased until the connection is closed, at which time client memory is reclaimed.

21.8.1 MySQL C API Implementations

The MySQL C API is a C-based API that client applications written in C can use to communicate with MySQL Server. Client programs refer to C API header files at compile time and link to a C API library file at link time. The library comes in two versions, depending on how the application is intended to communicate with the server:

- `libmysqlclient`: The client version of the library, used for applications that communicate over a network connection as a client of a standalone server process.

- `libmysqld`: The embedded server version of the library, used for applications intended to include an embedded MySQL server within the application itself. The application communicates with its own private server instance.

Both libraries have the same interface. In terms of C API calls, an application communicates with a standalone server the same way it communicates with an embedded server. A given client can be built to communicate with a standalone or embedded server, depending on whether it is linked against `libmysqlclient` or `libmysqld` at build time.

There are two ways to obtain the C API header and library files required to build C API client programs:

- Install a MySQL Server distribution. Server distributions include both `libmysqlclient` and `libmysqld`.

- Install a MySQL Connector/C distribution. Connector/C distributions include only `libmysqlclient`. They do not include `libmysqld`.

For both MySQL Server and MySQL Connector/C, you can install a binary distribution that contains the C API files pre-built, or you can use a source distribution and build the C API files yourself.

Normally, you install either a MySQL Server distribution or a MySQL Connector/C distribution, but not both. For information about issues involved with simultaneous MySQL Server and MySQL Connector/C installations, see Section 21.8.2, "Simultaneous MySQL Server and MySQL Connector/C Installations".

The names of the library files to use when linking C API client applications depend on the library type and platform for which a distribution is built:

- On Unix (and Unix-like) sytems, the static library is `libmysqlclient.a`. The dynamic library is `libmysqlclient.so` on most Unix systems and `libmysqlclient.dylib` on Mac OS X.

  For distributions that include embedded server libraries, the corresponding library names begin with `libmysqld` rather than `libmysqlclient`.

- On Windows, the static library is `mysqlclient.lib` and the dynamic library is `libmysql.dll`. Windows distributions also include `libmysql.lib`, a static import library needed for using the dynamic library.

  For distributions that include embedded server libraries, the corresponding library names are `mysqlserver.lib`, `libmysqld.dll`, and `libmysqld.lib`.

  Windows distributions also include a set of debug libraries. These have the same names as the nondebug libraries, but are located in the `lib/debug` library. You must use the debug libraries when compiling clients built using the debug C runtime.

On Unix, you may also see libraries that include `_r` in the names. Before MySQL 5.5, these were built as thread-safe (re-entrant) libraries separately from the non-`_r` libraries. As of 5.5, both libraries are the same and the `_r` names are symbolic links to the corresponding non-`_r` names. There is no need to use the `_r` libraries. For example, if you use `mysql_config` to obtain linker flags, you can use `mysql_config --libs` in all cases, even for threaded clients. There is no need to use `mysql_config --libs_r`.

21.8.2 Simultaneous MySQL Server and MySQL Connector/C Installations

MySQL Server and MySQL Connector/C installation packages both provide the files needed to build and run MySQL C API client programs. This section discusses when it is possible to install both products on the same system. For some packaging formats, this is possible without conflict. For others, both products cannot be installed at the same time.

This discussion assumes the use of similar package types for both products (for example, RPM packages for both products). It does not try to describe coexistence between packaging types (for example, use of RPM packages for one product and a `tar` file package for the other). Nor does it describe coexistence of packages provided by Oracle and those provided by third-party vendors.

If you install both products, it may be necessary to adjust your development tools or runtime environment to choose one set of header files and libraries over the other. See Section 21.8.4.1, "Building C API Client Programs", and Section 21.8.4.3, "Running C API Client Programs".

`tar` and Zip file packages install under the directory into which you unpack them. For example, you can unpack MySQL Server and MySQL Connector/C `tar` packages under `/usr/local` and they will unpack into distinct directory names without conflict.

Windows MSI installers use their own installation directory, so MySQL Server and MySQL Connector/C installers do not conflict.

Mac OS X DMG packages install under the same parent directory but in a different subdirectory, so there is no conflict. For example:

```
/usr/local/mysql-5.6.11-osx10.7-x86_64/
/usr/local/mysql-connector-c-6.1.0-osx10.7-x86/
```

Solaris PKG packages install under the same parent directory but in a different subdirectory, so there is no conflict. For example:

```
/opt/mysql/mysql
/opt/mysql/connector-c
```

The Solaris MySQL Connector/C installer does not create any symlinks from system directories such as `/usr/bin` or `/usr/lib` into the installation directory. That must be done manually if desired after installation.

For RPM installations, there are several types of RPM packages. MySQL Server `shared` and `devel` RPM packages are similar to the corresponding MySQL Connector/C RPM packages. These RPM package types cannot coexist because the MySQL Server and MySQL Connector/C RPM packages use the same installation locations for the client library-related files. This means the following conditions hold:

- If MySQL Server `shared` and `devel` RPM packages are installed, they provide the C API headers and libraries, and there is no need to install the MySQL Connector/C RPM packages. To install the MySQL Connector/C packages anyway, you must first remove the corresponding MySQL Server packages.
- To install MySQL Server RPM packages if you already have MySQL Connector/C RPM packages installed, you must first remove the MySQL Connector/C RPM packages.

MySQL Server RPM packages other than `shared` and `devel` do not conflict with MySQL Connector/C packages and can be installed if MySQL Connector/C is installed. This includes the main server RPM that includes the `mysqld` server itself.

21.8.3 Example C API Client Programs

Many of the clients in MySQL source distributions are written in C, such as `mysql`, `mysqladmin`, and `mysqlshow`. If you are looking for examples that demonstrate how to use the C API, take a look at these clients: Obtain a source distribution and look in its `client` directory. See Section 2.1.3, “How to Get MySQL”.

21.8.4 Building and Running C API Client Programs

The following sections provide information on building client programs that use the C API. Topics include compiling and linking clients, writing threaded clients, and troubleshooting runtime problems.

21.8.4.1 Building C API Client Programs

This section provides guidelines for compiling C programs that use the MySQL C API.

Compiling MySQL Clients on Unix

You may need to specify an `-I` option when you compile client programs that use MySQL header files, so that the compiler can find them. For example, if the header files are installed in `/usr/local/mysql/include`, use this option in the compile command:

```
-I/usr/local/mysql/include
```

MySQL clients must be linked using the `-lmysqlclient` option in the link command. You may also need to specify a `-L` option to tell the linker where to find the library. For example, if the library is installed in `/usr/local/mysql/lib`, use these options in the link command:

```
-L/usr/local/mysql/lib -lmysqlclient
```

The path names may differ on your system. Adjust the `-I` and `-L` options as necessary.

To make it simpler to compile MySQL programs on Unix, use the `mysql_config` script. See Section 4.7.1, “`mysql_config` — Display Options for Compiling Clients”.

`mysql_config` displays the options needed for compiling or linking:

```
shell> mysql_config --cflags
shell> mysql_config --libs
```

You can run those commands to get the proper options and add them manually to compilation or link commands. Alternatively, include the output from `mysql_config` directly within command lines using backticks:

```
shell> gcc -c `mysql_config --cflags` progname.c
shell> gcc -o progname progname.o `mysql_config --libs`
```

Compiling MySQL Clients on Microsoft Windows

To specify header and library file locations, use the facilities provided by your development environment.

To build C API clients on Windows, you must link in the C client library, as well as the Windows ws2_32 sockets library and Secur32 security library.

On Windows, you can link your code with either the dynamic or static C client library. The static library is named `mysqlclient.lib` and the dynamic library is named `libmysql.dll`. In addition, the `libmysql.lib` static import library is needed for using the dynamic library.

If you link with the static library, failure can occur unless these conditions are satisfied:

- The client application must be compiled with the same version of Visual Studio used to compile the library.

- The client application should link the C runtime statically by using the `/MT` compiler option.

If the client application is built in in debug mode and uses the static debug C runtime (`/MTd` compiler option), it can link to the `mysqlclient.lib` static library if that library was built using the same option. If the client application uses the dynamic C runtime (`/MD` option, or `/MDd` option in debug mode), it must must be linked to the `libmysql.dll` dynamic library. It cannot link to the static client library.

The MSDN page describing the link options can be found here: http://msdn.microsoft.com/en-us/library/2kzt1wy3.aspx

Troubleshooting Problems Linking to the MySQL Client Library

In MySQL 5.7, the MySQL client library includes SSL support built in. It is unnecessary to specify either `-lssl` or `-lcrypto` at link time. Doing so may in fact result in problems at runtime.

If the linker cannot find the MySQL client library, you might get undefined-reference errors for symbols that start with `mysql_`, such as those shown here:

```
/tmp/ccFKsdPa.o: In function `main':
/tmp/ccFKsdPa.o(.text+0xb): undefined reference to `mysql_init'
/tmp/ccFKsdPa.o(.text+0x31): undefined reference to `mysql_real_connect'
/tmp/ccFKsdPa.o(.text+0x69): undefined reference to `mysql_error'
```

```
/tmp/ccFKsdPa.o(.text+0x9a): undefined reference to `mysql_close'
```

You should be able to solve this problem by adding `-Ldir_path -lmysqlclient` at the end of your link command, where *`dir_path`* represents the path name of the directory where the client library is located. To determine the correct directory, try this command:

```
shell> mysql_config --libs
```

The output from `mysql_config` might indicate other libraries that should be specified on the link command as well. You can include `mysql_config` output directly in your compile or link command using backticks. For example:

```
shell> gcc -o progname progname.o `mysql_config --libs`
```

If an error occurs at link time that the `floor` symbol is undefined, link to the math library by adding `-lm` to the end of the compile/link line. Similarly, if you get undefined-reference errors for other functions that should exist on your system, such as `connect()`, check the manual page for the function in question to determine which libraries you should add to the link command.

If you get undefined-reference errors such as the following for functions that do not exist on your system, it usually means that your MySQL client library was compiled on a system that is not 100% compatible with yours:

```
mf_format.o(.text+0x201): undefined reference to `__lxstat'
```

In this case, you should download the latest MySQL or MySQL Connector/C source distribution and compile the MySQL client library yourself. See Section 2.8, “Installing MySQL from Source”, and MySQL Connector/C Developer Guide.

21.8.4.2 Writing C API Threaded Client Programs

The client library is almost thread-safe. The biggest problem is that the subroutines in `net.c` that read from sockets are not interrupt-safe. This was done with the thought that you might want to have your own alarm that can break a long read to a server. If you install interrupt handlers for the `SIGPIPE` interrupt, socket handling should be thread-safe.

To avoid aborting the program when a connection terminates, MySQL blocks `SIGPIPE` on the first call to `mysql_library_init()`, `mysql_init()`, or `mysql_connect()`. To use your own `SIGPIPE` handler, first call `mysql_library_init()`, then install your handler.

If “undefined symbol” errors occur when linking against the `libmysqlclient` client library, in most cases this is because you have not included the thread libraries on the link/compile command.

The client library is thread-safe per connection. You can let two threads share the same connection with the following caveats:

- Multiple threads cannot send a query to the MySQL server at the same time on the same connection. In particular, you must ensure that between calls to `mysql_query()` and `mysql_store_result()` in one thread, no other thread uses the same connection. You must have a mutex lock around your pair of `mysql_query()` and `mysql_store_result()` calls. After `mysql_store_result()` returns, the lock can be released and other threads may query the same connection.

  If you use POSIX threads, you can use `pthread_mutex_lock()` and `pthread_mutex_unlock()` to establish and release a mutex lock.

- Many threads can access different result sets that are retrieved with `mysql_store_result()`.

- To use mysql_use_result(), you must ensure that no other thread is using the same connection until the result set is closed. However, it really is best for threaded clients that share the same connection to use mysql_store_result().

You need to know the following if you have a thread that did not create the connection to the MySQL database but is calling MySQL functions:

When you call mysql_init(), MySQL creates a thread-specific variable for the thread that is used by the debug library (among other things). If you call a MySQL function before the thread has called mysql_init(), the thread does not have the necessary thread-specific variables in place and you are likely to end up with a core dump sooner or later. To avoid problems, you must do the following:

1. Call mysql_library_init() before any other MySQL functions. It is not thread-safe, so call it before threads are created, or protect the call with a mutex.

2. Arrange for mysql_thread_init() to be called early in the thread handler before calling any MySQL function. If you call mysql_init(), it will call mysql_thread_init() for you.

3. In the thread, call mysql_thread_end() before calling pthread_exit(). This frees the memory used by MySQL thread-specific variables.

The preceding notes regarding mysql_init() also apply to mysql_connect(), which calls mysql_init().

21.8.4.3 Running C API Client Programs

If, after an upgrade, you experience problems with compiled client programs, such as Commands out of sync or unexpected core dumps, the programs were probably compiled using old header or library files. In this case, check the date of the mysql.h file and libmysqlclient.a library used for compilation to verify that they are from the new MySQL distribution. If not, recompile the programs with the new headers and libraries. Recompilation might also be necessary for programs compiled against the shared client library if the library major version number has changed (for example, from libmysqlclient.so.17 to libmysqlclient.so.18).

The major client library version determines compatibility. (For example, for libmysqlclient.so.18.1.0, the major version is 18.) For this reason, the libraries shipped with newer versions of MySQL are drop-in replacements for older versions that have the same major number. As long as the major library version is the same, you can upgrade the library and old applications should continue to work with it.

Undefined-reference errors might occur at runtime when you try to execute a MySQL program. If these errors specify symbols that start with mysql_ or indicate that the libmysqlclient library cannot be found, it means that your system cannot find the shared libmysqlclient.so library. The solution to this problem is to tell your system to search for shared libraries in the directory where that library is located. Use whichever of the following methods is appropriate for your system:

- Add the path of the directory where libmysqlclient.so is located to the LD_LIBRARY_PATH or LD_LIBRARY environment variable.

- On Mac OS X, add the path of the directory where libmysqlclient.dylib is located to the DYLD_LIBRARY_PATH environment variable.

- Copy the shared-library files (such as libmysqlclient.so) to some directory that is searched by your system, such as /lib, and update the shared library information by executing ldconfig. Be sure to copy all related files. A shared library might exist under several names, using symlinks to provide the alternate names.

If the application is linked to the embedded server library, runtime error messages will indicate the `libmysqld` rather than `libmysqlclient` library, but the solution to the problem is the same as just described.

21.8.4.4 C API Server and Client Library Versions

The string and numeric forms of the MySQL server version are available at compile time as the values of the `MYSQL_SERVER_VERSION` and `MYSQL_VERSION_ID` macros, and at runtime as the values of the `mysql_get_server_info()` and `mysql_get_server_version()` functions.

As of MySQL 5.7.4 and Connector/C 6.1.3, the MySQL client library version depends on the type of distribution that provides the library:

- For MySQL distributions, the client library version is the MySQL version. The string and numeric forms of this version are available at compile time as the values of the `MYSQL_SERVER_VERSION` and `MYSQL_VERSION_ID` macros, and at runtime as the values of the `mysql_get_client_info()` and `mysql_get_client_version()` functions.

  The `LIBMYSQL_VERSION` and `LIBMYSQL_VERSION_ID` macros have the same values as `MYSQL_SERVER_VERSION` and `MYSQL_VERSION_ID` and the two sets of macros can be used interchangeably.

- For Connector/C distributions, the client library version is the Connector/C version. The string and numeric forms of this version are available at compile time as the values of the `LIBMYSQL_VERSION` and `LIBMYSQL_VERSION_ID` macros, and at runtime as the values of the `mysql_get_client_info()` and `mysql_get_client_version()` functions.

  The `MYSQL_SERVER_VERSION` and `MYSQL_VERSION_ID` macros indicate the string and numeric forms of the MySQL version on which the Connector/C distribution is based.

Prior to MySQL 5.7.4 and Connector/C 6.1.3, the client library version is the MySQL version. For Connector/C, this is the MySQL version on which the Connector/C distribution is based. The string and numeric forms of this version are available at compile time as the values of the `MYSQL_SERVER_VERSION` and `MYSQL_VERSION_ID` macros, and at runtime as the values of the `mysql_get_client_info()` and `mysql_get_client_version()` functions.

The `LIBMYSQL_VERSION` and `LIBMYSQL_VERSION_ID` macros are not defined before MySQL 5.7.4 and Connector/C 6.1.3.

21.8.5 C API Data Structures

This section describes C API data structures other than those used for prepared statements. For information about the latter, see Section 21.8.9, “C API Prepared Statement Data Structures”.

- `MYSQL`

  This structure represents a handle to one database connection. It is used for almost all MySQL functions. Do not try to make a copy of a `MYSQL` structure. There is no guarantee that such a copy will be usable.

- `MYSQL_RES`

  This structure represents the result of a query that returns rows (`SELECT`, `SHOW`, `DESCRIBE`, `EXPLAIN`). The information returned from a query is called the *result set* in the remainder of this section.

- `MYSQL_ROW`

This is a type-safe representation of one row of data. It is currently implemented as an array of counted byte strings. (You cannot treat these as null-terminated strings if field values may contain binary data, because such values may contain null bytes internally.) Rows are obtained by calling mysql_fetch_row().

- MYSQL_FIELD

  This structure contains metadata: information about a field, such as the field's name, type, and size. Its members are described in more detail later in this section. You may obtain the MYSQL_FIELD structures for each field by calling mysql_fetch_field() repeatedly. Field values are not part of this structure; they are contained in a MYSQL_ROW structure.

- MYSQL_FIELD_OFFSET

  This is a type-safe representation of an offset into a MySQL field list. (Used by mysql_field_seek().) Offsets are field numbers within a row, beginning at zero.

- my_ulonglong

  The type used for the number of rows and for mysql_affected_rows(), mysql_num_rows(), and mysql_insert_id(). This type provides a range of 0 to 1.84e19.

  On some systems, attempting to print a value of type my_ulonglong does not work. To print such a value, convert it to unsigned long and use a %lu print format. Example:

```
printf ("Number of rows: %lu\n",
        (unsigned long) mysql_num_rows(result));
```

- my_bool

  A boolean type, for values that are true (nonzero) or false (zero).

The MYSQL_FIELD structure contains the members described in the following list. The definitions apply primarily for columns of result sets such as those produced by SELECT statements. In MySQL 5.7, MYSQL_FIELD structures are also used to provide metadata for OUT and INOUT parameters returned from stored procedures executed using prepared CALL statements. For such parameters, some of the structure members have a meaning different from the meaning for column values.

- char * name

  The name of the field, as a null-terminated string. If the field was given an alias with an AS clause, the value of name is the alias. For a procedure parameter, the parameter name.

- char * org_name

  The name of the field, as a null-terminated string. Aliases are ignored. For expressions, the value is an empty string. For a procedure parameter, the parameter name.

- char * table

  The name of the table containing this field, if it is not a calculated field. For calculated fields, the table value is an empty string. If the column is selected from a view, table names the view. If the table or view was given an alias with an AS clause, the value of table is the alias. For a UNION, the value is the empty string. For a procedure parameter, the procedure name.

- char * org_table

The name of the table, as a null-terminated string. Aliases are ignored. If the column is selected from a view, org_table names the view. For a UNION, the value is the empty string. For a procedure parameter, the procedure name.

- char * db

  The name of the database that the field comes from, as a null-terminated string. If the field is a calculated field, db is an empty string. For a UNION, the value is the empty string. For a procedure parameter, the name of the database containing the procedure.

- char * catalog

  The catalog name. This value is always "def".

- char * def

  The default value of this field, as a null-terminated string. This is set only if you use mysql_list_fields().

- unsigned long length

  The width of the field. This corresponds to the display length, in bytes.

  The server determines the length value before it generates the result set, so this is the minimum length required for a data type capable of holding the largest possible value from the result column, without knowing in advance the actual values that will be produced by the query for the result set.

- unsigned long max_length

  The maximum width of the field for the result set (the length in bytes of the longest field value for the rows actually in the result set). If you use mysql_store_result() or mysql_list_fields(), this contains the maximum length for the field. If you use mysql_use_result(), the value of this variable is zero.

  The value of max_length is the length of the string representation of the values in the result set. For example, if you retrieve a FLOAT column and the “widest” value is -12.345, max_length is 7 (the length of '-12.345').

  If you are using prepared statements, max_length is not set by default because for the binary protocol the lengths of the values depend on the types of the values in the result set. (See Section 21.8.9, “C API Prepared Statement Data Structures”.) If you want the max_length values anyway, enable the STMT_ATTR_UPDATE_MAX_LENGTH option with mysql_stmt_attr_set() and the lengths will be set when you call mysql_stmt_store_result(). (See Section 21.8.11.3, “mysql_stmt_attr_set()”, and Section 21.8.11.28, “mysql_stmt_store_result()”.)

- unsigned int name_length

  The length of name.

- unsigned int org_name_length

  The length of org_name.

- unsigned int table_length

  The length of table.

- unsigned int org_table_length

The length of org_table.

- unsigned int db_length

  The length of db.

- unsigned int catalog_length

  The length of catalog.

- unsigned int def_length

  The length of def.

- unsigned int flags

  Bit-flags that describe the field. The flags value may have zero or more of the bits set that are shown in the following table.

Flag Value	Flag Description
NOT_NULL_FLAG	Field cannot be NULL
PRI_KEY_FLAG	Field is part of a primary key
UNIQUE_KEY_FLAG	Field is part of a unique key
MULTIPLE_KEY_FLAG	Field is part of a nonunique key
UNSIGNED_FLAG	Field has the UNSIGNED attribute
ZEROFILL_FLAG	Field has the ZEROFILL attribute
BINARY_FLAG	Field has the BINARY attribute
AUTO_INCREMENT_FLAG	Field has the AUTO_INCREMENT attribute
ENUM_FLAG	Field is an ENUM
SET_FLAG	Field is a SET
BLOB_FLAG	Field is a BLOB or TEXT (deprecated)
TIMESTAMP_FLAG	Field is a TIMESTAMP (deprecated)
NUM_FLAG	Field is numeric; see additional notes following table
NO_DEFAULT_VALUE_FLAG	Field has no default value; see additional notes following table

  Some of these flags indicate data type information and are superseded by or used in conjunction with the MYSQL_TYPE_*xxx* value in the field->type member described later:

  - To check for BLOB or TIMESTAMP values, check whether type is MYSQL_TYPE_BLOB or MYSQL_TYPE_TIMESTAMP. (The BLOB_FLAG and TIMESTAMP_FLAG flags are unneeded.)

  - ENUM and SET values are returned as strings. For these, check that the type value is MYSQL_TYPE_STRING and that the ENUM_FLAG or SET_FLAG flag is set in the flags value.

  NUM_FLAG indicates that a column is numeric. This includes columns with a type of MYSQL_TYPE_DECIMAL, MYSQL_TYPE_NEWDECIMAL, MYSQL_TYPE_TINY, MYSQL_TYPE_SHORT, MYSQL_TYPE_LONG, MYSQL_TYPE_FLOAT, MYSQL_TYPE_DOUBLE, MYSQL_TYPE_NULL, MYSQL_TYPE_LONGLONG, MYSQL_TYPE_INT24, and MYSQL_TYPE_YEAR.

NO_DEFAULT_VALUE_FLAG indicates that a column has no DEFAULT clause in its definition. This does not apply to NULL columns (because such columns have a default of NULL), or to AUTO_INCREMENT columns (which have an implied default value).

The following example illustrates a typical use of the flags value:

```
if (field->flags & NOT_NULL_FLAG)
    printf("Field cannot be null\n");
```

You may use the convenience macros shown in the following table to determine the boolean status of the flags value.

Flag Status	Description
IS_NOT_NULL(flags)	True if this field is defined as NOT NULL
IS_PRI_KEY(flags)	True if this field is a primary key
IS_BLOB(flags)	True if this field is a BLOB or TEXT (deprecated; test field->type instead)

- unsigned int decimals

  The number of decimals for numeric fields, and the fractional seconds precision for temporal fields.

- unsigned int charsetnr

  An ID number that indicates the character set/collation pair for the field.

  Normally, character values in result sets are converted to the character set indicated by the character_set_results system variable. In this case, charsetnr corresponds to the character set indicated by that variable. Character set conversion can be suppressed by setting character_set_results to NULL. In this case, charsetnr corresponds to the character set of the original table column or expression. See also Section 10.1.4, “Connection Character Sets and Collations”.

  To distinguish between binary and nonbinary data for string data types, check whether the charsetnr value is 63. If so, the character set is binary, which indicates binary rather than nonbinary data. This enables you to distinguish BINARY from CHAR, VARBINARY from VARCHAR, and the BLOB types from the TEXT types.

  charsetnr values are the same as those displayed in the Id column of the SHOW COLLATION statement or the ID column of the INFORMATION_SCHEMA COLLATIONS table. You can use those information sources to see which character set and collation specific charsetnr values indicate:

```
mysql> SHOW COLLATION WHERE Id = 63;
+-----------+---------+----+---------+----------+---------+
| Collation | Charset | Id | Default | Compiled | Sortlen |
+-----------+---------+----+---------+----------+---------+
| binary    | binary  | 63 | Yes     | Yes      |       1 |
+-----------+---------+----+---------+----------+---------+

mysql> SELECT COLLATION_NAME, CHARACTER_SET_NAME
    -> FROM INFORMATION_SCHEMA.COLLATIONS WHERE ID = 33;
+-----------------+--------------------+
| COLLATION_NAME  | CHARACTER_SET_NAME |
+-----------------+--------------------+
| utf8_general_ci | utf8               |
+-----------------+--------------------+
```

- `enum enum_field_types type`

  The type of the field. The `type` value may be one of the `MYSQL_TYPE_` symbols shown in the following table.

Type Value	Type Description
MYSQL_TYPE_TINY	TINYINT field
MYSQL_TYPE_SHORT	SMALLINT field
MYSQL_TYPE_LONG	INTEGER field
MYSQL_TYPE_INT24	MEDIUMINT field
MYSQL_TYPE_LONGLONG	BIGINT field
MYSQL_TYPE_DECIMAL	DECIMAL or NUMERIC field
MYSQL_TYPE_NEWDECIMAL	Precision math DECIMAL or NUMERIC
MYSQL_TYPE_FLOAT	FLOAT field
MYSQL_TYPE_DOUBLE	DOUBLE or REAL field
MYSQL_TYPE_BIT	BIT field
MYSQL_TYPE_TIMESTAMP	TIMESTAMP field
MYSQL_TYPE_DATE	DATE field
MYSQL_TYPE_TIME	TIME field
MYSQL_TYPE_DATETIME	DATETIME field
MYSQL_TYPE_YEAR	YEAR field
MYSQL_TYPE_STRING	CHAR or BINARY field
MYSQL_TYPE_VAR_STRING	VARCHAR or VARBINARY field
MYSQL_TYPE_BLOB	BLOB or TEXT field (use max_length to determine the maximum length)
MYSQL_TYPE_SET	SET field
MYSQL_TYPE_ENUM	ENUM field
MYSQL_TYPE_GEOMETRY	Spatial field
MYSQL_TYPE_NULL	NULL-type field

  The `MYSQL_TYPE_TIME2`, `MYSQL_TYPE_DATETIME2`, and `MYSQL_TYPE_TIMESTAMP2`) type codes are used only on the server side. Clients see the `MYSQL_TYPE_TIME`, `MYSQL_TYPE_DATETIME`, and `MYSQL_TYPE_TIMESTAMP` codes.

  You can use the `IS_NUM()` macro to test whether a field has a numeric type. Pass the `type` value to `IS_NUM()` and it evaluates to TRUE if the field is numeric:

```
if (IS_NUM(field->type))
    printf("Field is numeric\n");
```

  `ENUM` and `SET` values are returned as strings. For these, check that the `type` value is `MYSQL_TYPE_STRING` and that the `ENUM_FLAG` or `SET_FLAG` flag is set in the `flags` value.

21.8.6 C API Function Overview

The functions available in the C API are summarized here and described in greater detail in a later section. See Section 21.8.7, "C API Function Descriptions".

Table 21.4 C API Function Names and Descriptions

Function	Description
`my_init()`	Initialize global variables, and thread handler in thread-safe programs
`mysql_affected_rows()`	Returns the number of rows changed/deleted/inserted by the last `UPDATE`, `DELETE`, or `INSERT` query
`mysql_autocommit()`	Toggles autocommit mode on/off
`mysql_change_user()`	Changes user and database on an open connection
`mysql_character_set_name()`	Return default character set name for current connection
`mysql_client_find_plugin()`	Return pointer to plugin
`mysql_client_register_plugin()`	Register a plugin
`mysql_close()`	Closes a server connection
`mysql_commit()`	Commits the transaction
`mysql_connect()`	Connects to a MySQL server (this function is deprecated; use `mysql_real_connect()` instead)
`mysql_create_db()`	Creates a database (this function is deprecated; use the SQL statement `CREATE DATABASE` instead)
`mysql_data_seek()`	Seeks to an arbitrary row number in a query result set
`mysql_debug()`	Does a `DBUG_PUSH` with the given string
`mysql_drop_db()`	Drops a database (this function is deprecated; use the SQL statement `DROP DATABASE` instead)
`mysql_dump_debug_info()`	Makes the server write debug information to the log
`mysql_eof()`	Determines whether the last row of a result set has been read (this function is deprecated; `mysql_errno()` or `mysql_error()` may be used instead)
`mysql_errno()`	Returns the error number for the most recently invoked MySQL function
`mysql_error()`	Returns the error message for the most recently invoked MySQL function
`mysql_escape_string()`	Escapes special characters in a string for use in an SQL statement
`mysql_fetch_field()`	Returns the type of the next table field
`mysql_fetch_field_direct()`	Returns the type of a table field, given a field number
`mysql_fetch_fields()`	Returns an array of all field structures
`mysql_fetch_lengths()`	Returns the lengths of all columns in the current row
`mysql_fetch_row()`	Fetches the next row from the result set
`mysql_field_count()`	Returns the number of result columns for the most recent statement
`mysql_field_seek()`	Puts the column cursor on a specified column
`mysql_field_tell()`	Returns the position of the field cursor used for the last `mysql_fetch_field()`
`mysql_free_result()`	Frees memory used by a result set
`mysql_get_character_set_info()`	Return information about default character set

Function	Description
`mysql_get_client_info()`	Returns client version information as a string
`mysql_get_client_version()`	Returns client version information as an integer
`mysql_get_host_info()`	Returns a string describing the connection
`mysql_get_option()`	Returns the value of a `mysql_options()` option
`mysql_get_proto_info()`	Returns the protocol version used by the connection
`mysql_get_server_info()`	Returns the server version number
`mysql_get_server_version()`	Returns version number of server as an integer
`mysql_get_ssl_cipher()`	Return current SSL cipher
`mysql_hex_string()`	Encode string in hexadecimal format
`mysql_info()`	Returns information about the most recently executed query
`mysql_init()`	Gets or initializes a `MYSQL` structure
`mysql_insert_id()`	Returns the ID generated for an `AUTO_INCREMENT` column by the previous query
`mysql_kill()`	Kills a given thread
`mysql_library_end()`	Finalize the MySQL C API library
`mysql_library_init()`	Initialize the MySQL C API library
`mysql_list_dbs()`	Returns database names matching a simple regular expression
`mysql_list_fields()`	Returns field names matching a simple regular expression
`mysql_list_processes()`	Returns a list of the current server threads
`mysql_list_tables()`	Returns table names matching a simple regular expression
`mysql_load_plugin()`	Load a plugin
`mysql_load_plugin_v()`	Load a plugin
`mysql_more_results()`	Checks whether any more results exist
`mysql_next_result()`	Returns/initiates the next result in multiple-result executions
`mysql_num_fields()`	Returns the number of columns in a result set
`mysql_num_rows()`	Returns the number of rows in a result set
`mysql_options()`	Sets connect options for `mysql_real_connect()`
`mysql_options4()`	Sets connect options for `mysql_real_connect()`
`mysql_ping()`	Checks whether the connection to the server is working, reconnecting as necessary
`mysql_plugin_options()`	Set a plugin option
`mysql_query()`	Executes an SQL query specified as a null-terminated string
`mysql_real_connect()`	Connects to a MySQL server
`mysql_real_escape_string()`	Escapes special characters in a string for use in an SQL statement, taking into account the current character set of the connection
`mysql_real_query()`	Executes an SQL query specified as a counted string
`mysql_refresh()`	Flush or reset tables and caches
`mysql_reload()`	Tells the server to reload the grant tables
`mysql_reset_connection()`	Reset connection to clear session state

Function	Description
mysql_rollback()	Rolls back the transaction
mysql_row_seek()	Seeks to a row offset in a result set, using value returned from mysql_row_tell()
mysql_row_tell()	Returns the row cursor position
mysql_select_db()	Selects a database
mysql_server_end()	Finalize the MySQL C API library
mysql_server_init()	Initialize the MySQL C API library
mysql_session_track_get_first()	Get first part of session state-change information
mysql_session_track_get_next()	Get next part of session state-change information
mysql_set_character_set()	Set default character set for current connection
mysql_set_local_infile_default()	Set the LOAD DATA LOCAL INFILE handler callbacks to their default values
mysql_set_local_infile_handler()	Install application-specific LOAD DATA LOCAL INFILE handler callbacks
mysql_set_server_option()	Sets an option for the connection (like multi-statements)
mysql_sqlstate()	Returns the SQLSTATE error code for the last error
mysql_shutdown()	Shuts down the database server
mysql_ssl_set()	Prepare to establish SSL connection to server
mysql_stat()	Returns the server status as a string
mysql_store_result()	Retrieves a complete result set to the client
mysql_thread_end()	Finalize thread handler
mysql_thread_id()	Returns the current thread ID
mysql_thread_init()	Initialize thread handler
mysql_thread_safe()	Returns 1 if the clients are compiled as thread-safe
mysql_use_result()	Initiates a row-by-row result set retrieval
mysql_warning_count()	Returns the warning count for the previous SQL statement

Application programs should use this general outline for interacting with MySQL:

1. Initialize the MySQL library by calling mysql_library_init(). This function exists in both the libmysqlclient C client library and the libmysqld embedded server library, so it is used whether you build a regular client program by linking with the -libmysqlclient flag, or an embedded server application by linking with the -libmysqld flag.

2. Initialize a connection handler by calling mysql_init() and connect to the server by calling mysql_real_connect().

3. Issue SQL statements and process their results. (The following discussion provides more information about how to do this.)

4. Close the connection to the MySQL server by calling mysql_close().

5. End use of the MySQL library by calling mysql_library_end().

The purpose of calling mysql_library_init() and mysql_library_end() is to provide proper initialization and finalization of the MySQL library. For applications that are linked with the client library, they provide improved memory management. If you do not call `mysql_library_end()`, a block of memory remains allocated. (This does not increase the amount of memory used by the application, but some memory leak detectors will complain about it.) For applications that are linked with the embedded server, these calls start and stop the server.

In a nonmulti-threaded environment, the call to `mysql_library_init()` may be omitted, because `mysql_init()` will invoke it automatically as necessary. However, `mysql_library_init()` is not thread-safe in a multi-threaded environment, and thus neither is `mysql_init()`, which calls `mysql_library_init()`. You must either call `mysql_library_init()` prior to spawning any threads, or else use a mutex to protect the call, whether you invoke `mysql_library_init()` or indirectly through `mysql_init()`. This should be done prior to any other client library call.

To connect to the server, call `mysql_init()` to initialize a connection handler, then call `mysql_real_connect()` with that handler (along with other information such as the host name, user name, and password). Upon connection, `mysql_real_connect()` sets the `reconnect` flag (part of the `MYSQL` structure) to a value of `1` in versions of the API older than 5.0.3, or `0` in newer versions. A value of `1` for this flag indicates that if a statement cannot be performed because of a lost connection, to try reconnecting to the server before giving up. You can use the `MYSQL_OPT_RECONNECT` option to `mysql_options()` to control reconnection behavior. When you are done with the connection, call `mysql_close()` to terminate it.

While a connection is active, the client may send SQL statements to the server using `mysql_query()` or `mysql_real_query()`. The difference between the two is that `mysql_query()` expects the query to be specified as a null-terminated string whereas `mysql_real_query()` expects a counted string. If the string contains binary data (which may include null bytes), you must use `mysql_real_query()`.

For each non-`SELECT` query (for example, `INSERT`, `UPDATE`, `DELETE`), you can find out how many rows were changed (affected) by calling `mysql_affected_rows()`.

For `SELECT` queries, you retrieve the selected rows as a result set. (Note that some statements are `SELECT`-like in that they return rows. These include `SHOW`, `DESCRIBE`, and `EXPLAIN`. Treat these statements the same way as `SELECT` statements.)

There are two ways for a client to process result sets. One way is to retrieve the entire result set all at once by calling `mysql_store_result()`. This function acquires from the server all the rows returned by the query and stores them in the client. The second way is for the client to initiate a row-by-row result set retrieval by calling `mysql_use_result()`. This function initializes the retrieval, but does not actually get any rows from the server.

In both cases, you access rows by calling `mysql_fetch_row()`. With `mysql_store_result()`, `mysql_fetch_row()` accesses rows that have previously been fetched from the server. With `mysql_use_result()`, `mysql_fetch_row()` actually retrieves the row from the server. Information about the size of the data in each row is available by calling `mysql_fetch_lengths()`.

After you are done with a result set, call `mysql_free_result()` to free the memory used for it.

The two retrieval mechanisms are complementary. Choose the approach that is most appropriate for each client application. In practice, clients tend to use `mysql_store_result()` more commonly.

An advantage of `mysql_store_result()` is that because the rows have all been fetched to the client, you not only can access rows sequentially, you can move back and forth in the result set using `mysql_data_seek()` or `mysql_row_seek()` to change the current row position within the result set. You can also find out how many rows there are by calling `mysql_num_rows()`. On the other hand, the memory requirements for `mysql_store_result()` may be very high for large result sets and you are more likely to encounter out-of-memory conditions.

An advantage of mysql_use_result() is that the client requires less memory for the result set because it maintains only one row at a time (and because there is less allocation overhead, mysql_use_result() can be faster). Disadvantages are that you must process each row quickly to avoid tying up the server, you do not have random access to rows within the result set (you can only access rows sequentially), and the number of rows in the result set is unknown until you have retrieved them all. Furthermore, you *must* retrieve all the rows even if you determine in mid-retrieval that you've found the information you were looking for.

The API makes it possible for clients to respond appropriately to statements (retrieving rows only as necessary) without knowing whether the statement is a SELECT. You can do this by calling mysql_store_result() after each mysql_query() (or mysql_real_query()). If the result set call succeeds, the statement was a SELECT and you can read the rows. If the result set call fails, call mysql_field_count() to determine whether a result was actually to be expected. If mysql_field_count() returns zero, the statement returned no data (indicating that it was an INSERT, UPDATE, DELETE, and so forth), and was not expected to return rows. If mysql_field_count() is nonzero, the statement should have returned rows, but did not. This indicates that the statement was a SELECT that failed. See the description for mysql_field_count() for an example of how this can be done.

Both mysql_store_result() and mysql_use_result() enable you to obtain information about the fields that make up the result set (the number of fields, their names and types, and so forth). You can access field information sequentially within the row by calling mysql_fetch_field() repeatedly, or by field number within the row by calling mysql_fetch_field_direct(). The current field cursor position may be changed by calling mysql_field_seek(). Setting the field cursor affects subsequent calls to mysql_fetch_field(). You can also get information for fields all at once by calling mysql_fetch_fields().

For detecting and reporting errors, MySQL provides access to error information by means of the mysql_errno() and mysql_error() functions. These return the error code or error message for the most recently invoked function that can succeed or fail, enabling you to determine when an error occurred and what it was.

21.8.7 C API Function Descriptions

In the descriptions here, a parameter or return value of NULL means NULL in the sense of the C programming language, not a MySQL NULL value.

Functions that return a value generally return a pointer or an integer. Unless specified otherwise, functions returning a pointer return a non-NULL value to indicate success or a NULL value to indicate an error, and functions returning an integer return zero to indicate success or nonzero to indicate an error. Note that “nonzero” means just that. Unless the function description says otherwise, do not test against a value other than zero:

```
if (result)                   /* correct */
    ... error ...

if (result < 0)               /* incorrect */
    ... error ...

if (result == -1)             /* incorrect */
    ... error ...
```

When a function returns an error, the **Errors** subsection of the function description lists the possible types of errors. You can find out which of these occurred by calling mysql_errno(). A string representation of the error may be obtained by calling mysql_error().

21.8.7.1 mysql_affected_rows()

```
my_ulonglong mysql_affected_rows(MYSQL *mysql)
```

Description

mysql_affected_rows() may be called immediately after executing a statement with mysql_query() or mysql_real_query(). It returns the number of rows changed, deleted, or inserted by the last statement if it was an UPDATE, DELETE, or INSERT. For SELECT statements, mysql_affected_rows() works like mysql_num_rows().

For UPDATE statements, the affected-rows value by default is the number of rows actually changed. If you specify the CLIENT_FOUND_ROWS flag to mysql_real_connect() when connecting to mysqld, the affected-rows value is the number of rows “found”; that is, matched by the WHERE clause.

For REPLACE statements, the affected-rows value is 2 if the new row replaced an old row, because in this case, one row was inserted after the duplicate was deleted.

For INSERT ... ON DUPLICATE KEY UPDATE statements, the affected-rows value per row is 1 if the row is inserted as a new row, 2 if an existing row is updated, and 0 if an existing row is set to its current values. If you specify the CLIENT_FOUND_ROWS flag, the affected-rows value is 1 (not 0) if an existing row is set to its current values.

Following a CALL statement for a stored procedure, mysql_affected_rows() returns the value that it would return for the last statement executed within the procedure, or 0 if that statement would return -1. Within the procedure, you can use ROW_COUNT() at the SQL level to obtain the affected-rows value for individual statements.

In MySQL 5.7, mysql_affected_rows() returns a meaningful value for a wider range of statements. For details, see the description for ROW_COUNT() in Section 12.14, “Information Functions”.

Return Values

An integer greater than zero indicates the number of rows affected or retrieved. Zero indicates that no records were updated for an UPDATE statement, no rows matched the WHERE clause in the query or that no query has yet been executed. -1 indicates that the query returned an error or that, for a SELECT query, mysql_affected_rows() was called prior to calling mysql_store_result().

Because mysql_affected_rows() returns an unsigned value, you can check for -1 by comparing the return value to (my_ulonglong)-1 (or to (my_ulonglong)~0, which is equivalent).

Errors

None.

Example

```
char *stmt = "UPDATE products SET cost=cost*1.25
              WHERE group=10";
mysql_query(&mysql,stmt);
printf("%ld products updated",
       (long) mysql_affected_rows(&mysql));
```

21.8.7.2 mysql_autocommit()

```
my_bool mysql_autocommit(MYSQL *mysql, my_bool mode)
```

Description

Sets autocommit mode on if `mode` is 1, off if `mode` is 0.

Return Values

Zero for success. Nonzero if an error occurred.

Errors

None.

21.8.7.3 `mysql_change_user()`

```
my_bool mysql_change_user(MYSQL *mysql, const char *user, const char *password,
const char *db)
```

Description

Changes the user and causes the database specified by `db` to become the default (current) database on the connection specified by `mysql`. In subsequent queries, this database is the default for table references that include no explicit database specifier.

`mysql_change_user()` fails if the connected user cannot be authenticated or does not have permission to use the database. In this case, the user and database are not changed.

Pass a `db` parameter of `NULL` if you do not want to have a default database.

This function resets the session state as if one had done a new connect and reauthenticated. (See Section 21.8.16, "Controlling Automatic Reconnection Behavior".) It always performs a `ROLLBACK` of any active transactions, closes and drops all temporary tables, and unlocks all locked tables. Session system variables are reset to the values of the corresponding global system variables. Prepared statements are released and `HANDLER` variables are closed. Locks acquired with `GET_LOCK()` are released. These effects occur even if the user did not change.

To reset the connection state in a more lightweight manner without changing the user, use `mysql_reset_connection()`.

Return Values

Zero for success. Nonzero if an error occurred.

Errors

The same that you can get from `mysql_real_connect()`, plus:

- `CR_COMMANDS_OUT_OF_SYNC`

  Commands were executed in an improper order.

- `CR_SERVER_GONE_ERROR`

  The MySQL server has gone away.

- `CR_SERVER_LOST`

  The connection to the server was lost during the query.

- `CR_UNKNOWN_ERROR`

An unknown error occurred.

- ER_UNKNOWN_COM_ERROR

  The MySQL server does not implement this command (probably an old server).

- ER_ACCESS_DENIED_ERROR

  The user or password was wrong.

- ER_BAD_DB_ERROR

  The database did not exist.

- ER_DBACCESS_DENIED_ERROR

  The user did not have access rights to the database.

- ER_WRONG_DB_NAME

  The database name was too long.

Example

```
if (mysql_change_user(&mysql, "user", "password", "new_database"))
{
   fprintf(stderr, "Failed to change user.  Error: %s\n",
           mysql_error(&mysql));
}
```

21.8.7.4 mysql_character_set_name()

`const char *mysql_character_set_name(MYSQL *mysql)`

Description

Returns the default character set name for the current connection.

Return Values

The default character set name

Errors

None.

21.8.7.5 mysql_close()

`void mysql_close(MYSQL *mysql)`

Description

Closes a previously opened connection. `mysql_close()` also deallocates the connection handle pointed to by `mysql` if the handle was allocated automatically by `mysql_init()` or `mysql_connect()`.

Return Values

None.

Errors

None.

21.8.7.6 `mysql_commit()`

```
my_bool mysql_commit(MYSQL *mysql)
```

Description

Commits the current transaction.

The action of this function is subject to the value of the `completion_type` system variable. In particular, if the value of `completion_type` is `RELEASE` (or 2), the server performs a release after terminating a transaction and closes the client connection. Call `mysql_close()` from the client program to close the connection from the client side.

Return Values

Zero for success. Nonzero if an error occurred.

Errors

None.

21.8.7.7 `mysql_connect()`

```
MYSQL *mysql_connect(MYSQL *mysql, const char *host, const char *user, const
char *passwd)
```

Description

This function is deprecated. Use `mysql_real_connect()` instead.

`mysql_connect()` attempts to establish a connection to a MySQL database engine running on `host`. `mysql_connect()` must complete successfully before you can execute any of the other API functions, with the exception of `mysql_get_client_info()`.

The meanings of the parameters are the same as for the corresponding parameters for `mysql_real_connect()` with the difference that the connection parameter may be `NULL`. In this case, the C API allocates memory for the connection structure automatically and frees it when you call `mysql_close()`. The disadvantage of this approach is that you cannot retrieve an error message if the connection fails. (To get error information from `mysql_errno()` or `mysql_error()`, you must provide a valid `MYSQL` pointer.)

Return Values

Same as for `mysql_real_connect()`.

Errors

Same as for `mysql_real_connect()`.

21.8.7.8 `mysql_create_db()`

```
int mysql_create_db(MYSQL *mysql, const char *db)
```

Description

Creates the database named by the db parameter.

This function is deprecated. It is preferable to use mysql_query() to issue an SQL CREATE DATABASE statement instead.

Return Values

Zero for success. Nonzero if an error occurred.

Errors

- CR_COMMANDS_OUT_OF_SYNC

  Commands were executed in an improper order.

- CR_SERVER_GONE_ERROR

  The MySQL server has gone away.

- CR_SERVER_LOST

  The connection to the server was lost during the query.

- CR_UNKNOWN_ERROR

  An unknown error occurred.

Example

```
if(mysql_create_db(&mysql, "my_database"))
{
   fprintf(stderr, "Failed to create new database.  Error: %s\n",
           mysql_error(&mysql));
}
```

21.8.7.9 mysql_data_seek()

void mysql_data_seek(MYSQL_RES *result, my_ulonglong offset)

Description

Seeks to an arbitrary row in a query result set. The offset value is a row number. Specify a value in the range from 0 to mysql_num_rows(result)-1.

This function requires that the result set structure contains the entire result of the query, so mysql_data_seek() may be used only in conjunction with mysql_store_result(), not with mysql_use_result().

Return Values

None.

Errors

None.

21.8.7.10 mysql_debug()

```
void mysql_debug(const char *debug)
```

Description

Does a DBUG_PUSH with the given string. mysql_debug() uses the Fred Fish debug library. To use this function, you must compile the client library to support debugging. See Section 22.4.3, "The DBUG Package".

Return Values

None.

Errors

None.

Example

The call shown here causes the client library to generate a trace file in /tmp/client.trace on the client machine:

```
mysql_debug("d:t:O,/tmp/client.trace");
```

21.8.7.11 mysql_drop_db()

```
int mysql_drop_db(MYSQL *mysql, const char *db)
```

Description

Drops the database named by the db parameter.

This function is deprecated. It is preferable to use mysql_query() to issue an SQL DROP DATABASE statement instead.

Return Values

Zero for success. Nonzero if an error occurred.

Errors

- CR_COMMANDS_OUT_OF_SYNC

  Commands were executed in an improper order.

- CR_SERVER_GONE_ERROR

  The MySQL server has gone away.

- CR_SERVER_LOST

  The connection to the server was lost during the query.

- CR_UNKNOWN_ERROR

An unknown error occurred.

Example

```
if(mysql_drop_db(&mysql, "my_database"))
  fprintf(stderr, "Failed to drop the database: Error: %s\n",
          mysql_error(&mysql));
```

21.8.7.12 mysql_dump_debug_info()

`int mysql_dump_debug_info(MYSQL *mysql)`

Description

Instructs the server to write debugging information to the error log. The connected user must have the `SUPER` privilege.

Return Values

Zero for success. Nonzero if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC`

  Commands were executed in an improper order.

- `CR_SERVER_GONE_ERROR`

  The MySQL server has gone away.

- `CR_SERVER_LOST`

  The connection to the server was lost during the query.

- `CR_UNKNOWN_ERROR`

  An unknown error occurred.

21.8.7.13 mysql_eof()

`my_bool mysql_eof(MYSQL_RES *result)`

Description

This function is deprecated. `mysql_errno()` or `mysql_error()` may be used instead.

`mysql_eof()` determines whether the last row of a result set has been read.

If you acquire a result set from a successful call to `mysql_store_result()`, the client receives the entire set in one operation. In this case, a `NULL` return from `mysql_fetch_row()` always means the end of the result set has been reached and it is unnecessary to call `mysql_eof()`. When used with `mysql_store_result()`, `mysql_eof()` always returns true.

On the other hand, if you use `mysql_use_result()` to initiate a result set retrieval, the rows of the set are obtained from the server one by one as you call `mysql_fetch_row()` repeatedly. Because an error may occur on the connection during this process, a `NULL` return value from `mysql_fetch_row()`

does not necessarily mean the end of the result set was reached normally. In this case, you can use `mysql_eof()` to determine what happened. `mysql_eof()` returns a nonzero value if the end of the result set was reached and zero if an error occurred.

Historically, `mysql_eof()` predates the standard MySQL error functions `mysql_errno()` and `mysql_error()`. Because those error functions provide the same information, their use is preferred over `mysql_eof()`, which is deprecated. (In fact, they provide more information, because `mysql_eof()` returns only a boolean value whereas the error functions indicate a reason for the error when one occurs.)

Return Values

Zero for success. Nonzero if the end of the result set has been reached.

Errors

None.

Example

The following example shows how you might use `mysql_eof()`:

```
mysql_query(&mysql,"SELECT * FROM some_table");
result = mysql_use_result(&mysql);
while((row = mysql_fetch_row(result)))
{
    // do something with data
}
if(!mysql_eof(result))  // mysql_fetch_row() failed due to an error
{
    fprintf(stderr, "Error: %s\n", mysql_error(&mysql));
}
```

However, you can achieve the same effect with the standard MySQL error functions:

```
mysql_query(&mysql,"SELECT * FROM some_table");
result = mysql_use_result(&mysql);
while((row = mysql_fetch_row(result)))
{
    // do something with data
}
if(mysql_errno(&mysql))  // mysql_fetch_row() failed due to an error
{
    fprintf(stderr, "Error: %s\n", mysql_error(&mysql));
}
```

21.8.7.14 `mysql_errno()`

`unsigned int mysql_errno(MYSQL *mysql)`

Description

For the connection specified by `mysql`, `mysql_errno()` returns the error code for the most recently invoked API function that can succeed or fail. A return value of zero means that no error occurred. Client error message numbers are listed in the MySQL `errmsg.h` header file. Server error message numbers are listed in `mysqld_error.h`. Errors also are listed at Appendix C, *Errors, Error Codes, and Common Problems*.

Note that some functions like `mysql_fetch_row()` do not set `mysql_errno()` if they succeed.

A rule of thumb is that all functions that have to ask the server for information reset mysql_errno() if they succeed.

MySQL-specific error numbers returned by mysql_errno() differ from SQLSTATE values returned by mysql_sqlstate(). For example, the mysql client program displays errors using the following format, where 1146 is the mysql_errno() value and '42S02' is the corresponding mysql_sqlstate() value:

```
shell> SELECT * FROM no_such_table;
ERROR 1146 (42S02): Table 'test.no_such_table' doesn't exist
```

Return Values

An error code value for the last mysql_*xxx*() call, if it failed. zero means no error occurred.

Errors

None.

21.8.7.15 mysql_error()

`const char *mysql_error(MYSQL *mysql)`

Description

For the connection specified by mysql, mysql_error() returns a null-terminated string containing the error message for the most recently invoked API function that failed. If a function did not fail, the return value of mysql_error() may be the previous error or an empty string to indicate no error.

A rule of thumb is that all functions that have to ask the server for information reset mysql_error() if they succeed.

For functions that reset mysql_error(), either of these two tests can be used to check for an error:

```
if(*mysql_error(&mysql))
{
  // an error occurred
}

if(mysql_error(&mysql)[0])
{
  // an error occurred
}
```

The language of the client error messages may be changed by recompiling the MySQL client library. Currently, you can choose error messages in several different languages. See Section 10.2, “Setting the Error Message Language”.

Return Values

A null-terminated character string that describes the error. An empty string if no error occurred.

Errors

None.

21.8.7.16 mysql_escape_string()

Use mysql_real_escape_string() instead!

This function is identical to mysql_real_escape_string() except that mysql_real_escape_string() takes a connection handler as its first argument and escapes the string according to the current character set. mysql_escape_string() does not take a connection argument and does not respect the current character set.

21.8.7.17 mysql_fetch_field()

MYSQL_FIELD *mysql_fetch_field(MYSQL_RES *result)

Description

Returns the definition of one column of a result set as a MYSQL_FIELD structure. Call this function repeatedly to retrieve information about all columns in the result set. mysql_fetch_field() returns NULL when no more fields are left.

mysql_fetch_field() is reset to return information about the first field each time you execute a new SELECT query. The field returned by mysql_fetch_field() is also affected by calls to mysql_field_seek().

If you've called mysql_query() to perform a SELECT on a table but have not called mysql_store_result(), MySQL returns the default blob length (8KB) if you call mysql_fetch_field() to ask for the length of a BLOB field. (The 8KB size is chosen because MySQL does not know the maximum length for the BLOB. This should be made configurable sometime.) Once you've retrieved the result set, field->max_length contains the length of the largest value for this column in the specific query.

Return Values

The MYSQL_FIELD structure for the current column. NULL if no columns are left.

Errors

None.

Example

```
MYSQL_FIELD *field;

while((field = mysql_fetch_field(result)))
{
    printf("field name %s\n", field->name);
}
```

21.8.7.18 mysql_fetch_field_direct()

MYSQL_FIELD *mysql_fetch_field_direct(MYSQL_RES *result, unsigned int fieldnr)

Description

Given a field number fieldnr for a column within a result set, returns that column's field definition as a MYSQL_FIELD structure. Use this function to retrieve the definition for an arbitrary column. Specify a value for fieldnr in the range from 0 to mysql_num_fields(result)-1.

Return Values

The MYSQL_FIELD structure for the specified column.

Errors

None.

Example

```
unsigned int num_fields;
unsigned int i;
MYSQL_FIELD *field;

num_fields = mysql_num_fields(result);
for(i = 0; i < num_fields; i++)
{
    field = mysql_fetch_field_direct(result, i);
    printf("Field %u is %s\n", i, field->name);
}
```

21.8.7.19 mysql_fetch_fields()

```
MYSQL_FIELD *mysql_fetch_fields(MYSQL_RES *result)
```

Description

Returns an array of all `MYSQL_FIELD` structures for a result set. Each structure provides the field definition for one column of the result set.

Return Values

An array of `MYSQL_FIELD` structures for all columns of a result set.

Errors

None.

Example

```
unsigned int num_fields;
unsigned int i;
MYSQL_FIELD *fields;

num_fields = mysql_num_fields(result);
fields = mysql_fetch_fields(result);
for(i = 0; i < num_fields; i++)
{
   printf("Field %u is %s\n", i, fields[i].name);
}
```

21.8.7.20 mysql_fetch_lengths()

```
unsigned long *mysql_fetch_lengths(MYSQL_RES *result)
```

Description

Returns the lengths of the columns of the current row within a result set. If you plan to copy field values, this length information is also useful for optimization, because you can avoid calling `strlen()`. In addition, if the result set contains binary data, you **must** use this function to determine the size of the data, because `strlen()` returns incorrect results for any field containing null characters.

The length for empty columns and for columns containing `NULL` values is zero. To see how to distinguish these two cases, see the description for `mysql_fetch_row()`.

Return Values

An array of unsigned long integers representing the size of each column (not including any terminating null characters). NULL if an error occurred.

Errors

mysql_fetch_lengths() is valid only for the current row of the result set. It returns NULL if you call it before calling mysql_fetch_row() or after retrieving all rows in the result.

Example

```
MYSQL_ROW row;
unsigned long *lengths;
unsigned int num_fields;
unsigned int i;

row = mysql_fetch_row(result);
if (row)
{
    num_fields = mysql_num_fields(result);
    lengths = mysql_fetch_lengths(result);
    for(i = 0; i < num_fields; i++)
    {
         printf("Column %u is %lu bytes in length.\n",
                i, lengths[i]);
    }
}
```

21.8.7.21 mysql_fetch_row()

MYSQL_ROW mysql_fetch_row(MYSQL_RES *result)

Description

Retrieves the next row of a result set. When used after mysql_store_result(), mysql_fetch_row() returns NULL when there are no more rows to retrieve. When used after mysql_use_result(), mysql_fetch_row() returns NULL when there are no more rows to retrieve or if an error occurred.

The number of values in the row is given by mysql_num_fields(result). If row holds the return value from a call to mysql_fetch_row(), pointers to the values are accessed as row[0] to row[mysql_num_fields(result)-1]. NULL values in the row are indicated by NULL pointers.

The lengths of the field values in the row may be obtained by calling mysql_fetch_lengths(). Empty fields and fields containing NULL both have length 0; you can distinguish these by checking the pointer for the field value. If the pointer is NULL, the field is NULL; otherwise, the field is empty.

Return Values

A MYSQL_ROW structure for the next row. NULL if there are no more rows to retrieve or if an error occurred.

Errors

Note that error is not reset between calls to mysql_fetch_row()

- CR_SERVER_LOST

  The connection to the server was lost during the query.

- CR_UNKNOWN_ERROR

  An unknown error occurred.

Example

```
MYSQL_ROW row;
unsigned int num_fields;
unsigned int i;

num_fields = mysql_num_fields(result);
while ((row = mysql_fetch_row(result)))
{
   unsigned long *lengths;
   lengths = mysql_fetch_lengths(result);
   for(i = 0; i < num_fields; i++)
   {
       printf("[%.*s] ", (int) lengths[i],
              row[i] ? row[i] : "NULL");
   }
   printf("\n");
}
```

21.8.7.22 mysql_field_count()

`unsigned int mysql_field_count(MYSQL *mysql)`

Description

Returns the number of columns for the most recent query on the connection.

The normal use of this function is when mysql_store_result() returned NULL (and thus you have no result set pointer). In this case, you can call mysql_field_count() to determine whether mysql_store_result() should have produced a nonempty result. This enables the client program to take proper action without knowing whether the query was a SELECT (or SELECT-like) statement. The example shown here illustrates how this may be done.

See Section 21.8.15.1, “Why mysql_store_result() Sometimes Returns NULL After mysql_query() Returns Success”.

Return Values

An unsigned integer representing the number of columns in a result set.

Errors

None.

Example

```
MYSQL_RES *result;
unsigned int num_fields;
unsigned int num_rows;

if (mysql_query(&mysql,query_string))
{
    // error
}
else // query succeeded, process any data returned by it
{
    result = mysql_store_result(&mysql);
    if (result)  // there are rows
    {
        num_fields = mysql_num_fields(result);
        // retrieve rows, then call mysql_free_result(result)
    }
    else  // mysql_store_result() returned nothing; should it have?
    {
        if(mysql_field_count(&mysql) == 0)
        {
            // query does not return data
            // (it was not a SELECT)
            num_rows = mysql_affected_rows(&mysql);
        }
        else // mysql_store_result() should have returned data
        {
            fprintf(stderr, "Error: %s\n", mysql_error(&mysql));
        }
    }
}
```

An alternative is to replace the `mysql_field_count(&mysql)` call with `mysql_errno(&mysql)`. In this case, you are checking directly for an error from `mysql_store_result()` rather than inferring from the value of `mysql_field_count()` whether the statement was a `SELECT`.

21.8.7.23 `mysql_field_seek()`

`MYSQL_FIELD_OFFSET mysql_field_seek(MYSQL_RES *result, MYSQL_FIELD_OFFSET offset)`

Description

Sets the field cursor to the given offset. The next call to `mysql_fetch_field()` retrieves the field definition of the column associated with that offset.

To seek to the beginning of a row, pass an `offset` value of zero.

Return Values

The previous value of the field cursor.

Errors

None.

21.8.7.24 `mysql_field_tell()`

`MYSQL_FIELD_OFFSET mysql_field_tell(MYSQL_RES *result)`

Description

Returns the position of the field cursor used for the last `mysql_fetch_field()`. This value can be used as an argument to `mysql_field_seek()`.

Return Values

The current offset of the field cursor.

Errors

None.

21.8.7.25 mysql_free_result()

```
void mysql_free_result(MYSQL_RES *result)
```

Description

Frees the memory allocated for a result set by mysql_store_result(), mysql_use_result(), mysql_list_dbs(), and so forth. When you are done with a result set, you must free the memory it uses by calling mysql_free_result().

Do not attempt to access a result set after freeing it.

Return Values

None.

Errors

None.

21.8.7.26 mysql_get_character_set_info()

```
void mysql_get_character_set_info(MYSQL *mysql, MY_CHARSET_INFO *cs)
```

Description

This function provides information about the default client character set. The default character set may be changed with the mysql_set_character_set() function.

Example

This example shows the fields that are available in the MY_CHARSET_INFO structure:

```
if (!mysql_set_character_set(&mysql, "utf8"))
{
    MY_CHARSET_INFO cs;
    mysql_get_character_set_info(&mysql, &cs);
    printf("character set information:\n");
    printf("character set+collation number: %d\n", cs.number);
    printf("character set name: %s\n", cs.name);
    printf("collation name: %s\n", cs.csname);
    printf("comment: %s\n", cs.comment);
    printf("directory: %s\n", cs.dir);
    printf("multi byte character min. length: %d\n", cs.mbminlen);
    printf("multi byte character max. length: %d\n", cs.mbmaxlen);
}
```

21.8.7.27 mysql_get_client_info()

```
const char *mysql_get_client_info(void)
```

Description

Returns a string that represents the MySQL client library version; for example, "5.7.5".

As of MySQL 5.7.4 and Connector/C 6.1.3, the function value is the version of MySQL or Connector/C that provides the client library. Before MySQL 5.7.4 and Connector/C 6.1.3, the function value is the MySQL version. For Connector/C, this is the MySQL version on which the Connector/C distribution is based. For more information, see Section 21.8.4.4, “C API Server and Client Library Versions”.

Return Values

A character string that represents the MySQL client library version.

Errors

None.

21.8.7.28 mysql_get_client_version()

`unsigned long mysql_get_client_version(void)`

Description

Returns an integer that represents the MySQL client library version. The value has the format `XYYZZ` where `X` is the major version, `YY` is the release level (or minor version), and `ZZ` is the sub-version within the release level:

```
major_version*10000 + release_level*100 + sub_version
```

For example, `"5.7.5"` is returned as `50705`.

As of MySQL 5.7.4 and Connector/C 6.1.3, the function value is the version of MySQL or Connector/C that provides the client library. Before MySQL 5.7.4 and Connector/C 6.1.3, the function value is the MySQL version. For Connector/C, this is the MySQL version on which the Connector/C distribution is based. For more information, see Section 21.8.4.4, “C API Server and Client Library Versions”.

Return Values

An integer that represents the MySQL client library version.

Errors

None.

21.8.7.29 mysql_get_host_info()

`const char *mysql_get_host_info(MYSQL *mysql)`

Description

Returns a string describing the type of connection in use, including the server host name.

Return Values

A character string representing the server host name and the connection type.

Errors

None.

21.8.7.30 mysql_get_option()

`int mysql_get_option(MYSQL *mysql, enum mysql_option option, const void *arg)`

Description

Returns the current value of an option settable using `mysql_options()`. The value should be treated as read only. This function was added in MySQL 5.7.3.

The option argument is the option for which you want its value. The arg argument is a pointer to a variable in which to store the option value. arg must be a pointer to a variable of the type appropriate for the option argument. The following table shows which variable type to use for each option value.

arg Type	Applicable option Values
unsigned int	MYSQL_OPT_CONNECT_TIMEOUT, MYSQL_OPT_PROTOCOL, MYSQL_OPT_READ_TIMEOUT, MYSQL_OPT_WRITE_TIMEOUT
my_bool	MYSQL_ENABLE_CLEARTEXT_PLUGIN, MYSQL_OPT_CAN_HANDLE_EXPIRED_PASSWORDS, MYSQL_OPT_COMPRESS, MYSQL_OPT_GUESS_CONNECTION, MYSQL_OPT_LOCAL_INFILE, MYSQL_OPT_RECONNECT, MYSQL_OPT_SSL_ENFORCE, MYSQL_OPT_SSL_VERIFY_SERVER_CERT, MYSQL_OPT_USE_EMBEDDED_CONNECTION, MYSQL_OPT_USE_REMOTE_CONNECTION, MYSQL_REPORT_DATA_TRUNCATION, MYSQL_SECURE_AUTH
const char *	MYSQL_DEFAULT_AUTH, MYSQL_OPT_BIND, MYSQL_OPT_SSL_CA, MYSQL_OPT_SSL_CAPATH , MYSQL_OPT_SSL_CERT, MYSQL_OPT_SSL_CIPHER, MYSQL_OPT_SSL_CRL, MYSQL_OPT_SSL_CRLPATH, MYSQL_OPT_SSL_KEY, MYSQL_PLUGIN_DIR, MYSQL_READ_DEFAULT_FILE, MYSQL_READ_DEFAULT_GROUP, MYSQL_SERVER_PUBLIC_KEY, MYSQL_SET_CHARSET_DIR, MYSQL_SET_CHARSET_NAME, MYSQL_SET_CLIENT_IP, MYSQL_SHARED_MEMORY_BASE_NAME
cannot be queried (error is returned)	MYSQL_INIT_COMMAND, MYSQL_OPT_CONNECT_ATTR_DELETE, MYSQL_OPT_CONNECT_ATTR_RESET, MYSQL_OPT_NAMED_PIPE

Return Values

Zero for success. Nonzero if an error occurred; this occurs for option values that cannot be queried.

Example

The following call tests the MYSQL_OPT_RECONNECT option. After the call returns successfully, the value of reconnect is true or false to indicate whether automatic reconnection is enabled.

```
my_bool reconnect;

if (mysql_get_option(mysql, MYSQL_OPT_RECONNECT, &reconnect))
  fprintf(stderr, "mysql_get_options() failed\n");
```

21.8.7.31 mysql_get_proto_info()

unsigned int mysql_get_proto_info(MYSQL *mysql)

Description

Returns the protocol version used by current connection.

Return Values

An unsigned integer representing the protocol version used by the current connection.

Errors

None.

21.8.7.32 `mysql_get_server_info()`

```
const char *mysql_get_server_info(MYSQL *mysql)
```

Description

Returns a string that represents the MySQL server version; for example, `"5.7.5"`.

Return Values

A character string that represents the MySQL server version.

Errors

None.

21.8.7.33 `mysql_get_server_version()`

```
unsigned long mysql_get_server_version(MYSQL *mysql)
```

Description

Returns an integer that represents the MySQL server version. The value has the format `XYYZZ` where `X` is the major version, `YY` is the release level (or minor version), and `ZZ` is the sub-version within the release level:

```
major_version*10000 + release_level*100 + sub_version
```

For example, `"5.7.5"` is returned as `50705`.

This function is useful in client programs for determining whether some version-specific server capability exists.

Return Values

An integer that represents the MySQL server version.

Errors

None.

21.8.7.34 `mysql_get_ssl_cipher()`

```
const char *mysql_get_ssl_cipher(MYSQL *mysql)
```

Description

`mysql_get_ssl_cipher()` returns the SSL cipher used for the given connection to the server. `mysql` is the connection handler returned from `mysql_init()`.

Return Values

A string naming the SSL cipher used for the connection, or `NULL` if no cipher is being used.

21.8.7.35 `mysql_hex_string()`

```
unsigned long mysql_hex_string(char *to, const char *from, unsigned long
length)
```

Description

This function is used to create a legal SQL string that you can use in an SQL statement. See Section 9.1.1, “String Literals”.

The string in `from` is encoded to hexadecimal format, with each character encoded as two hexadecimal digits. The result is placed in `to` and a terminating null byte is appended.

The string pointed to by `from` must be `length` bytes long. You must allocate the `to` buffer to be at least `length*2+1` bytes long. When `mysql_hex_string()` returns, the contents of `to` is a null-terminated string. The return value is the length of the encoded string, not including the terminating null character.

The return value can be placed into an SQL statement using either `0x`*`value`* or `X'`*`value`*`'` format. However, the return value does not include the `0x` or `X'...'`. The caller must supply whichever of those is desired.

Example

```
char query[1000],*end;

end = strmov(query,"INSERT INTO test_table values(");
end = strmov(end,"0x");
end += mysql_hex_string(end,"What is this",12);
end = strmov(end,",0x");
end += mysql_hex_string(end,"binary data: \0\r\n",16);
*end++ = ')';

if (mysql_real_query(&mysql,query,(unsigned int) (end - query)))
{
   fprintf(stderr, "Failed to insert row, Error: %s\n",
           mysql_error(&mysql));
}
```

The `strmov()` function used in the example is included in the `libmysqlclient` library and works like `strcpy()` but returns a pointer to the terminating null of the first parameter.

Return Values

The length of the value placed into `to`, not including the terminating null character.

Errors

None.

21.8.7.36 `mysql_info()`

`const char *mysql_info(MYSQL *mysql)`

Description

Retrieves a string providing information about the most recently executed statement, but only for the statements listed here. For other statements, `mysql_info()` returns `NULL`. The format of the string varies depending on the type of statement, as described here. The numbers are illustrative only; the string contains values appropriate for the statement.

- `INSERT INTO ... SELECT ...`

  String format: `Records: 100 Duplicates: 0 Warnings: 0`

- `INSERT INTO ... VALUES (...),(...),(...)...`

  String format: `Records: 3 Duplicates: 0 Warnings: 0`
- `LOAD DATA INFILE ...`

  String format: `Records: 1 Deleted: 0 Skipped: 0 Warnings: 0`
- `ALTER TABLE`

  String format: `Records: 3 Duplicates: 0 Warnings: 0`
- `UPDATE`

  String format: `Rows matched: 40 Changed: 40 Warnings: 0`

Note that `mysql_info()` returns a non-`NULL` value for `INSERT ... VALUES` only for the multiple-row form of the statement (that is, only if multiple value lists are specified).

Return Values

A character string representing additional information about the most recently executed statement. `NULL` if no information is available for the statement.

Errors

None.

21.8.7.37 `mysql_init()`

```
MYSQL *mysql_init(MYSQL *mysql)
```

Description

Allocates or initializes a `MYSQL` object suitable for `mysql_real_connect()`. If `mysql` is a `NULL` pointer, the function allocates, initializes, and returns a new object. Otherwise, the object is initialized and the address of the object is returned. If `mysql_init()` allocates a new object, it is freed when `mysql_close()` is called to close the connection.

Return Values

An initialized `MYSQL*` handle. `NULL` if there was insufficient memory to allocate a new object.

Errors

In case of insufficient memory, `NULL` is returned.

21.8.7.38 `mysql_insert_id()`

```
my_ulonglong mysql_insert_id(MYSQL *mysql)
```

Description

Returns the value generated for an `AUTO_INCREMENT` column by the previous `INSERT` or `UPDATE` statement. Use this function after you have performed an `INSERT` statement into a table that contains an `AUTO_INCREMENT` field, or have used `INSERT` or `UPDATE` to set a column value with `LAST_INSERT_ID(`*`expr`*`)`.

The return value of mysql_insert_id() is always zero unless explicitly updated under one of the following conditions:

- INSERT statements that store a value into an AUTO_INCREMENT column. This is true whether the value is automatically generated by storing the special values NULL or 0 into the column, or is an explicit nonspecial value.
- In the case of a multiple-row INSERT statement, mysql_insert_id() returns the first automatically generated AUTO_INCREMENT value that was successfully inserted.

  If no rows are successfully inserted, mysql_insert_id() returns 0.
- If an INSERT ... SELECT statement is executed, and no automatically generated value is successfully inserted, mysql_insert_id() returns the ID of the last inserted row.
- If an INSERT ... SELECT statement uses LAST_INSERT_ID(*expr*), mysql_insert_id() returns *expr*.
- INSERT statements that generate an AUTO_INCREMENT value by inserting LAST_INSERT_ID(*expr*) into any column or by updating any column to LAST_INSERT_ID(*expr*).
- If the previous statement returned an error, the value of mysql_insert_id() is undefined.

The return value of mysql_insert_id() can be simplified to the following sequence:

1. If there is an AUTO_INCREMENT column, and an automatically generated value was successfully inserted, return the first such value.
2. If LAST_INSERT_ID(*expr*) occurred in the statement, return *expr*, even if there was an AUTO_INCREMENT column in the affected table.
3. The return value varies depending on the statement used. When called after an INSERT statement:

   - If there is an AUTO_INCREMENT column in the table, and there were some explicit values for this column that were successfully inserted into the table, return the last of the explicit values.

   When called after an INSERT ... ON DUPLICATE KEY UPDATE statement:

   - If there is an AUTO_INCREMENT column in the table and there were some explicit successfully inserted values or some updated values, return the last of the inserted or updated values.

mysql_insert_id() returns 0 if the previous statement does not use an AUTO_INCREMENT value. If you need to save the value for later, be sure to call mysql_insert_id() immediately after the statement that generates the value.

The value of mysql_insert_id() is affected only by statements issued within the current client connection. It is not affected by statements issued by other clients.

The LAST_INSERT_ID() SQL function will contain the value of the first automatically generated value that was successfully inserted. LAST_INSERT_ID() is not reset between statements because the value of that function is maintained in the server. Another difference from mysql_insert_id() is that LAST_INSERT_ID() is not updated if you set an AUTO_INCREMENT column to a specific nonspecial value. See Section 12.14, “Information Functions”.

mysql_insert_id() returns 0 following a CALL statement for a stored procedure that generates an AUTO_INCREMENT value because in this case mysql_insert_id() applies to CALL and not the statement within the procedure. Within the procedure, you can use LAST_INSERT_ID() at the SQL level to obtain the AUTO_INCREMENT value.

The reason for the differences between LAST_INSERT_ID() and mysql_insert_id() is that LAST_INSERT_ID() is made easy to use in scripts while mysql_insert_id() tries to provide more exact information about what happens to the AUTO_INCREMENT column.

Return Values

Described in the preceding discussion.

Errors

None.

21.8.7.39 mysql_kill()

```
int mysql_kill(MYSQL *mysql, unsigned long pid)
```

Description

Asks the server to kill the thread specified by pid.

This function is deprecated. It is preferable to use mysql_query() to issue an SQL KILL statement instead.

mysql_kill() cannot handle values larger than 32 bits, but to guard against killing the wrong thread returns an error in these cases:

- If given an ID larger than 32 bits, mysql_kill() returns a CR_INVALID_CONN_HANDLE error.
- After the server's internal thread ID counter reaches a value larger than 32 bits, it returns an ER_DATA_OUT_OF_RANGE error for any mysql_kill() invocation and mysql_kill() fails.

Return Values

Zero for success. Nonzero if an error occurred.

Errors

- CR_COMMANDS_OUT_OF_SYNC

  Commands were executed in an improper order.
- CR_INVALID_CONN_HANDLE

  The pid was larger than 32 bits.
- CR_SERVER_GONE_ERROR

  The MySQL server has gone away.
- CR_SERVER_LOST

  The connection to the server was lost during the query.
- CR_UNKNOWN_ERROR

  An unknown error occurred.
- ER_DATA_OUT_OF_RANGE

The server's internal thread ID counter has reached a value larger than 32 bits, at which point it rejects all mysql_kill() invocations.

21.8.7.40 mysql_library_end()

```
void mysql_library_end(void)
```

Description

This function finalizes the MySQL library. Call it when you are done using the library (for example, after disconnecting from the server). The action taken by the call depends on whether your application is linked to the MySQL client library or the MySQL embedded server library. For a client program linked against the libmysqlclient library by using the -lmysqlclient flag, mysql_library_end() performs some memory management to clean up. For an embedded server application linked against the libmysqld library by using the -lmysqld flag, mysql_library_end() shuts down the embedded server and then cleans up.

For usage information, see Section 21.8.6, "C API Function Overview", and Section 21.8.7.41, "mysql_library_init()".

21.8.7.41 mysql_library_init()

```
int mysql_library_init(int argc, char **argv, char **groups)
```

Description

Call this function to initialize the MySQL library before you call any other MySQL function, whether your application is a regular client program or uses the embedded server. If the application uses the embedded server, this call starts the server and initializes any subsystems (mysys, InnoDB, and so forth) that the server uses.

After your application is done using the MySQL library, call mysql_library_end() to clean up. See Section 21.8.7.40, "mysql_library_end()".

The choice of whether the application operates as a regular client or uses the embedded server depends on whether you use the libmysqlclient or libmysqld library at link time to produce the final executable. For additional information, see Section 21.8.6, "C API Function Overview".

In a nonmulti-threaded environment, the call to mysql_library_init() may be omitted, because mysql_init() will invoke it automatically as necessary. However, mysql_library_init() is not thread-safe in a multi-threaded environment, and thus neither is mysql_init(), which calls mysql_library_init(). You must either call mysql_library_init() prior to spawning any threads, or else use a mutex to protect the call, whether you invoke mysql_library_init() or indirectly through mysql_init(). Do this prior to any other client library call.

The argc and argv arguments are analogous to the arguments to main(), and enable passing of options to the embedded server. For convenience, argc may be 0 (zero) if there are no command-line arguments for the server. This is the usual case for applications intended for use only as regular (nonembedded) clients, and the call typically is written as mysql_library_init(0, NULL, NULL).

```
#include <mysql.h>
#include <stdlib.h>

int main(void) {
  if (mysql_library_init(0, NULL, NULL)) {
    fprintf(stderr, "could not initialize MySQL library\n");
    exit(1);
  }

  /* Use any MySQL API functions here */

  mysql_library_end();

  return EXIT_SUCCESS;
}
```

When arguments are to be passed (`argc` is greater than `0`), the first element of `argv` is ignored (it typically contains the program name). `mysql_library_init()` makes a copy of the arguments so it is safe to destroy `argv` or `groups` after the call.

For embedded applications, if you want to connect to an external server without starting the embedded server, you have to specify a negative value for `argc`.

The `groups` argument is an array of strings that indicate the groups in option files from which to read options. See Section 4.2.3.3, “Using Option Files”. Make the final entry in the array `NULL`. For convenience, if the `groups` argument itself is `NULL`, the `[server]` and `[embedded]` groups are used by default.

```
#include <mysql.h>
#include <stdlib.h>

static char *server_args[] = {
  "this_program",       /* this string is not used */
  "--datadir=.",
  "--key_buffer_size=32M"
};
static char *server_groups[] = {
  "embedded",
  "server",
  "this_program_SERVER",
  (char *)NULL
};

int main(void) {
  if (mysql_library_init(sizeof(server_args) / sizeof(char *),
                        server_args, server_groups)) {
    fprintf(stderr, "could not initialize MySQL library\n");
    exit(1);
  }

  /* Use any MySQL API functions here */

  mysql_library_end();

  return EXIT_SUCCESS;
}
```

Return Values

Zero for success. Nonzero if an error occurred.

21.8.7.42 `mysql_list_dbs()`

`MYSQL_RES *mysql_list_dbs(MYSQL *mysql, const char *wild)`

Description

Returns a result set consisting of database names on the server that match the simple regular expression specified by the `wild` parameter. `wild` may contain the wildcard characters “`%`” or “`_`”, or may be a `NULL` pointer to match all databases. Calling `mysql_list_dbs()` is similar to executing the query `SHOW DATABASES [LIKE wild]`.

You must free the result set with `mysql_free_result()`.

Return Values

A `MYSQL_RES` result set for success. `NULL` if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC`

  Commands were executed in an improper order.

- `CR_OUT_OF_MEMORY`

  Out of memory.

- `CR_SERVER_GONE_ERROR`

  The MySQL server has gone away.

- `CR_SERVER_LOST`

  The connection to the server was lost during the query.

- `CR_UNKNOWN_ERROR`

  An unknown error occurred.

21.8.7.43 `mysql_list_fields()`

```
MYSQL_RES *mysql_list_fields(MYSQL *mysql, const char *table, const char *wild)
```

Description

Returns an empty result set for which the metadata provides information aobut the columns in the given table that match the simple regular expression specified by the `wild` parameter. `wild` may contain the wildcard characters “`%`” or “`_`”, or may be a `NULL` pointer to match all fields. Calling `mysql_list_fields()` is similar to executing the query `SHOW COLUMNS FROM tbl_name [LIKE wild]`.

It is preferable to use `SHOW COLUMNS FROM tbl_name` instead of `mysql_list_fields()`.

You must free the result set with `mysql_free_result()`.

Return Values

A `MYSQL_RES` result set for success. `NULL` if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC`

  Commands were executed in an improper order.

- `CR_SERVER_GONE_ERROR`

  The MySQL server has gone away.

- CR_SERVER_LOST

  The connection to the server was lost during the query.

- CR_UNKNOWN_ERROR

  An unknown error occurred.

Example

```
int i;
MYSQL_RES *tbl_cols = mysql_list_fields(mysql, "mytbl", "f%");

unsigned int field_cnt = mysql_num_fields(tbl_cols);
printf("Number of columns: %d\n", field_cnt);

for (i=0; i < field_cnt; ++i)
{
  /* col describes i-th column of the table */
  MYSQL_FIELD *col = mysql_fetch_field_direct(tbl_cols, i);
  printf ("Column %d: %s\n", i, col->name);
}
mysql_free_result(tbl_cols);
```

21.8.7.44 mysql_list_processes()

MYSQL_RES *mysql_list_processes(MYSQL *mysql)

Description

Returns a result set describing the current server threads. This is the same kind of information as that reported by mysqladmin processlist or a SHOW PROCESSLIST query.

You must free the result set with mysql_free_result().

Return Values

A MYSQL_RES result set for success. NULL if an error occurred.

Errors

- CR_COMMANDS_OUT_OF_SYNC

  Commands were executed in an improper order.

- CR_SERVER_GONE_ERROR

  The MySQL server has gone away.

- CR_SERVER_LOST

  The connection to the server was lost during the query.

- CR_UNKNOWN_ERROR

  An unknown error occurred.

21.8.7.45 mysql_list_tables()

MYSQL_RES *mysql_list_tables(MYSQL *mysql, const char *wild)

Description

Returns a result set consisting of table names in the current database that match the simple regular expression specified by the `wild` parameter. `wild` may contain the wildcard characters "`%`" or "`_`", or may be a `NULL` pointer to match all tables. Calling `mysql_list_tables()` is similar to executing the query `SHOW TABLES [LIKE wild]`.

You must free the result set with `mysql_free_result()`.

Return Values

A `MYSQL_RES` result set for success. `NULL` if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC`

  Commands were executed in an improper order.

- `CR_SERVER_GONE_ERROR`

  The MySQL server has gone away.

- `CR_SERVER_LOST`

  The connection to the server was lost during the query.

- `CR_UNKNOWN_ERROR`

  An unknown error occurred.

21.8.7.46 `mysql_more_results()`

```
my_bool mysql_more_results(MYSQL *mysql)
```

Description

This function is used when you execute multiple statements specified as a single statement string, or when you execute `CALL` statements, which can return multiple result sets.

`mysql_more_results()` true if more results exist from the currently executed statement, in which case the application must call `mysql_next_result()` to fetch the results.

Return Values

`TRUE` (1) if more results exist. `FALSE` (0) if no more results exist.

In most cases, you can call `mysql_next_result()` instead to test whether more results exist and initiate retrieval if so.

See Section 21.8.17, "C API Support for Multiple Statement Execution", and Section 21.8.7.47, "`mysql_next_result()`".

Errors

None.

21.8.7.47 `mysql_next_result()`

```
int mysql_next_result(MYSQL *mysql)
```

Description

This function is used when you execute multiple statements specified as a single statement string, or when you use `CALL` statements to execute stored procedures, which can return multiple result sets.

`mysql_next_result()` reads the next statement result and returns a status to indicate whether more results exist. If `mysql_next_result()` returns an error, there are no more results.

Before each call to `mysql_next_result()`, you must call `mysql_free_result()` for the current statement if it is a statement that returned a result set (rather than just a result status).

After calling `mysql_next_result()` the state of the connection is as if you had called `mysql_real_query()` or `mysql_query()` for the next statement. This means that you can call `mysql_store_result()`, `mysql_warning_count()`, `mysql_affected_rows()`, and so forth.

If your program uses `CALL` statements to execute stored procedures, the `CLIENT_MULTI_RESULTS` flag must be enabled. This is because each `CALL` returns a result to indicate the call status, in addition to any result sets that might be returned by statements executed within the procedure. Because `CALL` can return multiple results, process them using a loop that calls `mysql_next_result()` to determine whether there are more results.

`CLIENT_MULTI_RESULTS` can be enabled when you call `mysql_real_connect()`, either explicitly by passing the `CLIENT_MULTI_RESULTS` flag itself, or implicitly by passing `CLIENT_MULTI_STATEMENTS` (which also enables `CLIENT_MULTI_RESULTS`). In MySQL 5.7, `CLIENT_MULTI_RESULTS` is enabled by default.

It is also possible to test whether there are more results by calling `mysql_more_results()`. However, this function does not change the connection state, so if it returns true, you must still call `mysql_next_result()` to advance to the next result.

For an example that shows how to use `mysql_next_result()`, see Section 21.8.17, “C API Support for Multiple Statement Execution”.

Return Values

Return Value	Description
0	Successful and there are more results
-1	Successful and there are no more results
>0	An error occurred

Errors

- `CR_COMMANDS_OUT_OF_SYNC`

  Commands were executed in an improper order. For example, if you did not call `mysql_use_result()` for a previous result set.

- `CR_SERVER_GONE_ERROR`

  The MySQL server has gone away.

- `CR_SERVER_LOST`

  The connection to the server was lost during the query.

- CR_UNKNOWN_ERROR

  An unknown error occurred.

21.8.7.48 mysql_num_fields()

unsigned int mysql_num_fields(MYSQL_RES *result)

To pass a MYSQL* argument instead, use unsigned int mysql_field_count(MYSQL *mysql).

Description

Returns the number of columns in a result set.

Note that you can get the number of columns either from a pointer to a result set or to a connection handle. You would use the connection handle if mysql_store_result() or mysql_use_result() returned NULL (and thus you have no result set pointer). In this case, you can call mysql_field_count() to determine whether mysql_store_result() should have produced a nonempty result. This enables the client program to take proper action without knowing whether the query was a SELECT (or SELECT-like) statement. The example shown here illustrates how this may be done.

See Section 21.8.15.1, "Why mysql_store_result() Sometimes Returns NULL After mysql_query() Returns Success".

Return Values

An unsigned integer representing the number of columns in a result set.

Errors

None.

Example

```
MYSQL_RES *result;
unsigned int num_fields;
unsigned int num_rows;

if (mysql_query(&mysql,query_string))
{
    // error
}
else // query succeeded, process any data returned by it
{
    result = mysql_store_result(&mysql);
    if (result)  // there are rows
    {
        num_fields = mysql_num_fields(result);
        // retrieve rows, then call mysql_free_result(result)
    }
    else  // mysql_store_result() returned nothing; should it have?
    {
        if (mysql_errno(&mysql))
        {
           fprintf(stderr, "Error: %s\n", mysql_error(&mysql));
        }
        else if (mysql_field_count(&mysql) == 0)
        {
            // query does not return data
            // (it was not a SELECT)
            num_rows = mysql_affected_rows(&mysql);
        }
    }
}
```

An alternative (if you know that your query should have returned a result set) is to replace the `mysql_errno(&mysql)` call with a check whether `mysql_field_count(&mysql)` returns 0. This happens only if something went wrong.

21.8.7.49 `mysql_num_rows()`

`my_ulonglong mysql_num_rows(MYSQL_RES *result)`

Description

Returns the number of rows in the result set.

The use of `mysql_num_rows()` depends on whether you use `mysql_store_result()` or `mysql_use_result()` to return the result set. If you use `mysql_store_result()`, `mysql_num_rows()` may be called immediately. If you use `mysql_use_result()`, `mysql_num_rows()` does not return the correct value until all the rows in the result set have been retrieved.

`mysql_num_rows()` is intended for use with statements that return a result set, such as `SELECT`. For statements such as `INSERT`, `UPDATE`, or `DELETE`, the number of affected rows can be obtained with `mysql_affected_rows()`.

Return Values

The number of rows in the result set.

Errors

None.

21.8.7.50 `mysql_options()`

`int mysql_options(MYSQL *mysql, enum mysql_option option, const void *arg)`

Description

Can be used to set extra connect options and affect behavior for a connection. This function may be called multiple times to set several options.

Call `mysql_options()` after `mysql_init()` and before `mysql_connect()` or `mysql_real_connect()`.

The `option` argument is the option that you want to set; the `arg` argument is the value for the option. If the option is an integer, specify a pointer to the value of the integer as the `arg` argument.

The following list describes the possible options, their effect, and how `arg` is used for each option. Several of the options apply only when the application is linked against the `libmysqld` embedded server library and are unused for applications linked against the `libmysqlclient` client library. For option descriptions that indicate `arg` is unused, its value is irrelevant; it is conventional to pass 0.

- `MYSQL_DEFAULT_AUTH` (argument type: `char *`)

  The name of the authentication plugin to use.

- `MYSQL_ENABLE_CLEARTEXT_PLUGIN` (argument type: `my_bool *`)

Enable the mysql_clear_password cleartext authentication plugin. (See Section 6.3.9.5, “The Cleartext Client-Side Authentication Plugin”.)

- MYSQL_INIT_COMMAND (argument type: char *)

  SQL statement to execute when connecting to the MySQL server. Automatically re-executed if reconnection occurs.

- MYSQL_OPT_BIND (argument: char *)

  The network interface from which to connect to the server. This is used when the client host has multiple network interfaces. The argument is a host name or IP address (specified as a string).

- MYSQL_OPT_CAN_HANDLE_EXPIRED_PASSWORDS (argument type: my_bool *)

  Indicate whether the client can handle expired passwords. For more information, see Section 6.3.7, “Password Expiration and Sandbox Mode”.

- MYSQL_OPT_COMPRESS (argument: not used)

  Use the compressed client/server protocol.

- MYSQL_OPT_CONNECT_ATTR_DELETE (argument types: char *)

  Given a key name, this option deletes a key/value pair from the current set of connection attributes to be passed to the server at connect time. The argument is a pointer to a null-terminated string naming the key. Comparison of the key name with existing keys is case sensitive.

  See also the description for the MYSQL_OPT_CONNECT_ATTR_RESET option, as well as the description for the MYSQL_OPT_CONNECT_ATTR_ADD option in the description of the mysql_options4() function. That function description also includes a usage example.

  Connection attributes are exposed through the session_connect_attrs and session_account_connect_attrs Performance Schema tables. See Section 20.9.9, “Performance Schema Connection Attribute Tables”.

- MYSQL_OPT_CONNECT_ATTR_RESET (argument not used)

  This option resets (clears) the current set of connection attributes to be passed to the server at connect time.

  See also the description for the MYSQL_OPT_CONNECT_ATTR_DELETE option, as well as the description for the MYSQL_OPT_CONNECT_ATTR_ADD option in the description of the mysql_options4() function. That function description also includes a usage example.

  Connection attributes are exposed through the session_connect_attrs and session_account_connect_attrs Performance Schema tables. See Section 20.9.9, “Performance Schema Connection Attribute Tables”.

- MYSQL_OPT_CONNECT_TIMEOUT (argument type: unsigned int *)

  Connect timeout in seconds.

- MYSQL_OPT_GUESS_CONNECTION (argument: not used)

  For an application linked against the libmysqld embedded server library, this enables the library to guess whether to use the embedded server or a remote server. “Guess” means that if the host name is set and is not localhost, it uses a remote server. This behavior is the default. MYSQL_OPT_USE_EMBEDDED_CONNECTION and MYSQL_OPT_USE_REMOTE_CONNECTION can be used to override it. This option is ignored for applications linked against the libmysqlclient client library.

- MYSQL_OPT_LOCAL_INFILE (argument type: optional pointer to unsigned int)

  If no pointer is given or if pointer points to an unsigned int that has a nonzero value, the LOAD LOCAL INFILE statement is enabled.

- MYSQL_OPT_NAMED_PIPE (argument: not used)

  Use named pipes to connect to a MySQL server on Windows, if the server permits named-pipe connections.

- MYSQL_OPT_PROTOCOL (argument type: unsigned int *)

  Type of protocol to use. Specify one of the enum values of mysql_protocol_type defined in mysql.h.

- MYSQL_OPT_READ_TIMEOUT (argument type: unsigned int *)

  The timeout in seconds for each attempt to read from the server. There are retries if necessary, so the total effective timeout value is three times the option value. You can set the value so that a lost connection can be detected earlier than the TCP/IP Close_Wait_Timeout value of 10 minutes.

- MYSQL_OPT_RECONNECT (argument type: my_bool *)

  Enable or disable automatic reconnection to the server if the connection is found to have been lost. Reconnect is off by default; this option provides a way to set reconnection behavior explicitly.

- MYSQL_OPT_SSL_CA (argument type: char *)

  The path to a file that contains a list of trusted SSL CAs.

- MYSQL_OPT_SSL_CAPATH (argument type: char *)

  The path to a directory that contains trusted SSL CA certificates in PEM format.

- MYSQL_OPT_SSL_CERT (argument type: char *)

  The name of the SSL certificate file to use for establishing a secure connection.

- MYSQL_OPT_SSL_CIPHER (argument type: char *)

  A list of permissible ciphers to use for SSL encryption.

- MYSQL_OPT_SSL_CRL (argument type: char *)

  The path to a file containing certificate revocation lists in PEM format.

- MYSQL_OPT_SSL_CRLPATH (argument type: char *)

  The path to a directory that contains files containing certificate revocation lists in PEM format.

- MYSQL_OPT_SSL_ENFORCE (argument type: my_bool *)

  Whether to require the connection to use SSL. If enabled and an encrypted connection cannot be established, the connection attempt fails. This option was added in MySQL 5.7.3.

- MYSQL_OPT_SSL_KEY (argument type: char *)

  The name of the SSL key file to use for establishing a secure connection.

- MYSQL_OPT_SSL_VERIFY_SERVER_CERT (argument type: my_bool *)

  Enable or disable verification of the server's Common Name value in its certificate against the host name used when connecting to the server. The connection is rejected if there is a mismatch. This feature can be used to prevent man-in-the-middle attacks. Verification is disabled by default.

- MYSQL_OPT_USE_EMBEDDED_CONNECTION (argument: not used)

  For an application linked against the libmysqld embedded server library, this forces the use of the embedded server for the connection. This option is ignored for applications linked against the libmysqlclient client library.

- MYSQL_OPT_USE_REMOTE_CONNECTION (argument: not used)

  For an application linked against the libmysqld embedded server library, this forces the use of a remote server for the connection. This option is ignored for applications linked against the libmysqlclient client library.

- MYSQL_OPT_USE_RESULT (argument: not used)

  This option is unused.

- MYSQL_OPT_WRITE_TIMEOUT (argument type: unsigned int *)

  The timeout in seconds for each attempt to write to the server. There is a retry if necessary, so the total effective timeout value is two times the option value.

- MYSQL_PLUGIN_DIR (argument type: char *)

  The directory in which to look for client plugins.

- MYSQL_READ_DEFAULT_FILE (argument type: char *)

  Read options from the named option file instead of from my.cnf.

- MYSQL_READ_DEFAULT_GROUP (argument type: char *)

  Read options from the named group from my.cnf or the file specified with MYSQL_READ_DEFAULT_FILE.

- MYSQL_REPORT_DATA_TRUNCATION (argument type: my_bool *)

  Enable or disable reporting of data truncation errors for prepared statements using the error member of MYSQL_BIND structures. (Default: enabled.)

- MYSQL_SECURE_AUTH (argument type: my_bool *)

  Whether to connect to a server that does not support the password hashing used in MySQL 4.1.1 and later. This option is enabled by default.

- MYSQL_SERVER_PUBLIC_KEY (argument type: char *)

  The path name to a file containing the server RSA public key. The file must be in PEM format. The public key is used for RSA encryption of the client password for connections to the server made using accounts that authenticate with the sha256_password plugin. This option is ignored for client accounts that do not authenticate with that plugin. It is also ignored if password encryption is not needed, as is the case when the client connects to the server using an SSL connection.

The server sends the public key to the client as needed, so it is not necessary to use this option for RSA password encryption to occur. It is more efficient to do so because then the server need not send the key.

For additional discussion regarding use of the sha256_password plugin, including how to get the RSA public key, see Section 6.3.9.4, “The SHA-256 Authentication Plugin”.

- MYSQL_SET_CHARSET_DIR (argument type: char *)

  The path name to the directory that contains character set definition files.

- MYSQL_SET_CHARSET_NAME (argument type: char *)

  The name of the character set to use as the default character set. The argument can be MYSQL_AUTODETECT_CHARSET_NAME to cause the character set to be autodetected based on the operating system setting (see Section 10.1.4, “Connection Character Sets and Collations”).

- MYSQL_SET_CLIENT_IP (argument type: char *)

  For an application linked against the libmysqld embedded server library (when libmysqld is compiled with authentication support), this means that the user is considered to have connected from the specified IP address (specified as a string) for authentication purposes. This option is ignored for applications linked against the libmysqlclient client library.

- MYSQL_SHARED_MEMORY_BASE_NAME (argument type: char *)

  The name of the shared-memory object for communication to the server on Windows, if the server supports shared-memory connections. Specify the same value as the --shared-memory-base-name option used for the mysqld server you want to connect to.

The client group is always read if you use MYSQL_READ_DEFAULT_FILE or MYSQL_READ_DEFAULT_GROUP.

The specified group in the option file may contain the following options.

Option	Description
character-sets-dir=*path*	The directory where character sets are installed.
compress	Use the compressed client/server protocol.
connect-timeout=*seconds*	Connect timeout in seconds. On Linux this timeout is also used for waiting for the first answer from the server.
database=*db_name*	Connect to this database if no database was specified in the connect command.
debug	Debug options.
default-character-set=*charset_name*	The default character set to use.
disable-local-infile	Disable use of LOAD DATA LOCAL.
enable-cleartext-plugin	Enable the mysql_clear_password cleartext authentication plugin.
host=*host_name*	Default host name.

Option	Description
`init-command=stmt`	Statement to execute when connecting to MySQL server. Automatically re-executed if reconnection occurs.
`interactive-timeout=seconds`	Same as specifying `CLIENT_INTERACTIVE` to `mysql_real_connect()`. See Section 21.8.7.54, "`mysql_real_connect()`".
`local-infile[={0|1}]`	If no argument or nonzero argument, enable use of `LOAD DATA LOCAL`; otherwise disable.
`max_allowed_packet=bytes`	Maximum size of packet that client can read from server.
`multi-queries`, `multi-results`	Enable multiple result sets from multiple-statement executions or stored procedures.
`multi-statements`	Enable the client to send multiple statements in a single string (separated by "`;`").
`password=password`	Default password.
`pipe`	Use named pipes to connect to a MySQL server on Windows.
`port=port_num`	Default port number.
`protocol={TCP|SOCKET|PIPE|MEMORY}`	The protocol to use when connecting to the server.
`return-found-rows`	Tell `mysql_info()` to return found rows instead of updated rows when using `UPDATE`.
`shared-memory-base-name=name`	Shared-memory name to use to connect to server.
`socket=path`	Default socket file.
`ssl-ca=file_name`	Certificate Authority file.
`ssl-capath=path`	Certificate Authority directory.
`ssl-cert=file_name`	Certificate file.
`ssl-cipher=cipher_list`	Permissible SSL ciphers.
`ssl-key=file_name`	Key file.
`timeout=seconds`	Like `connect-timeout`.
`user`	Default user.

`timeout` has been replaced by `connect-timeout`, but `timeout` is still supported in MySQL 5.7 for backward compatibility.

For more information about option files, see Section 4.2.3.3, "Using Option Files".

Return Values

Zero for success. Nonzero if you specify an unknown option.

Example

The following `mysql_options()` calls request the use of compression in the client/server protocol, cause options to be read from the `[odbc]` group of option files, and disable transaction autocommit mode:

```
MYSQL mysql;
```

```
mysql_init(&mysql);
mysql_options(&mysql,MYSQL_OPT_COMPRESS,0);
mysql_options(&mysql,MYSQL_READ_DEFAULT_GROUP,"odbc");
mysql_options(&mysql,MYSQL_INIT_COMMAND,"SET autocommit=0");
if (!mysql_real_connect(&mysql,"host","user","passwd","database",0,NULL,0))
{
    fprintf(stderr, "Failed to connect to database: Error: %s\n",
          mysql_error(&mysql));
}
```

This code requests that the client use the compressed client/server protocol and read the additional options from the odbc section in the my.cnf file.

21.8.7.51 mysql_options4()

int mysql_options4(MYSQL *mysql, enum mysql_option option, const void *arg1, const void *arg2)

Description

mysql_options4() is similar to mysql_options() but has an extra fourth argument so that two values can be passed for the option specified in the second argument.

The following list describes the permitted options, their effect, and how arg1 and arg2 are used.

- MYSQL_OPT_CONNECT_ATTR_ADD (argument types: char *, char *)

  This option adds a key/value pair to the current set of connection attributes to be passed to the server at connect time. Both arguments are pointers to null-terminated strings. The first and second strings indicate the key and value, respectively. If the key already exists in the current set of connection attributes, an error occurs. Comparison of the key name with existing keys is case sensitive.

  Key names that begin with an underscore (_) are reserved for internal use and should not be used by application programs.

  See also the descriptions for the MYSQL_OPT_CONNECT_ATTR_RESET MYSQL_OPT_CONNECT_ATTR_DELETE options in the description of the mysql_options() function.

  Connection attributes are exposed through the session_connect_attrs and session_account_connect_attrs Performance Schema tables. See Section 20.9.9, "Performance Schema Connection Attribute Tables".

Return Values

Zero for success. Nonzero if you specify an unknown option.

Example

This example demonstrates the calls that specify connection attributes:

```
MYSQL mysql;

mysql_init(&mysql);
mysql_options(&mysql,MYSQL_OPT_CONNECT_ATTR_RESET, 0);
mysql_options4(&mysql,MYSQL_OPT_CONNECT_ATTR_ADD, "key1", "value1");
mysql_options4(&mysql,MYSQL_OPT_CONNECT_ATTR_ADD, "key2", "value2");
mysql_options4(&mysql,MYSQL_OPT_CONNECT_ATTR_ADD, "key3", "value3");
mysql_options(&mysql,MYSQL_OPT_CONNECT_ATTR_DELETE, "key1");
if (!mysql_real_connect(&mysql,"host","user","passwd","database",0,NULL,0))
```

```
{
    fprintf(stderr, "Failed to connect to database: Error: %s\n",
        mysql_error(&mysql));
}
mysql_options(&mysql,MYSQL_OPT_CONNECT_ATTR_RESET, 0);
```

21.8.7.52 `mysql_ping()`

```
int mysql_ping(MYSQL *mysql)
```

Description

Checks whether the connection to the server is working. If the connection has gone down and auto-reconnect is enabled an attempt to reconnect is made. If the connection is down and auto-reconnect is disabled, `mysql_ping()` returns an error.

Auto-reconnect is disabled by default. To enable it, call `mysql_options()` with the `MYSQL_OPT_RECONNECT` option. For details, see Section 21.8.7.50, “`mysql_options()`”.

`mysql_ping()` can be used by clients that remain idle for a long while, to check whether the server has closed the connection and reconnect if necessary.

If `mysql_ping()`) does cause a reconnect, there is no explicit indication of it. To determine whether a reconnect occurs, call `mysql_thread_id()` to get the original connection identifier before calling `mysql_ping()`, then call `mysql_thread_id()` again to see whether the identifier has changed.

If reconnect occurs, some characteristics of the connection will have been reset. For details about these characteristics, see Section 21.8.16, “Controlling Automatic Reconnection Behavior”.

Return Values

Zero if the connection to the server is active. Nonzero if an error occurred. A nonzero return does not indicate whether the MySQL server itself is down; the connection might be broken for other reasons such as network problems.

Errors

- `CR_COMMANDS_OUT_OF_SYNC`

  Commands were executed in an improper order.

- `CR_SERVER_GONE_ERROR`

  The MySQL server has gone away.

- `CR_UNKNOWN_ERROR`

  An unknown error occurred.

21.8.7.53 `mysql_query()`

```
int mysql_query(MYSQL *mysql, const char *stmt_str)
```

Description

Executes the SQL statement pointed to by the null-terminated string `stmt_str`. Normally, the string must consist of a single SQL statement without a terminating semicolon (“`;`”) or `\g`. If multiple-statement execution has been enabled, the string can contain several statements separated by semicolons. See Section 21.8.17, “C API Support for Multiple Statement Execution”.

mysql_query() cannot be used for statements that contain binary data; you must use mysql_real_query() instead. (Binary data may contain the “\0” character, which mysql_query() interprets as the end of the statement string.)

If you want to know whether the statement returns a result set, you can use mysql_field_count() to check for this. See Section 21.8.7.22, “mysql_field_count()”.

Return Values

Zero for success. Nonzero if an error occurred.

Errors

- CR_COMMANDS_OUT_OF_SYNC

  Commands were executed in an improper order.
- CR_SERVER_GONE_ERROR

  The MySQL server has gone away.
- CR_SERVER_LOST

  The connection to the server was lost during the query.
- CR_UNKNOWN_ERROR

  An unknown error occurred.

21.8.7.54 mysql_real_connect()

```
MYSQL *mysql_real_connect(MYSQL *mysql, const char *host, const char *user,
const char *passwd, const char *db, unsigned int port, const char *unix_socket,
unsigned long client_flag)
```

Description

mysql_real_connect() attempts to establish a connection to a MySQL database engine running on host. mysql_real_connect() must complete successfully before you can execute any other API functions that require a valid MYSQL connection handle structure.

The parameters are specified as follows:

- For the first parameter, specify the address of an existing MYSQL structure. Before calling mysql_real_connect(), call mysql_init() to initialize the MYSQL structure. You can change a lot of connect options with the mysql_options() call. See Section 21.8.7.50, “mysql_options()”.
- The value of host may be either a host name or an IP address. If host is NULL or the string "localhost", a connection to the local host is assumed. For Windows, the client connects using a shared-memory connection, if the server has shared-memory connections enabled. Otherwise, TCP/IP is used. For Unix, the client connects using a Unix socket file. For local connections, you can also influence the type of connection to use with the MYSQL_OPT_PROTOCOL or MYSQL_OPT_NAMED_PIPE options to mysql_options(). The type of connection must be supported by the server. For a host value of "."

on Windows, the client connects using a named pipe, if the server has named-pipe connections enabled. If named-pipe connections are not enabled, an error occurs.

- The user parameter contains the user's MySQL login ID. If user is NULL or the empty string "", the current user is assumed. Under Unix, this is the current login name. Under Windows ODBC, the current user name must be specified explicitly. See the Connector/ODBC section of Chapter 21, *Connectors and APIs*.
- The passwd parameter contains the password for user. If passwd is NULL, only entries in the user table for the user that have a blank (empty) password field are checked for a match. This enables the database administrator to set up the MySQL privilege system in such a way that users get different privileges depending on whether they have specified a password.

Note

Do not attempt to encrypt the password before calling mysql_real_connect(); password encryption is handled automatically by the client API.

- The user and passwd parameters use whatever character set has been configured for the MYSQL object. By default, this is latin1, but can be changed by calling mysql_options(mysql, MYSQL_SET_CHARSET_NAME, "*charset_name*") prior to connecting.
- db is the database name. If db is not NULL, the connection sets the default database to this value.
- If port is not 0, the value is used as the port number for the TCP/IP connection. Note that the host parameter determines the type of the connection.
- If unix_socket is not NULL, the string specifies the socket or named pipe to use. Note that the host parameter determines the type of the connection.
- The value of client_flag is usually 0, but can be set to a combination of the following flags to enable certain features.

Flag Name	Flag Description
CAN_HANDLE_EXPIRED_PASSWORDS	The client can handle expired passwords. For more information, see Section 6.3.7, “Password Expiration and Sandbox Mode”.
CLIENT_COMPRESS	Use compression protocol.
CLIENT_FOUND_ROWS	Return the number of found (matched) rows, not the number of changed rows.
CLIENT_IGNORE_SIGPIPE	Prevents the client library from installing a SIGPIPE signal handler. This can be used to avoid conflicts with a handler that the application has already installed.
CLIENT_IGNORE_SPACE	Permit spaces after function names. Makes all functions names reserved words.
CLIENT_INTERACTIVE	Permit interactive_timeout seconds (instead of wait_timeout seconds) of inactivity before closing the connection. The client's session wait_timeout variable is set to the value of the session interactive_timeout variable.
CLIENT_LOCAL_FILES	Enable LOAD DATA LOCAL handling.
CLIENT_MULTI_RESULTS	Tell the server that the client can handle multiple result sets from multiple-statement executions or stored procedures. This flag is automatically enabled if CLIENT_MULTI_STATEMENTS is enabled.

Flag Name	Flag Description
	See the note following this table for more information about this flag.
CLIENT_MULTI_STATEMENTS	Tell the server that the client may send multiple statements in a single string (separated by “;”). If this flag is not set, multiple-statement execution is disabled. See the note following this table for more information about this flag.
CLIENT_NO_SCHEMA	Do not permit the *db_name.tbl_name.col_name* syntax. This is for ODBC. It causes the parser to generate an error if you use that syntax, which is useful for trapping bugs in some ODBC programs.
CLIENT_ODBC	Unused.
CLIENT_SSL	Use SSL (encrypted protocol). Do not set this option within an application program; it is set internally in the client library. Instead, use mysql_ssl_set() before calling mysql_real_connect().
CLIENT_REMEMBER_OPTIONS	Remember options specified by calls to mysql_options(). Without this option, if mysql_real_connect() fails, you must repeat the mysql_options() calls before trying to connect again. With this option, the mysql_options() calls need not be repeated.

If your program uses CALL statements to execute stored procedures, the CLIENT_MULTI_RESULTS flag must be enabled. This is because each CALL returns a result to indicate the call status, in addition to any result sets that might be returned by statements executed within the procedure. Because CALL can return multiple results, process them using a loop that calls mysql_next_result() to determine whether there are more results.

CLIENT_MULTI_RESULTS can be enabled when you call mysql_real_connect(), either explicitly by passing the CLIENT_MULTI_RESULTS flag itself, or implicitly by passing CLIENT_MULTI_STATEMENTS (which also enables CLIENT_MULTI_RESULTS). In MySQL 5.7, CLIENT_MULTI_RESULTS is enabled by default.

If you enable CLIENT_MULTI_STATEMENTS or CLIENT_MULTI_RESULTS, process the result for every call to mysql_query() or mysql_real_query() by using a loop that calls mysql_next_result() to determine whether there are more results. For an example, see Section 21.8.17, “C API Support for Multiple Statement Execution”.

For some parameters, it is possible to have the value taken from an option file rather than from an explicit value in the mysql_real_connect() call. To do this, call mysql_options() with the MYSQL_READ_DEFAULT_FILE or MYSQL_READ_DEFAULT_GROUP option before calling mysql_real_connect(). Then, in the mysql_real_connect() call, specify the “no-value” value for each parameter to be read from an option file:

- For host, specify a value of NULL or the empty string ("").
- For user, specify a value of NULL or the empty string.
- For passwd, specify a value of NULL. (For the password, a value of the empty string in the mysql_real_connect() call cannot be overridden in an option file, because the empty string indicates explicitly that the MySQL account must have an empty password.)
- For db, specify a value of NULL or the empty string.
- For port, specify a value of 0.

- For unix_socket, specify a value of NULL.

If no value is found in an option file for a parameter, its default value is used as indicated in the descriptions given earlier in this section.

Return Values

A MYSQL* connection handle if the connection was successful, NULL if the connection was unsuccessful. For a successful connection, the return value is the same as the value of the first parameter.

Errors

- CR_CONN_HOST_ERROR

  Failed to connect to the MySQL server.

- CR_CONNECTION_ERROR

  Failed to connect to the local MySQL server.

- CR_IPSOCK_ERROR

  Failed to create an IP socket.

- CR_OUT_OF_MEMORY

  Out of memory.

- CR_SOCKET_CREATE_ERROR

  Failed to create a Unix socket.

- CR_UNKNOWN_HOST

  Failed to find the IP address for the host name.

- CR_VERSION_ERROR

  A protocol mismatch resulted from attempting to connect to a server with a client library that uses a different protocol version.

- CR_NAMEDPIPEOPEN_ERROR

  Failed to create a named pipe on Windows.

- CR_NAMEDPIPEWAIT_ERROR

  Failed to wait for a named pipe on Windows.

- CR_NAMEDPIPESETSTATE_ERROR

  Failed to get a pipe handler on Windows.

- CR_SERVER_LOST

  If connect_timeout > 0 and it took longer than connect_timeout seconds to connect to the server or if the server died while executing the init-command.

- CR_ALREADY_CONNECTED

The MYSQL connection handle is already connected.

Example

```
MYSQL mysql;

mysql_init(&mysql);
mysql_options(&mysql,MYSQL_READ_DEFAULT_GROUP,"your_prog_name");
if (!mysql_real_connect(&mysql,"host","user","passwd","database",0,NULL,0))
{
    fprintf(stderr, "Failed to connect to database: Error: %s\n",
          mysql_error(&mysql));
}
```

By using mysql_options() the MySQL library reads the [client] and [your_prog_name] sections in the my.cnf file which ensures that your program works, even if someone has set up MySQL in some nonstandard way.

Note that upon connection, mysql_real_connect() sets the reconnect flag (part of the MYSQL structure) to a value of 1 in versions of the API older than 5.0.3, or 0 in newer versions. A value of 1 for this flag indicates that if a statement cannot be performed because of a lost connection, to try reconnecting to the server before giving up. You can use the MYSQL_OPT_RECONNECT option to mysql_options() to control reconnection behavior.

21.8.7.55 mysql_real_escape_string()

unsigned long mysql_real_escape_string(MYSQL *mysql, char *to, const char *from, unsigned long length)

Note that mysql must be a valid, open connection. This is needed because the escaping depends on the character set in use by the server.

Description

This function is used to create a legal SQL string that you can use in an SQL statement. See Section 9.1.1, “String Literals”.

The string in from is encoded to an escaped SQL string, taking into account the current character set of the connection. The result is placed in to and a terminating null byte is appended. Characters encoded are “\”, “'”, “"”, NUL (ASCII 0), “\n”, “\r”, and Control+Z. Strictly speaking, MySQL requires only that backslash and the quote character used to quote the string in the query be escaped. mysql_real_escape_string() quotes the other characters to make them easier to read in log files. For comparison, see the quoting rules for literal strings and the QUOTE() SQL function in Section 9.1.1, “String Literals”, and Section 12.5, “String Functions”.

The string pointed to by from must be length bytes long. You must allocate the to buffer to be at least length*2+1 bytes long. (In the worst case, each character may need to be encoded as using two bytes, and you need room for the terminating null byte.) When mysql_real_escape_string() returns, the contents of to is a null-terminated string. The return value is the length of the encoded string, not including the terminating null character.

If you need to change the character set of the connection, use the mysql_set_character_set() function rather than executing a SET NAMES (or SET CHARACTER SET) statement. mysql_set_character_set() works like SET NAMES but also affects the character set used by mysql_real_escape_string(), which SET NAMES does not.

Example

```
char query[1000],*end;

end = strmov(query,"INSERT INTO test_table values(");
*end++ = '\'';
end += mysql_real_escape_string(&mysql, end,"What is this",12);
*end++ = '\'';
*end++ = ',';
*end++ = '\'';
end += mysql_real_escape_string(&mysql, end,"binary data: \0\r\n",16);
*end++ = '\'';
*end++ = ')';

if (mysql_real_query(&mysql,query,(unsigned int) (end - query)))
{
   fprintf(stderr, "Failed to insert row, Error: %s\n",
           mysql_error(&mysql));
}
```

The `strmov()` function used in the example is included in the `libmysqlclient` library and works like `strcpy()` but returns a pointer to the terminating null of the first parameter.

Return Values

The length of the value placed into `to`, not including the terminating null character.

Errors

None.

21.8.7.56 `mysql_real_query()`

`int mysql_real_query(MYSQL *mysql, const char *stmt_str, unsigned long length)`

Description

Executes the SQL statement pointed to by `stmt_str`, a string `length` bytes long. Normally, the string must consist of a single SQL statement without a terminating semicolon (“`;`”) or `\g`. If multiple-statement execution has been enabled, the string can contain several statements separated by semicolons. See Section 21.8.17, “C API Support for Multiple Statement Execution”.

`mysql_query()` cannot be used for statements that contain binary data; you must use `mysql_real_query()` instead. (Binary data may contain the “`\0`” character, which `mysql_query()` interprets as the end of the statement string.) In addition, `mysql_real_query()` is faster than `mysql_query()` because it does not call `strlen()` on the statement string.

If you want to know whether the statement returns a result set, you can use `mysql_field_count()` to check for this. See Section 21.8.7.22, “`mysql_field_count()`”.

Return Values

Zero for success. Nonzero if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC`

Commands were executed in an improper order.

- CR_SERVER_GONE_ERROR

  The MySQL server has gone away.
- CR_SERVER_LOST

  The connection to the server was lost during the query.
- CR_UNKNOWN_ERROR

  An unknown error occurred.

21.8.7.57 mysql_refresh()

```
int mysql_refresh(MYSQL *mysql, unsigned int options)
```

Description

This function flushes tables or caches, or resets replication server information. The connected user must have the RELOAD privilege.

The options argument is a bit mask composed from any combination of the following values. Multiple values can be OR'ed together to perform multiple operations with a single call.

- REFRESH_GRANT

  Refresh the grant tables, like FLUSH PRIVILEGES.
- REFRESH_LOG

  Flush the logs, like FLUSH LOGS.
- REFRESH_TABLES

  Flush the table cache, like FLUSH TABLES.
- REFRESH_HOSTS

  Flush the host cache, like FLUSH HOSTS.
- REFRESH_STATUS

  Reset status variables, like FLUSH STATUS.
- REFRESH_THREADS

  Flush the thread cache.
- REFRESH_SLAVE

  On a slave replication server, reset the master server information and restart the slave, like RESET SLAVE.
- REFRESH_MASTER

  On a master replication server, remove the binary log files listed in the binary log index and truncate the index file, like RESET MASTER.

Return Values

Zero for success. Nonzero if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC`

  Commands were executed in an improper order.

- `CR_SERVER_GONE_ERROR`

  The MySQL server has gone away.

- `CR_SERVER_LOST`

  The connection to the server was lost during the query.

- `CR_UNKNOWN_ERROR`

  An unknown error occurred.

21.8.7.58 `mysql_reload()`

```
int mysql_reload(MYSQL *mysql)
```

Description

Asks the MySQL server to reload the grant tables. The connected user must have the `RELOAD` privilege.

This function is deprecated. It is preferable to use `mysql_query()` to issue an SQL `FLUSH PRIVILEGES` statement instead.

Return Values

Zero for success. Nonzero if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC`

  Commands were executed in an improper order.

- `CR_SERVER_GONE_ERROR`

  The MySQL server has gone away.

- `CR_SERVER_LOST`

  The connection to the server was lost during the query.

- `CR_UNKNOWN_ERROR`

  An unknown error occurred.

21.8.7.59 `mysql_reset_connection()`

```
int mysql_reset_connection(MYSQL *mysql)
```

Description

Resets the connection to clear the session state. This function was added in MySQL 5.7.3.

`mysql_reset_connection()` has effects similar to `mysql_change_user()` or an auto-reconnect except that the connection is not closed and reopened, and reauthentication is not done. See Section 21.8.7.3, “`mysql_change_user()`”) and see Section 21.8.16, “Controlling Automatic Reconnection Behavior”).

The connection-related state is affected as follows:

- Any active transactions are rolled back and autocommit mode is reset.
- All table locks are released.
- All `TEMPORARY` tables are closed (and dropped).
- Session system variables are reinitialized to the values of the corresponding global system variables, including system variables that are set implicitly by statements such as `SET NAMES`.
- User variable settings are lost.
- Prepared statements are released.
- `HANDLER` variables are closed.
- The value of `LAST_INSERT_ID()` is reset to 0.
- Locks acquired with `GET_LOCK()` are released.

Return Values

Zero for success. Nonzero if an error occurred.

21.8.7.60 `mysql_rollback()`

```
my_bool mysql_rollback(MYSQL *mysql)
```

Description

Rolls back the current transaction.

The action of this function is subject to the value of the `completion_type` system variable. In particular, if the value of `completion_type` is `RELEASE` (or 2), the server performs a release after terminating a transaction and closes the client connection. Call `mysql_close()` from the client program to close the connection from the client side.

Return Values

Zero for success. Nonzero if an error occurred.

Errors

None.

21.8.7.61 `mysql_row_seek()`

```
MYSQL_ROW_OFFSET mysql_row_seek(MYSQL_RES *result, MYSQL_ROW_OFFSET offset)
```

Description

Sets the row cursor to an arbitrary row in a query result set. The `offset` value is a row offset, typically a value returned from `mysql_row_tell()` or from `mysql_row_seek()`. This value is not a row number; to seek to a row within a result set by number, use `mysql_data_seek()` instead.

This function requires that the result set structure contains the entire result of the query, so `mysql_row_seek()` may be used only in conjunction with `mysql_store_result()`, not with `mysql_use_result()`.

Return Values

The previous value of the row cursor. This value may be passed to a subsequent call to `mysql_row_seek()`.

Errors

None.

21.8.7.62 `mysql_row_tell()`

```
MYSQL_ROW_OFFSET mysql_row_tell(MYSQL_RES *result)
```

Description

Returns the current position of the row cursor for the last `mysql_fetch_row()`. This value can be used as an argument to `mysql_row_seek()`.

Use `mysql_row_tell()` only after `mysql_store_result()`, not after `mysql_use_result()`.

Return Values

The current offset of the row cursor.

Errors

None.

21.8.7.63 `mysql_select_db()`

```
int mysql_select_db(MYSQL *mysql, const char *db)
```

Description

Causes the database specified by `db` to become the default (current) database on the connection specified by `mysql`. In subsequent queries, this database is the default for table references that include no explicit database specifier.

`mysql_select_db()` fails unless the connected user can be authenticated as having permission to use the database.

Return Values

Zero for success. Nonzero if an error occurred.

Errors

- CR_COMMANDS_OUT_OF_SYNC

  Commands were executed in an improper order.

- CR_SERVER_GONE_ERROR

  The MySQL server has gone away.

- CR_SERVER_LOST

  The connection to the server was lost during the query.

- CR_UNKNOWN_ERROR

  An unknown error occurred.

21.8.7.64 mysql_session_track_get_first()

```
int mysql_session_track_get_first(MYSQL *mysql, enum enum_session_state_type
type, const char **data, size_t *length)
```

Description

This function fetches the first session state-change information received from the server. It was added in MySQL 5.7.4.

To control notification for changes to session state, use the session_track_state_change, session_track_schema, and session_track_system_variables system variables (see Section 5.1.4, “Server System Variables”).

The function parameters are used as follows. These descriptions also apply to mysql_session_track_get_first(), which takes the same parameters.

- mysql: The connection handle.

- type: The type of information to retrieve. Permitted values for this parameter are the members of the enum_session_state_type enumeration (defined in mysql_com.h):

  ```
  enum enum_session_state_type
  {
    SESSION_TRACK_SYSTEM_VARIABLES, /* Session system variables */
    SESSION_TRACK_SCHEMA,           /* Current schema */
    SESSION_TRACK_STATE_CHANGE      /* track session state changes */
  };
  ```

  To make it easy to loop over all possible types of session information, the SESSION_TRACK_BEGIN and SESSION_TRACK_END macros are defined as the first and last members of the enum_session_state_type enumeration. The example code shown later in this section demonstrates this technique.

- data: The address of a const char * variable. Following a successful call, this variable points to the returned data, which should be considered read only.

- length: The address of a size_t variable. Following a successful call, this variable contains the length of the data pointed to by the data parameter.

Following a successful call, interpret the `data` and `length` values according to the `type` value, as follows:

- `SESSION_TRACK_SCHEMA`: The length is the length of the new default schema name and the data is the name.
- `SESSION_TRACK_SYSTEM_VARIABLES`: When a session system variable changes, two values per variable are returned (in separate calls). For the first, the length is the length of the variable name and the data is the name. For the second, the length is the length of the variable value and the data is the value. Both data values are represented as strings.
- `SESSION_TRACK_STATE_CHANGE`: The length should be 1 and the data is a byte containing a boolean flag that indicates whether session state changes occurred. This flag is represented as an ASCII value, not a binary (for example, `'1'`, not `0x01`).

Return Values

Zero for success. Nonzero if an error occurred.

Errors

None.

Example

The following example shows how to call `mysql_session_track_get_first()` and `mysql_session_track_get_next()` to retrieve and display all available session state-change information following successful execution of a SQL statement string (represented by `stmt_str`).

```
printf("Execute: %s\n", stmt_str);

if (mysql_query(mysql, stmt_str) != 0)
{
  fprintf(stderr, "Error %u: %s\n",
           mysql_errno(mysql), mysql_error(mysql));
  return;
}

MYSQL_RES *result = mysql_store_result(mysql);
if (result) /* there is a result set to fetch */
{
  /* ... process rows here ... */
  printf("Number of rows returned: %lu\n",
          (unsigned long) mysql_num_rows(result));
  mysql_free_result(result);
}
else        /* there is no result set */
{
  if (mysql_field_count(mysql) == 0)
  {
    printf("Number of rows affected: %lu\n",
            (unsigned long) mysql_affected_rows(mysql));
  }
  else      /* an error occurred */
  {
    fprintf(stderr, "Error %u: %s\n",
             mysql_errno(mysql), mysql_error(mysql));
  }
}

/* extract any available session state-change information */
enum enum_session_state_type type;
```

```
for (type = SESSION_TRACK_BEGIN; type <= SESSION_TRACK_END; type++)
{
  const char *data;
  size_t length;

  if (mysql_session_track_get_first(mysql, type, &data, &length) == 0)
  {
    printf("Type=%d:\n", type);
    printf("mysql_session_track_get_first() returns: %*.*s\n",
           (int) length, (int) length, data);

    /* check for more data */
    while (mysql_session_track_get_next(mysql, type, &data, &length) == 0)
    {
      printf("mysql_session_track_get_next() returns: %*.*s\n",
             (int) length, (int) length, data);
    }
  }
}
```

21.8.7.65 `mysql_session_track_get_next()`

`int mysql_session_track_get_next(MYSQL *mysql, enum enum_session_state_type type, const char **data, size_t *length)`

Description

This function fetches session state-change information received from the server, following that retrieved by `mysql_session_track_get_first()`. It was added in MySQL 5.7.4.

Following a successful call to `mysql_session_track_get_first()`, call `mysql_session_track_get_next()` repeatedly until it returns nonzero to indicate no more information is available. The calling sequence for `mysql_session_track_get_next()` is similar to that for `mysql_session_track_get_first()`. For more information and an example that demonstrates both functions, see Section 21.8.7.64, “`mysql_session_track_get_first()`”.

Return Values

Zero for success. Nonzero if an error occurred.

Errors

None.

21.8.7.66 `mysql_set_character_set()`

`int mysql_set_character_set(MYSQL *mysql, const char *csname)`

Description

This function is used to set the default character set for the current connection. The string `csname` specifies a valid character set name. The connection collation becomes the default collation of the character set. This function works like the `SET NAMES` statement, but also sets the value of `mysql->charset`, and thus affects the character set used by `mysql_real_escape_string()`

Return Values

Zero for success. Nonzero if an error occurred.

Example

```
MYSQL mysql;

mysql_init(&mysql);
if (!mysql_real_connect(&mysql,"host","user","passwd","database",0,NULL,0))
{
    fprintf(stderr, "Failed to connect to database: Error: %s\n",
          mysql_error(&mysql));
}

if (!mysql_set_character_set(&mysql, "utf8"))
{
    printf("New client character set: %s\n",
           mysql_character_set_name(&mysql));
}
```

21.8.7.67 mysql_set_local_infile_default()

void mysql_set_local_infile_default(MYSQL *mysql);

Description

Sets the LOAD LOCAL DATA INFILE handler callback functions to the defaults used internally by the C client library. The library calls this function automatically if mysql_set_local_infile_handler() has not been called or does not supply valid functions for each of its callbacks.

Return Values

None.

Errors

None.

21.8.7.68 mysql_set_local_infile_handler()

void mysql_set_local_infile_handler(MYSQL *mysql, int (*local_infile_init)(void **, const char *, void *), int (*local_infile_read)(void *, char *, unsigned int), void (*local_infile_end)(void *), int (*local_infile_error)(void *, char*, unsigned int), void *userdata);

Description

This function installs callbacks to be used during the execution of LOAD DATA LOCAL INFILE statements. It enables application programs to exert control over local (client-side) data file reading. The arguments are the connection handler, a set of pointers to callback functions, and a pointer to a data area that the callbacks can use to share information.

To use mysql_set_local_infile_handler(), you must write the following callback functions:

```
int
local_infile_init(void **ptr, const char *filename, void *userdata);
```

The initialization function. This is called once to do any setup necessary, open the data file, allocate data structures, and so forth. The first void** argument is a pointer to a pointer. You can set the pointer (that is, *ptr) to a value that will be passed to each of the other callbacks (as a void*). The callbacks can use this pointed-to value to maintain state information. The userdata argument is the same value that is passed to mysql_set_local_infile_handler().

Make the initialization function return zero for success, nonzero for an error.

```
int
local_infile_read(void *ptr, char *buf, unsigned int buf_len);
```

The data-reading function. This is called repeatedly to read the data file. buf points to the buffer where the read data is stored, and buf_len is the maximum number of bytes that the callback can read and store in the buffer. (It can read fewer bytes, but should not read more.)

The return value is the number of bytes read, or zero when no more data could be read (this indicates EOF). Return a value less than zero if an error occurs.

```
void
local_infile_end(void *ptr)
```

The termination function. This is called once after local_infile_read() has returned zero (EOF) or an error. Within this function, deallocate any memory allocated by local_infile_init() and perform any other cleanup necessary. It is invoked even if the initialization function returns an error.

```
int
local_infile_error(void *ptr,
                   char *error_msg,
                   unsigned int error_msg_len);
```

The error-handling function. This is called to get a textual error message to return to the user in case any of your other functions returns an error. error_msg points to the buffer into which the message is written, and error_msg_len is the length of the buffer. Write the message as a null-terminated string, at most error_msg_len–1 bytes long.

The return value is the error number.

Typically, the other callbacks store the error message in the data structure pointed to by ptr, so that local_infile_error() can copy the message from there into error_msg.

After calling mysql_set_local_infile_handler() in your C code and passing pointers to your callback functions, you can then issue a LOAD DATA LOCAL INFILE statement (for example, by using mysql_query()). The client library automatically invokes your callbacks. The file name specified in LOAD DATA LOCAL INFILE will be passed as the second parameter to the local_infile_init() callback.

Return Values

None.

Errors

None.

21.8.7.69 mysql_set_server_option()

```
int mysql_set_server_option(MYSQL *mysql, enum enum_mysql_set_option option)
```

Description

Enables or disables an option for the connection. option can have one of the following values.

Option	Description
MYSQL_OPTION_MULTI_STATEMENTS_ON	Enable multiple-statement support
MYSQL_OPTION_MULTI_STATEMENTS_OFF	Disable multiple-statement support

If you enable multiple-statement support, you should retrieve results from calls to mysql_query() or mysql_real_query() by using a loop that calls mysql_next_result() to determine whether there are more results. For an example, see Section 21.8.17, “C API Support for Multiple Statement Execution”.

Enabling multiple-statement support with MYSQL_OPTION_MULTI_STATEMENTS_ON does not have quite the same effect as enabling it by passing the CLIENT_MULTI_STATEMENTS flag to mysql_real_connect(): CLIENT_MULTI_STATEMENTS also enables CLIENT_MULTI_RESULTS. If you are using the CALL SQL statement in your programs, multiple-result support must be enabled; this means that MYSQL_OPTION_MULTI_STATEMENTS_ON by itself is insufficient to permit the use of CALL.

Return Values

Zero for success. Nonzero if an error occurred.

Errors

- CR_COMMANDS_OUT_OF_SYNC

  Commands were executed in an improper order.

- CR_SERVER_GONE_ERROR

  The MySQL server has gone away.

- CR_SERVER_LOST

  The connection to the server was lost during the query.

- ER_UNKNOWN_COM_ERROR

  The server did not support mysql_set_server_option() (which is the case that the server is older than 4.1.1) or the server did not support the option one tried to set.

21.8.7.70 mysql_shutdown()

```
int mysql_shutdown(MYSQL *mysql, enum mysql_enum_shutdown_level shutdown_level)
```

Description

Asks the database server to shut down. The connected user must have the SHUTDOWN privilege. MySQL 5.7 servers support only one type of shutdown; shutdown_level must be equal to SHUTDOWN_DEFAULT. Additional shutdown levels are planned to make it possible to choose the desired level. Dynamically linked executables which have been compiled with older versions of the libmysqlclient headers and call mysql_shutdown() need to be used with the old libmysqlclient dynamic library.

The shutdown process is described in Section 5.1.12, “The Shutdown Process”.

Return Values

Zero for success. Nonzero if an error occurred.

Errors

- CR_COMMANDS_OUT_OF_SYNC

  Commands were executed in an improper order.

- CR_SERVER_GONE_ERROR

  The MySQL server has gone away.

- CR_SERVER_LOST

  The connection to the server was lost during the query.

- CR_UNKNOWN_ERROR

  An unknown error occurred.

21.8.7.71 mysql_sqlstate()

```
const char *mysql_sqlstate(MYSQL *mysql)
```

Description

Returns a null-terminated string containing the SQLSTATE error code for the most recently executed SQL statement. The error code consists of five characters. '00000' means "no error." The values are specified by ANSI SQL and ODBC. For a list of possible values, see Appendix C, *Errors, Error Codes, and Common Problems*.

SQLSTATE values returned by mysql_sqlstate() differ from MySQL-specific error numbers returned by mysql_errno(). For example, the mysql client program displays errors using the following format, where 1146 is the mysql_errno() value and '42S02' is the corresponding mysql_sqlstate() value:

```
shell> SELECT * FROM no_such_table;
ERROR 1146 (42S02): Table 'test.no_such_table' doesn't exist
```

Not all MySQL error numbers are mapped to SQLSTATE error codes. The value 'HY000' (general error) is used for unmapped error numbers.

If you call mysql_sqlstate() after mysql_real_connect() fails, mysql_sqlstate() might not return a useful value. For example, this happens if a host is blocked by the server and the connection is closed without any SQLSTATE value being sent to the client.

Return Values

A null-terminated character string containing the SQLSTATE error code.

See Also

See Section 21.8.7.14, "mysql_errno()", Section 21.8.7.15, "mysql_error()", and Section 21.8.11.27, "mysql_stmt_sqlstate()".

21.8.7.72 mysql_ssl_set()

```
my_bool mysql_ssl_set(MYSQL *mysql, const char *key, const char *cert, const
char *ca, const char *capath, const char *cipher)
```

Description

`mysql_ssl_set()` is used for establishing secure connections using SSL. It must be called before `mysql_real_connect()`.

`mysql_ssl_set()` does nothing unless SSL support is enabled in the client library.

`mysql` is the connection handler returned from `mysql_init()`. The other parameters are specified as follows:

- `key` is the path name to the key file.
- `cert` is the path name to the certificate file.
- `ca` is the path name to the certificate authority file.
- `capath` is the path name to a directory that contains trusted SSL CA certificates in PEM format.
- `cipher` is a list of permissible ciphers to use for SSL encryption.

Any unused SSL parameters may be given as `NULL`.

Return Values

This function always returns `0`. If SSL setup is incorrect, `mysql_real_connect()` returns an error when you attempt to connect.

21.8.7.73 `mysql_stat()`

```
const char *mysql_stat(MYSQL *mysql)
```

Description

Returns a character string containing information similar to that provided by the `mysqladmin status` command. This includes uptime in seconds and the number of running threads, questions, reloads, and open tables.

Return Values

A character string describing the server status. `NULL` if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC`

  Commands were executed in an improper order.

- `CR_SERVER_GONE_ERROR`

  The MySQL server has gone away.

- `CR_SERVER_LOST`

  The connection to the server was lost during the query.

- CR_UNKNOWN_ERROR

  An unknown error occurred.

21.8.7.74 mysql_store_result()

```
MYSQL_RES *mysql_store_result(MYSQL *mysql)
```

Description

After invoking mysql_query() or mysql_real_query(), you must call mysql_store_result() or mysql_use_result() for every statement that successfully produces a result set (SELECT, SHOW, DESCRIBE, EXPLAIN, CHECK TABLE, and so forth). You must also call mysql_free_result() after you are done with the result set.

You need not call mysql_store_result() or mysql_use_result() for other statements, but it does not do any harm or cause any notable performance degradation if you call mysql_store_result() in all cases. You can detect whether the statement has a result set by checking whether mysql_store_result() returns a nonzero value (more about this later).

If you enable multiple-statement support, you should retrieve results from calls to mysql_query() or mysql_real_query() by using a loop that calls mysql_next_result() to determine whether there are more results. For an example, see Section 21.8.17, "C API Support for Multiple Statement Execution".

If you want to know whether a statement should return a result set, you can use mysql_field_count() to check for this. See Section 21.8.7.22, "mysql_field_count()".

mysql_store_result() reads the entire result of a query to the client, allocates a MYSQL_RES structure, and places the result into this structure.

mysql_store_result() returns a null pointer if the statement did not return a result set (for example, if it was an INSERT statement).

mysql_store_result() also returns a null pointer if reading of the result set failed. You can check whether an error occurred by checking whether mysql_error() returns a nonempty string, mysql_errno() returns nonzero, or mysql_field_count() returns zero.

An empty result set is returned if there are no rows returned. (An empty result set differs from a null pointer as a return value.)

After you have called mysql_store_result() and gotten back a result that is not a null pointer, you can call mysql_num_rows() to find out how many rows are in the result set.

You can call mysql_fetch_row() to fetch rows from the result set, or mysql_row_seek() and mysql_row_tell() to obtain or set the current row position within the result set.

See Section 21.8.15.1, "Why mysql_store_result() Sometimes Returns NULL After mysql_query() Returns Success".

Return Values

A MYSQL_RES result structure with the results. NULL (0) if an error occurred.

Errors

mysql_store_result() resets mysql_error() and mysql_errno() if it succeeds.

- `CR_COMMANDS_OUT_OF_SYNC`

  Commands were executed in an improper order.

- `CR_OUT_OF_MEMORY`

  Out of memory.

- `CR_SERVER_GONE_ERROR`

  The MySQL server has gone away.

- `CR_SERVER_LOST`

  The connection to the server was lost during the query.

- `CR_UNKNOWN_ERROR`

  An unknown error occurred.

21.8.7.75 `mysql_thread_id()`

```
unsigned long mysql_thread_id(MYSQL *mysql)
```

Description

Returns the thread ID of the current connection. This value can be used as an argument to `mysql_kill()` to kill the thread.

If the connection is lost and you reconnect with `mysql_ping()`, the thread ID changes. This means you should not get the thread ID and store it for later. You should get it when you need it.

> **Note**
>
> This function does not work correctly if thread IDs become larger than 32 bits, which can occur on some systems. To avoid problems with `mysql_thread_id()`, do not use it. To get the connection ID, execute a `SELECT CONNECTION_ID()` query and retrieve the result.

Return Values

The thread ID of the current connection.

Errors

None.

21.8.7.76 `mysql_use_result()`

```
MYSQL_RES *mysql_use_result(MYSQL *mysql)
```

Description

After invoking `mysql_query()` or `mysql_real_query()`, you must call `mysql_store_result()` or `mysql_use_result()` for every statement that successfully produces a result set (`SELECT`, `SHOW`, `DESCRIBE`, `EXPLAIN`, `CHECK TABLE`, and so forth). You must also call `mysql_free_result()` after you are done with the result set.

mysql_use_result() initiates a result set retrieval but does not actually read the result set into the client like mysql_store_result() does. Instead, each row must be retrieved individually by making calls to mysql_fetch_row(). This reads the result of a query directly from the server without storing it in a temporary table or local buffer, which is somewhat faster and uses much less memory than mysql_store_result(). The client allocates memory only for the current row and a communication buffer that may grow up to max_allowed_packet bytes.

On the other hand, you should not use mysql_use_result() if you are doing a lot of processing for each row on the client side, or if the output is sent to a screen on which the user may type a ^S (stop scroll). This ties up the server and prevent other threads from updating any tables from which the data is being fetched.

When using mysql_use_result(), you must execute mysql_fetch_row() until a NULL value is returned, otherwise, the unfetched rows are returned as part of the result set for your next query. The C API gives the error Commands out of sync; you can't run this command now if you forget to do this!

You may not use mysql_data_seek(), mysql_row_seek(), mysql_row_tell(), mysql_num_rows(), or mysql_affected_rows() with a result returned from mysql_use_result(), nor may you issue other queries until mysql_use_result() has finished. (However, after you have fetched all the rows, mysql_num_rows() accurately returns the number of rows fetched.)

You must call mysql_free_result() once you are done with the result set.

When using the libmysqld embedded server, the memory benefits are essentially lost because memory usage incrementally increases with each row retrieved until mysql_free_result() is called.

Return Values

A MYSQL_RES result structure. NULL if an error occurred.

Errors

mysql_use_result() resets mysql_error() and mysql_errno() if it succeeds.

- CR_COMMANDS_OUT_OF_SYNC

  Commands were executed in an improper order.

- CR_OUT_OF_MEMORY

  Out of memory.

- CR_SERVER_GONE_ERROR

  The MySQL server has gone away.

- CR_SERVER_LOST

  The connection to the server was lost during the query.

- CR_UNKNOWN_ERROR

  An unknown error occurred.

21.8.7.77 mysql_warning_count()

```
unsigned int mysql_warning_count(MYSQL *mysql)
```

Description

Returns the number of errors, warnings, and notes generated during execution of the previous SQL statement.

Return Values

The warning count.

Errors

None.

21.8.8 C API Prepared Statements

The MySQL client/server protocol provides for the use of prepared statements. This capability uses the `MYSQL_STMT` statement handler data structure returned by the `mysql_stmt_init()` initialization function. Prepared execution is an efficient way to execute a statement more than once. The statement is first parsed to prepare it for execution. Then it is executed one or more times at a later time, using the statement handle returned by the initialization function.

Prepared execution is faster than direct execution for statements executed more than once, primarily because the query is parsed only once. In the case of direct execution, the query is parsed every time it is executed. Prepared execution also can provide a reduction of network traffic because for each execution of the prepared statement, it is necessary only to send the data for the parameters.

Prepared statements might not provide a performance increase in some situations. For best results, test your application both with prepared and nonprepared statements and choose whichever yields best performance.

Another advantage of prepared statements is that it uses a binary protocol that makes data transfer between client and server more efficient.

For a list of SQL statements that can be used as prepared statements, see Section 13.5, “SQL Syntax for Prepared Statements”.

Metadata changes to tables or views referred to by prepared statements are detected and cause automatic repreparation of the statement when it is next executed. For more information, see Section 8.9.4, “Caching of Prepared Statements and Stored Programs”.

21.8.9 C API Prepared Statement Data Structures

Prepared statements use several data structures:

- To obtain a statement handle, pass a `MYSQL` connection handler to `mysql_stmt_init()`, which returns a pointer to a `MYSQL_STMT` data structure. This structure is used for further operations with the statement. To specify the statement to prepare, pass the `MYSQL_STMT` pointer and the statement string to `mysql_stmt_prepare()`.

- To provide input parameters for a prepared statement, set up `MYSQL_BIND` structures and pass them to `mysql_stmt_bind_param()`. To receive output column values, set up `MYSQL_BIND` structures and pass them to `mysql_stmt_bind_result()`.

- The `MYSQL_TIME` structure is used to transfer temporal data in both directions.

The following discussion describes the prepared statement data types in detail. For examples that show how to use them, see Section 21.8.11.10, “`mysql_stmt_execute()`”, and Section 21.8.11.11, “`mysql_stmt_fetch()`”.

- `MYSQL_STMT`

  This structure is a handle for a prepared statement. A handle is created by calling `mysql_stmt_init()`, which returns a pointer to a `MYSQL_STMT`. The handle is used for all subsequent operations with the statement until you close it with `mysql_stmt_close()`, at which point the handle becomes invalid.

  The `MYSQL_STMT` structure has no members intended for application use. Applications should not try to copy a `MYSQL_STMT` structure. There is no guarantee that such a copy will be usable.

  Multiple statement handles can be associated with a single connection. The limit on the number of handles depends on the available system resources.

- `MYSQL_BIND`

  This structure is used both for statement input (data values sent to the server) and output (result values returned from the server):

  - For input, use `MYSQL_BIND` structures with `mysql_stmt_bind_param()` to bind parameter data values to buffers for use by `mysql_stmt_execute()`.

  - For output, use `MYSQL_BIND` structures with `mysql_stmt_bind_result()` to bind buffers to result set columns, for use in fetching rows with `mysql_stmt_fetch()`.

  To use a `MYSQL_BIND` structure, zero its contents to initialize it, then set its members appropriately. For example, to declare and initialize an array of three `MYSQL_BIND` structures, use this code:

  ```
  MYSQL_BIND bind[3];
  memset(bind, 0, sizeof(bind));
  ```

  The `MYSQL_BIND` structure contains the following members for use by application programs. For several of the members, the manner of use depends on whether the structure is used for input or output.

  - `enum enum_field_types buffer_type`

    The type of the buffer. This member indicates the data type of the C language variable bound to a statement parameter or result set column. For input, `buffer_type` indicates the type of the variable containing the value to be sent to the server. For output, it indicates the type of the variable into which a value received from the server should be stored. For permissible `buffer_type` values, see Section 21.8.9.1, “C API Prepared Statement Type Codes”.

  - `void *buffer`

    A pointer to the buffer to be used for data transfer. This is the address of a C language variable.

    For input, `buffer` is a pointer to the variable in which you store the data value for a statement parameter. When you call `mysql_stmt_execute()`, MySQL use the value stored in the variable in place of the corresponding parameter marker in the statement (specified with `?` in the statement string).

For output, buffer is a pointer to the variable in which to return a result set column value. When you call mysql_stmt_fetch(), MySQL stores a column value from the current row of the result set in this variable. You can access the value when the call returns.

To minimize the need for MySQL to perform type conversions between C language values on the client side and SQL values on the server side, use C variables that have types similar to those of the corresponding SQL values:

- For numeric data types, buffer should point to a variable of the proper numeric C type. For integer variables (which can be char for single-byte values or an integer type for larger values), you should also indicate whether the variable has the unsigned attribute by setting the is_unsigned member, described later.

- For character (nonbinary) and binary string data types, buffer should point to a character buffer.

- For date and time data types, buffer should point to a MYSQL_TIME structure.

For guidelines about mapping between C types and SQL types and notes about type conversions, see Section 21.8.9.1, "C API Prepared Statement Type Codes", and Section 21.8.9.2, "C API Prepared Statement Type Conversions".

- unsigned long buffer_length

  The actual size of *buffer in bytes. This indicates the maximum amount of data that can be stored in the buffer. For character and binary C data, the buffer_length value specifies the length of *buffer when used with mysql_stmt_bind_param() to specify input values, or the maximum number of output data bytes that can be fetched into the buffer when used with mysql_stmt_bind_result().

- unsigned long *length

  A pointer to an unsigned long variable that indicates the actual number of bytes of data stored in *buffer. length is used for character or binary C data.

  For input parameter data binding, set *length to indicate the actual length of the parameter value stored in *buffer. This is used by mysql_stmt_execute().

  For output value binding, MySQL sets *length when you call mysql_stmt_fetch(). The mysql_stmt_fetch() return value determines how to interpret the length:

  - If the return value is 0, *length indicates the actual length of the parameter value.

  - If the return value is MYSQL_DATA_TRUNCATED, *length indicates the nontruncated length of the parameter value. In this case, the minimum of *length and buffer_length indicates the actual length of the value.

  length is ignored for numeric and temporal data types because the buffer_type value determines the length of the data value.

  If you must determine the length of a returned value before fetching it, see Section 21.8.11.11, "mysql_stmt_fetch()", for some strategies.

- my_bool *is_null

  This member points to a my_bool variable that is true if a value is NULL, false if it is not NULL. For input, set *is_null to true to indicate that you are passing a NULL value as a statement parameter.

    `is_null` is a *pointer* to a boolean scalar, not a boolean scalar, to provide flexibility in how you specify `NULL` values:

    - If your data values are always `NULL`, use `MYSQL_TYPE_NULL` as the `buffer_type` value when you bind the column. The other `MYSQL_BIND` members, including `is_null`, do not matter.

    - If your data values are always `NOT NULL`, set `is_null = (my_bool*) 0`, and set the other members appropriately for the variable you are binding.

    - In all other cases, set the other members appropriately and set `is_null` to the address of a `my_bool` variable. Set that variable's value to true or false appropriately between executions to indicate whether the corresponding data value is `NULL` or `NOT NULL`, respectively.

    For output, when you fetch a row, MySQL sets the value pointed to by `is_null` to true or false according to whether the result set column value returned from the statement is or is not `NULL`.

  - `my_bool is_unsigned`

    This member applies for C variables with data types that can be `unsigned` (`char`, `short int`, `int`, `long long int`). Set `is_unsigned` to true if the variable pointed to by `buffer` is `unsigned` and false otherwise. For example, if you bind a `signed char` variable to `buffer`, specify a type code of `MYSQL_TYPE_TINY` and set `is_unsigned` to false. If you bind an `unsigned char` instead, the type code is the same but `is_unsigned` should be true. (For `char`, it is not defined whether it is signed or unsigned, so it is best to be explicit about signedness by using `signed char` or `unsigned char`.)

    `is_unsigned` applies only to the C language variable on the client side. It indicates nothing about the signedness of the corresponding SQL value on the server side. For example, if you use an `int` variable to supply a value for a `BIGINT UNSIGNED` column, `is_unsigned` should be false because `int` is a signed type. If you use an `unsigned int` variable to supply a value for a `BIGINT` column, `is_unsigned` should be true because `unsigned int` is an unsigned type. MySQL performs the proper conversion between signed and unsigned values in both directions, although a warning occurs if truncation results.

  - `my_bool *error`

    For output, set this member to point to a `my_bool` variable to have truncation information for the parameter stored there after a row fetching operation. When truncation reporting is enabled, `mysql_stmt_fetch()` returns `MYSQL_DATA_TRUNCATED` and `*error` is true in the `MYSQL_BIND` structures for parameters in which truncation occurred. Truncation indicates loss of sign or significant digits, or that a string was too long to fit in a column. Truncation reporting is enabled by default, but can be controlled by calling `mysql_options()` with the `MYSQL_REPORT_DATA_TRUNCATION` option.

- `MYSQL_TIME`

  This structure is used to send and receive `DATE`, `TIME`, `DATETIME`, and `TIMESTAMP` data directly to and from the server. Set the `buffer` member to point to a `MYSQL_TIME` structure, and set the `buffer_type` member of a `MYSQL_BIND` structure to one of the temporal types (`MYSQL_TYPE_TIME`, `MYSQL_TYPE_DATE`, `MYSQL_TYPE_DATETIME`, `MYSQL_TYPE_TIMESTAMP`).

  The `MYSQL_TIME` structure contains the members listed in the following table.

Member	Description
`unsigned int year`	The year

Member	Description
unsigned int month	The month of the year
unsigned int day	The day of the month
unsigned int hour	The hour of the day
unsigned int minute	The minute of the hour
unsigned int second	The second of the minute
my_bool neg	A boolean flag indicating whether the time is negative
unsigned long second_part	The fractional part of the second in microseconds

Only those parts of a MYSQL_TIME structure that apply to a given type of temporal value are used. The year, month, and day elements are used for DATE, DATETIME, and TIMESTAMP values. The hour, minute, and second elements are used for TIME, DATETIME, and TIMESTAMP values. See Section 21.8.19, “C API Prepared Statement Handling of Date and Time Values”.

21.8.9.1 C API Prepared Statement Type Codes

The buffer_type member of MYSQL_BIND structures indicates the data type of the C language variable bound to a statement parameter or result set column. For input, buffer_type indicates the type of the variable containing the value to be sent to the server. For output, it indicates the type of the variable into which a value received from the server should be stored.

The following table shows the permissible values for the buffer_type member of MYSQL_BIND structures for input values sent to the server. The table shows the C variable types that you can use, the corresponding type codes, and the SQL data types for which the supplied value can be used without conversion. Choose the buffer_type value according to the data type of the C language variable that you are binding. For the integer types, you should also set the is_unsigned member to indicate whether the variable is signed or unsigned.

Input Variable C Type	buffer_type Value	SQL Type of Destination Value
signed char	MYSQL_TYPE_TINY	TINYINT
short int	MYSQL_TYPE_SHORT	SMALLINT
int	MYSQL_TYPE_LONG	INT
long long int	MYSQL_TYPE_LONGLONG	BIGINT
float	MYSQL_TYPE_FLOAT	FLOAT
double	MYSQL_TYPE_DOUBLE	DOUBLE
MYSQL_TIME	MYSQL_TYPE_TIME	TIME
MYSQL_TIME	MYSQL_TYPE_DATE	DATE
MYSQL_TIME	MYSQL_TYPE_DATETIME	DATETIME
MYSQL_TIME	MYSQL_TYPE_TIMESTAMP	TIMESTAMP
char[]	MYSQL_TYPE_STRING	TEXT, CHAR, VARCHAR
char[]	MYSQL_TYPE_BLOB	BLOB, BINARY, VARBINARY
	MYSQL_TYPE_NULL	NULL

Use MYSQL_TYPE_NULL as indicated in the description for the is_null member in Section 21.8.9, “C API Prepared Statement Data Structures”.

For input string data, use MYSQL_TYPE_STRING or MYSQL_TYPE_BLOB depending on whether the value is a character (nonbinary) or binary string:

- MYSQL_TYPE_STRING indicates character input string data. The value is assumed to be in the character set indicated by the character_set_client system variable. If the server stores the value into a column with a different character set, it converts the value to that character set.
- MYSQL_TYPE_BLOB indicates binary input string data. The value is treated as having the binary character set. That is, it is treated as a byte string and no conversion occurs.

The following table shows the permissible values for the buffer_type member of MYSQL_BIND structures for output values received from the server. The table shows the SQL types of received values, the corresponding type codes that such values have in result set metadata, and the recommended C language data types to bind to the MYSQL_BIND structure to receive the SQL values without conversion. Choose the buffer_type value according to the data type of the C language variable that you are binding. For the integer types, you should also set the is_unsigned member to indicate whether the variable is signed or unsigned.

SQL Type of Received Value	buffer_type Value	Output Variable C Type
TINYINT	MYSQL_TYPE_TINY	signed char
SMALLINT	MYSQL_TYPE_SHORT	short int
MEDIUMINT	MYSQL_TYPE_INT24	int
INT	MYSQL_TYPE_LONG	int
BIGINT	MYSQL_TYPE_LONGLONG	long long int
FLOAT	MYSQL_TYPE_FLOAT	float
DOUBLE	MYSQL_TYPE_DOUBLE	double
DECIMAL	MYSQL_TYPE_NEWDECIMAL	char[]
YEAR	MYSQL_TYPE_SHORT	short int
TIME	MYSQL_TYPE_TIME	MYSQL_TIME
DATE	MYSQL_TYPE_DATE	MYSQL_TIME
DATETIME	MYSQL_TYPE_DATETIME	MYSQL_TIME
TIMESTAMP	MYSQL_TYPE_TIMESTAMP	MYSQL_TIME
CHAR, BINARY	MYSQL_TYPE_STRING	char[]
VARCHAR, VARBINARY	MYSQL_TYPE_VAR_STRING	char[]
TINYBLOB, TINYTEXT	MYSQL_TYPE_TINY_BLOB	char[]
BLOB, TEXT	MYSQL_TYPE_BLOB	char[]
MEDIUMBLOB, MEDIUMTEXT	MYSQL_TYPE_MEDIUM_BLOB	char[]
LONGBLOB, LONGTEXT	MYSQL_TYPE_LONG_BLOB	char[]
BIT	MYSQL_TYPE_BIT	char[]

21.8.9.2 C API Prepared Statement Type Conversions

Prepared statements transmit data between the client and server using C language variables on the client side that correspond to SQL values on the server side. If there is a mismatch between the C variable type on the client side and the corresponding SQL value type on the server side, MySQL performs implicit type conversions in both directions.

MySQL knows the type code for the SQL value on the server side. The `buffer_type` value in the `MYSQL_BIND` structure indicates the type code of the C variable that holds the value on the client side. The two codes together tell MySQL what conversion must be performed, if any. Here are some examples:

- If you use `MYSQL_TYPE_LONG` with an `int` variable to pass an integer value to the server that is to be stored into a `FLOAT` column, MySQL converts the value to floating-point format before storing it.

- If you fetch an SQL `MEDIUMINT` column value, but specify a `buffer_type` value of `MYSQL_TYPE_LONGLONG` and use a C variable of type `long long int` as the destination buffer, MySQL converts the `MEDIUMINT` value (which requires less than 8 bytes) for storage into the `long long int` (an 8-byte variable).

- If you fetch a numeric column with a value of 255 into a `char[4]` character array and specify a `buffer_type` value of `MYSQL_TYPE_STRING`, the resulting value in the array is a 4-byte string `'255\0'`.

- MySQL returns `DECIMAL` values as the string representation of the original server-side value, which is why the corresponding C type is `char[]`. For example, `12.345` is returned to the client as `'12.345'`. If you specify `MYSQL_TYPE_NEWDECIMAL` and bind a string buffer to the `MYSQL_BIND` structure, `mysql_stmt_fetch()` stores the value in the buffer as a string without conversion. If instead you specify a numeric variable and type code, `mysql_stmt_fetch()` converts the string-format `DECIMAL` value to numeric form.

- For the `MYSQL_TYPE_BIT` type code, `BIT` values are returned into a string buffer, which is why the corresponding C type is `char[]`. The value represents a bit string that requires interpretation on the client side. To return the value as a type that is easier to deal with, you can cause the value to be cast to integer using either of the following types of expressions:

  ```
  SELECT bit_col + 0 FROM t
  SELECT CAST(bit_col AS UNSIGNED) FROM t
  ```

  To retrieve the value, bind an integer variable large enough to hold the value and specify the appropriate corresponding integer type code.

Before binding variables to the `MYSQL_BIND` structures that are to be used for fetching column values, you can check the type codes for each column of the result set. This might be desirable if you want to determine which variable types would be best to use to avoid type conversions. To get the type codes, call `mysql_stmt_result_metadata()` after executing the prepared statement with `mysql_stmt_execute()`. The metadata provides access to the type codes for the result set as described in Section 21.8.11.23, “`mysql_stmt_result_metadata()`”, and Section 21.8.5, “C API Data Structures”.

To determine whether output string values in a result set returned from the server contain binary or nonbinary data, check whether the `charsetnr` value of the result set metadata is 63 (see Section 21.8.5, “C API Data Structures”). If so, the character set is `binary`, which indicates binary rather than nonbinary data. This enables you to distinguish `BINARY` from `CHAR`, `VARBINARY` from `VARCHAR`, and the `BLOB` types from the `TEXT` types.

If you cause the `max_length` member of the `MYSQL_FIELD` column metadata structures to be set (by calling `mysql_stmt_attr_set()`), be aware that the `max_length` values for the result set indicate the lengths of the longest string representation of the result values, not the lengths of the binary representation. That is, `max_length` does not necessarily correspond to the size of the buffers needed to fetch the values with the binary protocol used for prepared statements. Choose the size of the buffers according to the types of the variables into which you fetch the values. For example, a `TINYINT` column containing the value -128 might have a `max_length` value of 4. But the binary representation of any

TINYINT value requires only 1 byte for storage, so you can supply a signed char variable in which to store the value and set is_unsigned to indicate that values are signed.

Metadata changes to tables or views referred to by prepared statements are detected and cause automatic repreparation of the statement when it is next executed. For more information, see Section 8.9.4, “Caching of Prepared Statements and Stored Programs”.

21.8.10 C API Prepared Statement Function Overview

The functions available for prepared statement processing are summarized here and described in greater detail in a later section. See Section 21.8.11, “C API Prepared Statement Function Descriptions”.

Function	Description
mysql_stmt_affected_rows()	Returns the number of rows changed, deleted, or inserted by prepared UPDATE, DELETE, or INSERT statement
mysql_stmt_attr_get()	Gets value of an attribute for a prepared statement
mysql_stmt_attr_set()	Sets an attribute for a prepared statement
mysql_stmt_bind_param()	Associates application data buffers with the parameter markers in a prepared SQL statement
mysql_stmt_bind_result()	Associates application data buffers with columns in a result set
mysql_stmt_close()	Frees memory used by a prepared statement
mysql_stmt_data_seek()	Seeks to an arbitrary row number in a statement result set
mysql_stmt_errno()	Returns the error number for the last statement execution
mysql_stmt_error()	Returns the error message for the last statement execution
mysql_stmt_execute()	Executes a prepared statement
mysql_stmt_fetch()	Fetches the next row of data from a result set and returns data for all bound columns
mysql_stmt_fetch_column()	Fetch data for one column of the current row of a result set
mysql_stmt_field_count()	Returns the number of result columns for the most recent statement
mysql_stmt_free_result()	Free the resources allocated to a statement handle
mysql_stmt_init()	Allocates memory for a MYSQL_STMT structure and initializes it
mysql_stmt_insert_id()	Returns the ID generated for an AUTO_INCREMENT column by a prepared statement
mysql_stmt_next_result()	Returns/initiates the next result in a multiple-result execution
mysql_stmt_num_rows()	Returns the row count from a buffered statement result set
mysql_stmt_param_count()	Returns the number of parameters in a prepared statement
mysql_stmt_param_metadata()	(Return parameter metadata in the form of a result set) Currently, this function does nothing
mysql_stmt_prepare()	Prepares an SQL statement string for execution
mysql_stmt_reset()	Resets the statement buffers in the server
mysql_stmt_result_metadata()	Returns prepared statement metadata in the form of a result set
mysql_stmt_row_seek()	Seeks to a row offset in a statement result set, using value returned from mysql_stmt_row_tell()
mysql_stmt_row_tell()	Returns the statement row cursor position

Function	Description
mysql_stmt_send_long_data()	Sends long data in chunks to server
mysql_stmt_sqlstate()	Returns the SQLSTATE error code for the last statement execution
mysql_stmt_store_result()	Retrieves a complete result set to the client

Call mysql_stmt_init() to create a statement handle, then mysql_stmt_prepare() to prepare the statement string, mysql_stmt_bind_param() to supply the parameter data, and mysql_stmt_execute() to execute the statement. You can repeat the mysql_stmt_execute() by changing parameter values in the respective buffers supplied through mysql_stmt_bind_param().

You can send text or binary data in chunks to server using mysql_stmt_send_long_data(). See Section 21.8.11.26, “mysql_stmt_send_long_data()”.

If the statement is a SELECT or any other statement that produces a result set, mysql_stmt_prepare() also returns the result set metadata information in the form of a MYSQL_RES result set through mysql_stmt_result_metadata().

You can supply the result buffers using mysql_stmt_bind_result(), so that the mysql_stmt_fetch() automatically returns data to these buffers. This is row-by-row fetching.

When statement execution has been completed, close the statement handle using mysql_stmt_close() so that all resources associated with it can be freed.

If you obtained a SELECT statement's result set metadata by calling mysql_stmt_result_metadata(), you should also free the metadata using mysql_free_result().

Execution Steps

To prepare and execute a statement, an application follows these steps:

1. Create a prepared statement handle with mysql_stmt_init(). To prepare the statement on the server, call mysql_stmt_prepare() and pass it a string containing the SQL statement.

2. If the statement will produce a result set, call mysql_stmt_result_metadata() to obtain the result set metadata. This metadata is itself in the form of result set, albeit a separate one from the one that contains the rows returned by the query. The metadata result set indicates how many columns are in the result and contains information about each column.

3. Set the values of any parameters using mysql_stmt_bind_param(). All parameters must be set. Otherwise, statement execution returns an error or produces unexpected results.

4. Call mysql_stmt_execute() to execute the statement.

5. If the statement produces a result set, bind the data buffers to use for retrieving the row values by calling mysql_stmt_bind_result().

6. Fetch the data into the buffers row by row by calling mysql_stmt_fetch() repeatedly until no more rows are found.

7. Repeat steps 3 through 6 as necessary, by changing the parameter values and re-executing the statement.

When mysql_stmt_prepare() is called, the MySQL client/server protocol performs these actions:

- The server parses the statement and sends the okay status back to the client by assigning a statement ID. It also sends total number of parameters, a column count, and its metadata if it is a result set oriented statement. All syntax and semantics of the statement are checked by the server during this call.

- The client uses this statement ID for the further operations, so that the server can identify the statement from among its pool of statements.

When `mysql_stmt_execute()` is called, the MySQL client/server protocol performs these actions:

- The client uses the statement handle and sends the parameter data to the server.
- The server identifies the statement using the ID provided by the client, replaces the parameter markers with the newly supplied data, and executes the statement. If the statement produces a result set, the server sends the data back to the client. Otherwise, it sends an okay status and the number of rows changed, deleted, or inserted.

When `mysql_stmt_fetch()` is called, the MySQL client/server protocol performs these actions:

- The client reads the data from the current row of the result set and places it into the application data buffers by doing the necessary conversions. If the application buffer type is same as that of the field type returned from the server, the conversions are straightforward.

If an error occurs, you can get the statement error number, error message, and SQLSTATE code using `mysql_stmt_errno()`, `mysql_stmt_error()`, and `mysql_stmt_sqlstate()`, respectively.

Prepared Statement Logging

For prepared statements that are executed with the `mysql_stmt_prepare()` and `mysql_stmt_execute()` C API functions, the server writes `Prepare` and `Execute` lines to the general query log so that you can tell when statements are prepared and executed.

Suppose that you prepare and execute a statement as follows:

1. Call `mysql_stmt_prepare()` to prepare the statement string `"SELECT ?"`.
2. Call `mysql_stmt_bind_param()` to bind the value `3` to the parameter in the prepared statement.
3. Call `mysql_stmt_execute()` to execute the prepared statement.

As a result of the preceding calls, the server writes the following lines to the general query log:

```
Prepare  [1] SELECT ?
Execute  [1] SELECT 3
```

Each `Prepare` and `Execute` line in the log is tagged with a `[N]` statement identifier so that you can keep track of which prepared statement is being logged. *N* is a positive integer. If there are multiple prepared statements active simultaneously for the client, *N* may be greater than 1. Each `Execute` lines shows a prepared statement after substitution of data values for `?` parameters.

21.8.11 C API Prepared Statement Function Descriptions

To prepare and execute queries, use the functions described in detail in the following sections.

All functions that operate with a `MYSQL_STMT` structure begin with the prefix `mysql_stmt_`.

To create a `MYSQL_STMT` handle, use the `mysql_stmt_init()` function.

21.8.11.1 `mysql_stmt_affected_rows()`

```
my_ulonglong mysql_stmt_affected_rows(MYSQL_STMT *stmt)
```

Description

mysql_stmt_affected_rows() may be called immediately after executing a statement with mysql_stmt_execute(). It is like mysql_affected_rows() but for prepared statements. For a description of what the affected-rows value returned by this function means, See Section 21.8.7.1, “mysql_affected_rows()”.

Errors

None.

Example

See the Example in Section 21.8.11.10, “mysql_stmt_execute()”.

21.8.11.2 mysql_stmt_attr_get()

```
my_bool mysql_stmt_attr_get(MYSQL_STMT *stmt, enum enum_stmt_attr_type option,
void *arg)
```

Description

Can be used to get the current value for a statement attribute.

The option argument is the option that you want to get; the arg should point to a variable that should contain the option value. If the option is an integer, arg should point to the value of the integer.

See Section 21.8.11.3, “mysql_stmt_attr_set()”, for a list of options and option types.

Return Values

Zero for success. Nonzero if option is unknown.

Errors

None.

21.8.11.3 mysql_stmt_attr_set()

```
my_bool mysql_stmt_attr_set(MYSQL_STMT *stmt, enum enum_stmt_attr_type option,
const void *arg)
```

Description

Can be used to affect behavior for a prepared statement. This function may be called multiple times to set several options.

The option argument is the option that you want to set. The arg argument is the value for the option. arg should point to a variable that is set to the desired attribute value. The variable type is as indicated in the following table.

The following table shows the possible option values.

Option	Argument Type	Function
STMT_ATTR_UPDATE_MAX_LENGTH	my_bool *	If set to 1, causes mysql_stmt_store_result() to update the metadata MYSQL_FIELD->max_length value.

Option	Argument Type	Function
STMT_ATTR_CURSOR_TYPE	unsigned long *	Type of cursor to open for statement when mysql_stmt_execute() is invoked. *arg can be CURSOR_TYPE_NO_CURSOR (the default) or CURSOR_TYPE_READ_ONLY.
STMT_ATTR_PREFETCH_ROWS	unsigned long *	Number of rows to fetch from server at a time when using a cursor. *arg can be in the range from 1 to the maximum value of unsigned long. The default is 1.

If you use the STMT_ATTR_CURSOR_TYPE option with CURSOR_TYPE_READ_ONLY, a cursor is opened for the statement when you invoke mysql_stmt_execute(). If there is already an open cursor from a previous mysql_stmt_execute() call, it closes the cursor before opening a new one. mysql_stmt_reset() also closes any open cursor before preparing the statement for re-execution. mysql_stmt_free_result() closes any open cursor.

If you open a cursor for a prepared statement, mysql_stmt_store_result() is unnecessary, because that function causes the result set to be buffered on the client side.

Return Values

Zero for success. Nonzero if option is unknown.

Errors

None.

Example

The following example opens a cursor for a prepared statement and sets the number of rows to fetch at a time to 5:

```
MYSQL_STMT *stmt;
int rc;
unsigned long type;
unsigned long prefetch_rows = 5;

stmt = mysql_stmt_init(mysql);
type = (unsigned long) CURSOR_TYPE_READ_ONLY;
rc = mysql_stmt_attr_set(stmt, STMT_ATTR_CURSOR_TYPE, (void*) &type);
/* ... check return value ... */
rc = mysql_stmt_attr_set(stmt, STMT_ATTR_PREFETCH_ROWS,
                         (void*) &prefetch_rows);
/* ... check return value ... */
```

21.8.11.4 mysql_stmt_bind_param()

my_bool mysql_stmt_bind_param(MYSQL_STMT *stmt, MYSQL_BIND *bind)

Description

mysql_stmt_bind_param() is used to bind input data for the parameter markers in the SQL statement that was passed to mysql_stmt_prepare(). It uses MYSQL_BIND structures to supply the data. bind is the address of an array of MYSQL_BIND structures. The client library expects the array to contain one element for each ? parameter marker that is present in the query.

Suppose that you prepare the following statement:

```
INSERT INTO mytbl VALUES(?,?,?)
```

When you bind the parameters, the array of MYSQL_BIND structures must contain three elements, and can be declared like this:

```
MYSQL_BIND bind[3];
```

Section 21.8.9, "C API Prepared Statement Data Structures", describes the members of each MYSQL_BIND element and how they should be set to provide input values.

Return Values

Zero for success. Nonzero if an error occurred.

Errors

- CR_UNSUPPORTED_PARAM_TYPE

  The conversion is not supported. Possibly the buffer_type value is invalid or is not one of the supported types.

- CR_OUT_OF_MEMORY

  Out of memory.

- CR_UNKNOWN_ERROR

  An unknown error occurred.

Example

See the Example in Section 21.8.11.10, "mysql_stmt_execute()".

21.8.11.5 mysql_stmt_bind_result()

my_bool mysql_stmt_bind_result(MYSQL_STMT *stmt, MYSQL_BIND *bind)

Description

mysql_stmt_bind_result() is used to associate (that is, bind) output columns in the result set to data buffers and length buffers. When mysql_stmt_fetch() is called to fetch data, the MySQL client/server protocol places the data for the bound columns into the specified buffers.

All columns must be bound to buffers prior to calling mysql_stmt_fetch(). bind is the address of an array of MYSQL_BIND structures. The client library expects the array to contain one element for each column of the result set. If you do not bind columns to MYSQL_BIND structures, mysql_stmt_fetch() simply ignores the data fetch. The buffers should be large enough to hold the data values, because the protocol does not return data values in chunks.

A column can be bound or rebound at any time, even after a result set has been partially retrieved. The new binding takes effect the next time mysql_stmt_fetch() is called. Suppose that an application binds the columns in a result set and calls mysql_stmt_fetch(). The client/server protocol returns data in the bound buffers. Then suppose that the application binds the columns to a different set of buffers. The protocol places data into the newly bound buffers when the next call to mysql_stmt_fetch() occurs.

To bind a column, an application calls `mysql_stmt_bind_result()` and passes the type, address, and length of the output buffer into which the value should be stored. Section 21.8.9, "C API Prepared Statement Data Structures", describes the members of each `MYSQL_BIND` element and how they should be set to receive output values.

Return Values

Zero for success. Nonzero if an error occurred.

Errors

- `CR_UNSUPPORTED_PARAM_TYPE`

  The conversion is not supported. Possibly the `buffer_type` value is invalid or is not one of the supported types.

- `CR_OUT_OF_MEMORY`

  Out of memory.

- `CR_UNKNOWN_ERROR`

  An unknown error occurred.

Example

See the Example in Section 21.8.11.11, "`mysql_stmt_fetch()`".

21.8.11.6 `mysql_stmt_close()`

```
my_bool mysql_stmt_close(MYSQL_STMT *)
```

Description

Closes the prepared statement. `mysql_stmt_close()` also deallocates the statement handle pointed to by `stmt`.

If the current statement has pending or unread results, this function cancels them so that the next query can be executed.

Return Values

Zero for success. Nonzero if an error occurred.

Errors

- `CR_SERVER_GONE_ERROR`

  The MySQL server has gone away.

- `CR_UNKNOWN_ERROR`

  An unknown error occurred.

Example

See the Example in Section 21.8.11.10, "`mysql_stmt_execute()`".

21.8.11.7 `mysql_stmt_data_seek()`

```
void mysql_stmt_data_seek(MYSQL_STMT *stmt, my_ulonglong offset)
```

Description

Seeks to an arbitrary row in a statement result set. The `offset` value is a row number and should be in the range from `0` to `mysql_stmt_num_rows(stmt)-1`.

This function requires that the statement result set structure contains the entire result of the last executed query, so `mysql_stmt_data_seek()` may be used only in conjunction with `mysql_stmt_store_result()`.

Return Values

None.

Errors

None.

21.8.11.8 `mysql_stmt_errno()`

`unsigned int mysql_stmt_errno(MYSQL_STMT *stmt)`

Description

For the statement specified by `stmt`, `mysql_stmt_errno()` returns the error code for the most recently invoked statement API function that can succeed or fail. A return value of zero means that no error occurred. Client error message numbers are listed in the MySQL `errmsg.h` header file. Server error message numbers are listed in `mysqld_error.h`. Errors also are listed at Appendix C, *Errors, Error Codes, and Common Problems*.

Return Values

An error code value. Zero if no error occurred.

Errors

None.

21.8.11.9 `mysql_stmt_error()`

`const char *mysql_stmt_error(MYSQL_STMT *stmt)`

Description

For the statement specified by `stmt`, `mysql_stmt_error()` returns a null-terminated string containing the error message for the most recently invoked statement API function that can succeed or fail. An empty string (`""`) is returned if no error occurred. Either of these two tests can be used to check for an error:

```
if(*mysql_stmt_errno(stmt))
{
  // an error occurred
}

if (mysql_stmt_error(stmt)[0])
{
  // an error occurred
}
```

The language of the client error messages may be changed by recompiling the MySQL client library. Currently, you can choose error messages in several different languages.

Return Values

A character string that describes the error. An empty string if no error occurred.

Errors

None.

21.8.11.10 `mysql_stmt_execute()`

```
int mysql_stmt_execute(MYSQL_STMT *stmt)
```

Description

`mysql_stmt_execute()` executes the prepared query associated with the statement handle. The currently bound parameter marker values are sent to server during this call, and the server replaces the markers with this newly supplied data.

Statement processing following `mysql_stmt_execute()` depends on the type of statement:

- For an `UPDATE`, `DELETE`, or `INSERT`, the number of changed, deleted, or inserted rows can be found by calling `mysql_stmt_affected_rows()`.
- For a statement such as `SELECT` that generates a result set, you must call `mysql_stmt_fetch()` to fetch the data prior to calling any other functions that result in query processing. For more information on how to fetch the results, refer to Section 21.8.11.11, "`mysql_stmt_fetch()`".

  Do not following invocation of `mysql_stmt_execute()` with a call to `mysql_store_result()` or `mysql_use_result()`. Those functions are not intended for processing results from prepared statements.

For statements that generate a result set, you can request that `mysql_stmt_execute()` open a cursor for the statement by calling `mysql_stmt_attr_set()` before executing the statement. If you execute a statement multiple times, `mysql_stmt_execute()` closes any open cursor before opening a new one.

Metadata changes to tables or views referred to by prepared statements are detected and cause automatic repreparation of the statement when it is next executed. For more information, see Section 8.9.4, "Caching of Prepared Statements and Stored Programs".

Return Values

Zero for success. Nonzero if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC`

  Commands were executed in an improper order.
- `CR_OUT_OF_MEMORY`

  Out of memory.
- `CR_SERVER_GONE_ERROR`

  The MySQL server has gone away.
- `CR_SERVER_LOST`

  The connection to the server was lost during the query.

- CR_UNKNOWN_ERROR

  An unknown error occurred.

Example

The following example demonstrates how to create and populate a table using mysql_stmt_init(), mysql_stmt_prepare(), mysql_stmt_param_count(), mysql_stmt_bind_param(), mysql_stmt_execute(), and mysql_stmt_affected_rows(). The mysql variable is assumed to be a valid connection handle. For an example that shows how to retrieve data, see Section 21.8.11.11, “mysql_stmt_fetch()”.

```
#define STRING_SIZE 50

#define DROP_SAMPLE_TABLE "DROP TABLE IF EXISTS test_table"
#define CREATE_SAMPLE_TABLE "CREATE TABLE test_table(col1 INT,\
                                                 col2 VARCHAR(40),\
                                                 col3 SMALLINT,\
                                                 col4 TIMESTAMP)"
#define INSERT_SAMPLE "INSERT INTO \
                       test_table(col1,col2,col3) \
                       VALUES(?,?,?)"

MYSQL_STMT    *stmt;
MYSQL_BIND    bind[3];
my_ulonglong  affected_rows;
int           param_count;
short         small_data;
int           int_data;
char          str_data[STRING_SIZE];
unsigned long str_length;
my_bool       is_null;

if (mysql_query(mysql, DROP_SAMPLE_TABLE))
{
  fprintf(stderr, " DROP TABLE failed\n");
  fprintf(stderr, " %s\n", mysql_error(mysql));
  exit(0);
}

if (mysql_query(mysql, CREATE_SAMPLE_TABLE))
{
  fprintf(stderr, " CREATE TABLE failed\n");
  fprintf(stderr, " %s\n", mysql_error(mysql));
  exit(0);
}

/* Prepare an INSERT query with 3 parameters */
/* (the TIMESTAMP column is not named; the server */
/*  sets it to the current date and time) */
stmt = mysql_stmt_init(mysql);
if (!stmt)
{
  fprintf(stderr, " mysql_stmt_init(), out of memory\n");
  exit(0);
}
if (mysql_stmt_prepare(stmt, INSERT_SAMPLE, strlen(INSERT_SAMPLE)))
{
  fprintf(stderr, " mysql_stmt_prepare(), INSERT failed\n");
  fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
  exit(0);
}
fprintf(stdout, " prepare, INSERT successful\n");
```

```
/* Get the parameter count from the statement */
param_count= mysql_stmt_param_count(stmt);
fprintf(stdout, " total parameters in INSERT: %d\n", param_count);

if (param_count != 3) /* validate parameter count */
{
  fprintf(stderr, " invalid parameter count returned by MySQL\n");
  exit(0);
}

/* Bind the data for all 3 parameters */

memset(bind, 0, sizeof(bind));

/* INTEGER PARAM */
/* This is a number type, so there is no need
   to specify buffer_length */
bind[0].buffer_type= MYSQL_TYPE_LONG;
bind[0].buffer= (char *)&int_data;
bind[0].is_null= 0;
bind[0].length= 0;

/* STRING PARAM */
bind[1].buffer_type= MYSQL_TYPE_STRING;
bind[1].buffer= (char *)str_data;
bind[1].buffer_length= STRING_SIZE;
bind[1].is_null= 0;
bind[1].length= &str_length;

/* SMALLINT PARAM */
bind[2].buffer_type= MYSQL_TYPE_SHORT;
bind[2].buffer= (char *)&small_data;
bind[2].is_null= &is_null;
bind[2].length= 0;

/* Bind the buffers */
if (mysql_stmt_bind_param(stmt, bind))
{
  fprintf(stderr, " mysql_stmt_bind_param() failed\n");
  fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
  exit(0);
}

/* Specify the data values for the first row */
int_data= 10;             /* integer */
strncpy(str_data, "MySQL", STRING_SIZE); /* string  */
str_length= strlen(str_data);

/* INSERT SMALLINT data as NULL */
is_null= 1;

/* Execute the INSERT statement - 1*/
if (mysql_stmt_execute(stmt))
{
  fprintf(stderr, " mysql_stmt_execute(), 1 failed\n");
  fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
  exit(0);
}

/* Get the number of affected rows */
affected_rows= mysql_stmt_affected_rows(stmt);
fprintf(stdout, " total affected rows(insert 1): %lu\n",
                (unsigned long) affected_rows);

if (affected_rows != 1) /* validate affected rows */
{
  fprintf(stderr, " invalid affected rows by MySQL\n");
  exit(0);
}

/* Specify data values for second row,
   then re-execute the statement */
int_data= 1000;
strncpy(str_data, "
        The most popular Open Source database",
        STRING_SIZE);
str_length= strlen(str_data);
small_data= 1000;         /* smallint */
is_null= 0;               /* reset */

/* Execute the INSERT statement - 2*/
if (mysql_stmt_execute(stmt))
{
  fprintf(stderr, " mysql_stmt_execute, 2 failed\n");
  fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
  exit(0);
}

/* Get the total rows affected */
affected_rows= mysql_stmt_affected_rows(stmt);
fprintf(stdout, " total affected rows(insert 2): %lu\n",
                (unsigned long) affected_rows);

if (affected_rows != 1) /* validate affected rows */
{
  fprintf(stderr, " invalid affected rows by MySQL\n");
  exit(0);
}

/* Close the statement */
if (mysql_stmt_close(stmt))
{
  fprintf(stderr, " failed while closing the statement\n");
  fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
  exit(0);
}
```

Note

For complete examples on the use of prepared statement functions, refer to the file `tests/mysql_client_test.c`. This file can be obtained from a MySQL source distribution or from the Bazaar source repository.

21.8.11.11 `mysql_stmt_fetch()`

`int mysql_stmt_fetch(MYSQL_STMT *stmt)`

Description

`mysql_stmt_fetch()` returns the next row in the result set. It can be called only while the result set exists; that is, after a call to `mysql_stmt_execute()` for a statement such as `SELECT` that produces a result set.

`mysql_stmt_fetch()` returns row data using the buffers bound by `mysql_stmt_bind_result()`. It returns the data in those buffers for all the columns in the current row set and the lengths are returned to the `length` pointer. All columns must be bound by the application before it calls `mysql_stmt_fetch()`.

By default, result sets are fetched unbuffered a row at a time from the server. To buffer the entire result set on the client, call `mysql_stmt_store_result()` after binding the data buffers and before calling `mysql_stmt_fetch()`.

If a fetched data value is a NULL value, the *is_null value of the corresponding MYSQL_BIND structure contains TRUE (1). Otherwise, the data and its length are returned in the *buffer and *length elements based on the buffer type specified by the application. Each numeric and temporal type has a fixed length, as listed in the following table. The length of the string types depends on the length of the actual data value, as indicated by data_length.

Type	Length
MYSQL_TYPE_TINY	1
MYSQL_TYPE_SHORT	2
MYSQL_TYPE_LONG	4
MYSQL_TYPE_LONGLONG	8
MYSQL_TYPE_FLOAT	4
MYSQL_TYPE_DOUBLE	8
MYSQL_TYPE_TIME	sizeof(MYSQL_TIME)
MYSQL_TYPE_DATE	sizeof(MYSQL_TIME)
MYSQL_TYPE_DATETIME	sizeof(MYSQL_TIME)
MYSQL_TYPE_STRING	data length
MYSQL_TYPE_BLOB	data_length

In some cases you might want to determine the length of a column value before fetching it with mysql_stmt_fetch(). For example, the value might be a long string or BLOB value for which you want to know how much space must be allocated. To accomplish this, you can use these strategies:

- Before invoking mysql_stmt_fetch() to retrieve individual rows, pass STMT_ATTR_UPDATE_MAX_LENGTH to mysql_stmt_attr_set(), then invoke mysql_stmt_store_result() to buffer the entire result on the client side. Setting the STMT_ATTR_UPDATE_MAX_LENGTH attribute causes the maximal length of column values to be indicated by the max_length member of the result set metadata returned by mysql_stmt_result_metadata().

- Invoke mysql_stmt_fetch() with a zero-length buffer for the column in question and a pointer in which the real length can be stored. Then use the real length with mysql_stmt_fetch_column().

```
real_length= 0;

bind[0].buffer= 0;
bind[0].buffer_length= 0;
bind[0].length= &real_length
mysql_stmt_bind_result(stmt, bind);

mysql_stmt_fetch(stmt);
if (real_length > 0)
{
  data= malloc(real_length);
  bind[0].buffer= data;
  bind[0].buffer_length= real_length;
  mysql_stmt_fetch_column(stmt, bind, 0, 0);
}
```

Return Values

Return Value	Description
0	Successful, the data has been fetched to application data buffers.

Return Value	Description
1	Error occurred. Error code and message can be obtained by calling `mysql_stmt_errno()` and `mysql_stmt_error()`.
`MYSQL_NO_DATA`	No more rows/data exists
`MYSQL_DATA_TRUNCATED`	Data truncation occurred

`MYSQL_DATA_TRUNCATED` is returned when truncation reporting is enabled. To determine which column values were truncated when this value is returned, check the `error` members of the `MYSQL_BIND` structures used for fetching values. Truncation reporting is enabled by default, but can be controlled by calling `mysql_options()` with the `MYSQL_REPORT_DATA_TRUNCATION` option.

Errors

- `CR_COMMANDS_OUT_OF_SYNC`

  Commands were executed in an improper order.

- `CR_OUT_OF_MEMORY`

  Out of memory.

- `CR_SERVER_GONE_ERROR`

  The MySQL server has gone away.

- `CR_SERVER_LOST`

  The connection to the server was lost during the query.

- `CR_UNKNOWN_ERROR`

  An unknown error occurred.

- `CR_UNSUPPORTED_PARAM_TYPE`

  The buffer type is `MYSQL_TYPE_DATE`, `MYSQL_TYPE_TIME`, `MYSQL_TYPE_DATETIME`, or `MYSQL_TYPE_TIMESTAMP`, but the data type is not `DATE`, `TIME`, `DATETIME`, or `TIMESTAMP`.

- All other unsupported conversion errors are returned from `mysql_stmt_bind_result()`.

Example

The following example demonstrates how to fetch data from a table using `mysql_stmt_result_metadata()`, `mysql_stmt_bind_result()`, and `mysql_stmt_fetch()`. (This example expects to retrieve the two rows inserted by the example shown in Section 21.8.11.10, “`mysql_stmt_execute()`”.) The `mysql` variable is assumed to be a valid connection handle.

```
#define STRING_SIZE 50

#define SELECT_SAMPLE "SELECT col1, col2, col3, col4 \
                       FROM test_table"

MYSQL_STMT    *stmt;
MYSQL_BIND    bind[4];
MYSQL_RES     *prepare_meta_result;
MYSQL_TIME    ts;
unsigned long length[4];
int           param_count, column_count, row_count;
```

```
short         small_data;
int           int_data;
char          str_data[STRING_SIZE];
my_bool       is_null[4];
my_bool       error[4];

/* Prepare a SELECT query to fetch data from test_table */
stmt = mysql_stmt_init(mysql);
if (!stmt)
{
  fprintf(stderr, " mysql_stmt_init(), out of memory\n");
  exit(0);
}
if (mysql_stmt_prepare(stmt, SELECT_SAMPLE, strlen(SELECT_SAMPLE)))
{
  fprintf(stderr, " mysql_stmt_prepare(), SELECT failed\n");
  fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
  exit(0);
}
fprintf(stdout, " prepare, SELECT successful\n");

/* Get the parameter count from the statement */
param_count= mysql_stmt_param_count(stmt);
fprintf(stdout, " total parameters in SELECT: %d\n", param_count);

if (param_count != 0) /* validate parameter count */
{
  fprintf(stderr, " invalid parameter count returned by MySQL\n");
  exit(0);
}

/* Fetch result set meta information */
prepare_meta_result = mysql_stmt_result_metadata(stmt);
if (!prepare_meta_result)
{
  fprintf(stderr,
         " mysql_stmt_result_metadata(), \
           returned no meta information\n");
  fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
  exit(0);
}

/* Get total columns in the query */
column_count= mysql_num_fields(prepare_meta_result);
fprintf(stdout,
        " total columns in SELECT statement: %d\n",
        column_count);

if (column_count != 4) /* validate column count */
{
  fprintf(stderr, " invalid column count returned by MySQL\n");
  exit(0);
}

/* Execute the SELECT query */
if (mysql_stmt_execute(stmt))
{
  fprintf(stderr, " mysql_stmt_execute(), failed\n");
  fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
  exit(0);
}

/* Bind the result buffers for all 4 columns before fetching them */

memset(bind, 0, sizeof(bind));

/* INTEGER COLUMN */
```

```
bind[0].buffer_type= MYSQL_TYPE_LONG;
bind[0].buffer= (char *)&int_data;
bind[0].is_null= &is_null[0];
bind[0].length= &length[0];
bind[0].error= &error[0];

/* STRING COLUMN */
bind[1].buffer_type= MYSQL_TYPE_STRING;
bind[1].buffer= (char *)str_data;
bind[1].buffer_length= STRING_SIZE;
bind[1].is_null= &is_null[1];
bind[1].length= &length[1];
bind[1].error= &error[1];

/* SMALLINT COLUMN */
bind[2].buffer_type= MYSQL_TYPE_SHORT;
bind[2].buffer= (char *)&small_data;
bind[2].is_null= &is_null[2];
bind[2].length= &length[2];
bind[2].error= &error[2];

/* TIMESTAMP COLUMN */
bind[3].buffer_type= MYSQL_TYPE_TIMESTAMP;
bind[3].buffer= (char *)&ts;
bind[3].is_null= &is_null[3];
bind[3].length= &length[3];
bind[3].error= &error[3];

/* Bind the result buffers */
if (mysql_stmt_bind_result(stmt, bind))
{
  fprintf(stderr, " mysql_stmt_bind_result() failed\n");
  fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
  exit(0);
}

/* Now buffer all results to client (optional step) */
if (mysql_stmt_store_result(stmt))
{
  fprintf(stderr, " mysql_stmt_store_result() failed\n");
  fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
  exit(0);
}

/* Fetch all rows */
row_count= 0;
fprintf(stdout, "Fetching results ...\n");
while (!mysql_stmt_fetch(stmt))
{
  row_count++;
  fprintf(stdout, "  row %d\n", row_count);

  /* column 1 */
  fprintf(stdout, "   column1 (integer)  : ");
  if (is_null[0])
    fprintf(stdout, " NULL\n");
  else
    fprintf(stdout, " %d(%ld)\n", int_data, length[0]);

  /* column 2 */
  fprintf(stdout, "   column2 (string)   : ");
  if (is_null[1])
    fprintf(stdout, " NULL\n");
  else
    fprintf(stdout, " %s(%ld)\n", str_data, length[1]);

  /* column 3 */
  fprintf(stdout, "   column3 (smallint) : ");
  if (is_null[2])
    fprintf(stdout, " NULL\n");
  else
    fprintf(stdout, " %d(%ld)\n", small_data, length[2]);

  /* column 4 */
  fprintf(stdout, "   column4 (timestamp): ");
  if (is_null[3])
    fprintf(stdout, " NULL\n");
  else
    fprintf(stdout, " %04d-%02d-%02d %02d:%02d:%02d (%ld)\n",
                     ts.year, ts.month, ts.day,
                     ts.hour, ts.minute, ts.second,
                     length[3]);
  fprintf(stdout, "\n");
}

/* Validate rows fetched */
fprintf(stdout, " total rows fetched: %d\n", row_count);
if (row_count != 2)
{
  fprintf(stderr, " MySQL failed to return all rows\n");
  exit(0);
}

/* Free the prepared result metadata */
mysql_free_result(prepare_meta_result);


/* Close the statement */
if (mysql_stmt_close(stmt))
{
  fprintf(stderr, " failed while closing the statement\n");
  fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
  exit(0);
}
```

21.8.11.12 mysql_stmt_fetch_column()

```
int mysql_stmt_fetch_column(MYSQL_STMT *stmt, MYSQL_BIND *bind, unsigned int
column, unsigned long offset)
```

Description

Fetch one column from the current result set row. bind provides the buffer where data should be placed. It should be set up the same way as for mysql_stmt_bind_result(). column indicates which column to fetch. The first column is numbered 0. offset is the offset within the data value at which to begin retrieving data. This can be used for fetching the data value in pieces. The beginning of the value is offset 0.

Return Values

Zero for success. Nonzero if an error occurred.

Errors

- CR_INVALID_PARAMETER_NO

  Invalid column number.

- CR_NO_DATA

The end of the result set has already been reached.

21.8.11.13 mysql_stmt_field_count()

```
unsigned int mysql_stmt_field_count(MYSQL_STMT *stmt)
```

Description

Returns the number of columns for the most recent statement for the statement handler. This value is zero for statements such as INSERT or DELETE that do not produce result sets.

mysql_stmt_field_count() can be called after you have prepared a statement by invoking mysql_stmt_prepare().

Return Values

An unsigned integer representing the number of columns in a result set.

Errors

None.

21.8.11.14 mysql_stmt_free_result()

```
my_bool mysql_stmt_free_result(MYSQL_STMT *stmt)
```

Description

Releases memory associated with the result set produced by execution of the prepared statement. If there is a cursor open for the statement, mysql_stmt_free_result() closes it.

Return Values

Zero for success. Nonzero if an error occurred.

Errors

21.8.11.15 mysql_stmt_init()

```
MYSQL_STMT *mysql_stmt_init(MYSQL *mysql)
```

Description

Create a MYSQL_STMT handle. The handle should be freed with mysql_stmt_close(MYSQL_STMT *).

See also Section 21.8.9, “C API Prepared Statement Data Structures”, for more information.

Return Values

A pointer to a MYSQL_STMT structure in case of success. NULL if out of memory.

Errors

- CR_OUT_OF_MEMORY

  Out of memory.

21.8.11.16 `mysql_stmt_insert_id()`

```
my_ulonglong mysql_stmt_insert_id(MYSQL_STMT *stmt)
```

Description

Returns the value generated for an `AUTO_INCREMENT` column by the prepared `INSERT` or `UPDATE` statement. Use this function after you have executed a prepared `INSERT` statement on a table which contains an `AUTO_INCREMENT` field.

See Section 21.8.7.38, "`mysql_insert_id()`", for more information.

Return Values

Value for `AUTO_INCREMENT` column which was automatically generated or explicitly set during execution of prepared statement, or value generated by `LAST_INSERT_ID(expr)` function. Return value is undefined if statement does not set `AUTO_INCREMENT` value.

Errors

None.

21.8.11.17 `mysql_stmt_next_result()`

```
int mysql_stmt_next_result(MYSQL_STMT *mysql)
```

Description

This function is used when you use prepared `CALL` statements to execute stored procedures, which can return multiple result sets. Use a loop that calls `mysql_stmt_next_result()` to determine whether there are more results. If a procedure has `OUT` or `INOUT` parameters, their values will be returned as a single-row result set following any other result sets. The values will appear in the order in which they are declared in the procedure parameter list.

`mysql_stmt_next_result()` returns a status to indicate whether more results exist. If `mysql_stmt_next_result()` returns an error, there are no more results.

Before each call to `mysql_stmt_next_result()`, you must call `mysql_stmt_free_result()` for the current result if it produced a result set (rather than just a result status).

After calling `mysql_stmt_next_result()` the state of the connection is as if you had called `mysql_stmt_execute()`. This means that you can call `mysql_stmt_bind_result()`, `mysql_stmt_affected_rows()`, and so forth.

It is also possible to test whether there are more results by calling `mysql_more_results()`. However, this function does not change the connection state, so if it returns true, you must still call `mysql_stmt_next_result()` to advance to the next result.

For an example that shows how to use `mysql_stmt_next_result()`, see Section 21.8.20, "C API Support for Prepared `CALL` Statements".

Return Values

Return Value	Description
0	Successful and there are more results
-1	Successful and there are no more results

Return Value	Description
>0	An error occurred

Errors

- `CR_COMMANDS_OUT_OF_SYNC`

  Commands were executed in an improper order.

- `CR_SERVER_GONE_ERROR`

  The MySQL server has gone away.

- `CR_SERVER_LOST`

  The connection to the server was lost during the query.

- `CR_UNKNOWN_ERROR`

  An unknown error occurred.

21.8.11.18 `mysql_stmt_num_rows()`

```
my_ulonglong mysql_stmt_num_rows(MYSQL_STMT *stmt)
```

Description

Returns the number of rows in the result set.

The use of `mysql_stmt_num_rows()` depends on whether you used `mysql_stmt_store_result()` to buffer the entire result set in the statement handle. If you use `mysql_stmt_store_result()`, `mysql_stmt_num_rows()` may be called immediately. Otherwise, the row count is unavailable unless you count the rows as you fetch them.

`mysql_stmt_num_rows()` is intended for use with statements that return a result set, such as `SELECT`. For statements such as `INSERT`, `UPDATE`, or `DELETE`, the number of affected rows can be obtained with `mysql_stmt_affected_rows()`.

Return Values

The number of rows in the result set.

Errors

None.

21.8.11.19 `mysql_stmt_param_count()`

```
unsigned long mysql_stmt_param_count(MYSQL_STMT *stmt)
```

Description

Returns the number of parameter markers present in the prepared statement.

Return Values

An unsigned long integer representing the number of parameters in a statement.

Errors

None.

Example

See the Example in Section 21.8.11.10, “mysql_stmt_execute()”.

21.8.11.20 mysql_stmt_param_metadata()

```
MYSQL_RES *mysql_stmt_param_metadata(MYSQL_STMT *stmt)
```

This function currently does nothing.

Description

Return Values

Errors

21.8.11.21 mysql_stmt_prepare()

```
int mysql_stmt_prepare(MYSQL_STMT *stmt, const char *stmt_str, unsigned long
length)
```

Description

Given the statement handle returned by mysql_stmt_init(), prepares the SQL statement pointed to by the string stmt_str and returns a status value. The string length should be given by the length argument. The string must consist of a single SQL statement. You should not add a terminating semicolon (“;”) or \g to the statement.

The application can include one or more parameter markers in the SQL statement by embedding question mark (?) characters into the SQL string at the appropriate positions.

The markers are legal only in certain places in SQL statements. For example, they are permitted in the VALUES() list of an INSERT statement (to specify column values for a row), or in a comparison with a column in a WHERE clause to specify a comparison value. However, they are not permitted for identifiers (such as table or column names), or to specify both operands of a binary operator such as the = equal sign. The latter restriction is necessary because it would be impossible to determine the parameter type. In general, parameters are legal only in Data Manipulation Language (DML) statements, and not in Data Definition Language (DDL) statements.

The parameter markers must be bound to application variables using mysql_stmt_bind_param() before executing the statement.

Metadata changes to tables or views referred to by prepared statements are detected and cause automatic repreparation of the statement when it is next executed. For more information, see Section 8.9.4, “Caching of Prepared Statements and Stored Programs”.

Return Values

Zero for success. Nonzero if an error occurred.

Errors

- CR_COMMANDS_OUT_OF_SYNC

Commands were executed in an improper order.

- `CR_OUT_OF_MEMORY`

  Out of memory.

- `CR_SERVER_GONE_ERROR`

  The MySQL server has gone away.

- `CR_SERVER_LOST`

  The connection to the server was lost during the query

- `CR_UNKNOWN_ERROR`

  An unknown error occurred.

If the prepare operation was unsuccessful (that is, `mysql_stmt_prepare()` returns nonzero), the error message can be obtained by calling `mysql_stmt_error()`.

Example

See the Example in Section 21.8.11.10, "`mysql_stmt_execute()`".

21.8.11.22 `mysql_stmt_reset()`

```
my_bool mysql_stmt_reset(MYSQL_STMT *stmt)
```

Description

Resets a prepared statement on client and server to state after prepare. It resets the statement on the server, data sent using `mysql_stmt_send_long_data()`, unbuffered result sets and current errors. It does not clear bindings or stored result sets. Stored result sets will be cleared when executing the prepared statement (or closing it).

To re-prepare the statement with another query, use `mysql_stmt_prepare()`.

Return Values

Zero for success. Nonzero if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC`

  Commands were executed in an improper order.

- `CR_SERVER_GONE_ERROR`

  The MySQL server has gone away.

- `CR_SERVER_LOST`

  The connection to the server was lost during the query

- `CR_UNKNOWN_ERROR`

An unknown error occurred.

21.8.11.23 mysql_stmt_result_metadata()

```
MYSQL_RES *mysql_stmt_result_metadata(MYSQL_STMT *stmt)
```

Description

If a statement passed to mysql_stmt_prepare() is one that produces a result set, mysql_stmt_result_metadata() returns the result set metadata in the form of a pointer to a MYSQL_RES structure that can be used to process the meta information such as number of fields and individual field information. This result set pointer can be passed as an argument to any of the field-based API functions that process result set metadata, such as:

- mysql_num_fields()
- mysql_fetch_field()
- mysql_fetch_field_direct()
- mysql_fetch_fields()
- mysql_field_count()
- mysql_field_seek()
- mysql_field_tell()
- mysql_free_result()

The result set structure should be freed when you are done with it, which you can do by passing it to mysql_free_result(). This is similar to the way you free a result set obtained from a call to mysql_store_result().

The result set returned by mysql_stmt_result_metadata() contains only metadata. It does not contain any row results. The rows are obtained by using the statement handle with mysql_stmt_fetch().

Return Values

A MYSQL_RES result structure. NULL if no meta information exists for the prepared query.

Errors

- CR_OUT_OF_MEMORY

  Out of memory.
- CR_UNKNOWN_ERROR

  An unknown error occurred.

Example

See the Example in Section 21.8.11.11, “mysql_stmt_fetch()”.

21.8.11.24 mysql_stmt_row_seek()

```
MYSQL_ROW_OFFSET mysql_stmt_row_seek(MYSQL_STMT *stmt, MYSQL_ROW_OFFSET offset)
```

Description

Sets the row cursor to an arbitrary row in a statement result set. The `offset` value is a row offset that should be a value returned from `mysql_stmt_row_tell()` or from `mysql_stmt_row_seek()`. This value is not a row number; if you want to seek to a row within a result set by number, use `mysql_stmt_data_seek()` instead.

This function requires that the result set structure contains the entire result of the query, so `mysql_stmt_row_seek()` may be used only in conjunction with `mysql_stmt_store_result()`.

Return Values

The previous value of the row cursor. This value may be passed to a subsequent call to `mysql_stmt_row_seek()`.

Errors

None.

21.8.11.25 `mysql_stmt_row_tell()`

```
MYSQL_ROW_OFFSET mysql_stmt_row_tell(MYSQL_STMT *stmt)
```

Description

Returns the current position of the row cursor for the last `mysql_stmt_fetch()`. This value can be used as an argument to `mysql_stmt_row_seek()`.

You should use `mysql_stmt_row_tell()` only after `mysql_stmt_store_result()`.

Return Values

The current offset of the row cursor.

Errors

None.

21.8.11.26 `mysql_stmt_send_long_data()`

```
my_bool mysql_stmt_send_long_data(MYSQL_STMT *stmt, unsigned int
parameter_number, const char *data, unsigned long length)
```

Description

Enables an application to send parameter data to the server in pieces (or “chunks”). Call this function after `mysql_stmt_bind_param()` and before `mysql_stmt_execute()`. It can be called multiple times to send the parts of a character or binary data value for a column, which must be one of the `TEXT` or `BLOB` data types.

`parameter_number` indicates which parameter to associate the data with. Parameters are numbered beginning with 0. `data` is a pointer to a buffer containing data to be sent, and `length` indicates the number of bytes in the buffer.

Note

The next mysql_stmt_execute() call ignores the bind buffer for all parameters that have been used with mysql_stmt_send_long_data() since last mysql_stmt_execute() or mysql_stmt_reset().

If you want to reset/forget the sent data, you can do it with mysql_stmt_reset(). See Section 21.8.11.22, “mysql_stmt_reset()”.

The max_allowed_packet system variable controls the maximum size of parameter values that can be sent with mysql_stmt_send_long_data().

Return Values

Zero for success. Nonzero if an error occurred.

Errors

- CR_INVALID_BUFFER_USE

  The parameter does not have a string or binary type.

- CR_COMMANDS_OUT_OF_SYNC

  Commands were executed in an improper order.

- CR_SERVER_GONE_ERROR

  The MySQL server has gone away.

- CR_OUT_OF_MEMORY

  Out of memory.

- CR_UNKNOWN_ERROR

  An unknown error occurred.

Example

The following example demonstrates how to send the data for a TEXT column in chunks. It inserts the data value 'MySQL - The most popular Open Source database' into the text_column column. The mysql variable is assumed to be a valid connection handle.

```
#define INSERT_QUERY "INSERT INTO \
                      test_long_data(text_column) VALUES(?)"

MYSQL_BIND bind[1];
long       length;

stmt = mysql_stmt_init(mysql);
if (!stmt)
{
  fprintf(stderr, " mysql_stmt_init(), out of memory\n");
  exit(0);
}
if (mysql_stmt_prepare(stmt, INSERT_QUERY, strlen(INSERT_QUERY)))
{
  fprintf(stderr, "\n mysql_stmt_prepare(), INSERT failed");
  fprintf(stderr, "\n %s", mysql_stmt_error(stmt));
  exit(0);
}
 memset(bind, 0, sizeof(bind));
 bind[0].buffer_type= MYSQL_TYPE_STRING;
 bind[0].length= &length;
 bind[0].is_null= 0;

/* Bind the buffers */
if (mysql_stmt_bind_param(stmt, bind))
{
  fprintf(stderr, "\n param bind failed");
  fprintf(stderr, "\n %s", mysql_stmt_error(stmt));
  exit(0);
}

 /* Supply data in chunks to server */
 if (mysql_stmt_send_long_data(stmt,0,"MySQL",5))
{
  fprintf(stderr, "\n send_long_data failed");
  fprintf(stderr, "\n %s", mysql_stmt_error(stmt));
  exit(0);
}

 /* Supply the next piece of data */
 if (mysql_stmt_send_long_data(stmt,0,
           " - The most popular Open Source database",40))
{
  fprintf(stderr, "\n send_long_data failed");
  fprintf(stderr, "\n %s", mysql_stmt_error(stmt));
  exit(0);
}

 /* Now, execute the query */
 if (mysql_stmt_execute(stmt))
{
  fprintf(stderr, "\n mysql_stmt_execute failed");
  fprintf(stderr, "\n %s", mysql_stmt_error(stmt));
  exit(0);
}
```

21.8.11.27 `mysql_stmt_sqlstate()`

```
const char *mysql_stmt_sqlstate(MYSQL_STMT *stmt)
```

Description

For the statement specified by `stmt`, `mysql_stmt_sqlstate()` returns a null-terminated string containing the SQLSTATE error code for the most recently invoked prepared statement API function that can succeed or fail. The error code consists of five characters. `"00000"` means “no error.” The values are specified by ANSI SQL and ODBC. For a list of possible values, see Appendix C, *Errors, Error Codes, and Common Problems*.

Note that not all MySQL errors are yet mapped to SQLSTATE codes. The value `"HY000"` (general error) is used for unmapped errors.

Return Values

A null-terminated character string containing the SQLSTATE error code.

21.8.11.28 `mysql_stmt_store_result()`

```
int mysql_stmt_store_result(MYSQL_STMT *stmt)
```

Description

Result sets are produced by calling `mysql_stmt_execute()` to executed prepared statements for SQL statements such as `SELECT`, `SHOW`, `DESCRIBE`, and `EXPLAIN`. By default, result sets for successfully executed prepared statements are not buffered on the client and `mysql_stmt_fetch()` fetches them one at a time from the server. To cause the complete result set to be buffered on the client, call `mysql_stmt_store_result()` after binding data buffers with `mysql_stmt_bind_result()` and before calling `mysql_stmt_fetch()` to fetch rows. (For an example, see Section 21.8.11.11, “`mysql_stmt_fetch()`”.)

`mysql_stmt_store_result()` is optional for result set processing, unless you will call `mysql_stmt_data_seek()`, `mysql_stmt_row_seek()`, or `mysql_stmt_row_tell()`. Those functions require a seekable result set.

It is unnecessary to call `mysql_stmt_store_result()` after executing an SQL statement that does not produce a result set, but if you do, it does not harm or cause any notable performance problem. You can detect whether the statement produced a result set by checking if `mysql_stmt_result_metadata()` returns `NULL`. For more information, refer to Section 21.8.11.23, “`mysql_stmt_result_metadata()`”.

> **Note**
>
> MySQL does not by default calculate `MYSQL_FIELD->max_length` for all columns in `mysql_stmt_store_result()` because calculating this would slow down `mysql_stmt_store_result()` considerably and most applications do not need `max_length`. If you want `max_length` to be updated, you can call `mysql_stmt_attr_set(MYSQL_STMT, STMT_ATTR_UPDATE_MAX_LENGTH, &flag)` to enable this. See Section 21.8.11.3, “`mysql_stmt_attr_set()`”.

Return Values

Zero for success. Nonzero if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC`

  Commands were executed in an improper order.

- `CR_OUT_OF_MEMORY`

  Out of memory.

- `CR_SERVER_GONE_ERROR`

  The MySQL server has gone away.

- `CR_SERVER_LOST`

  The connection to the server was lost during the query.

- `CR_UNKNOWN_ERROR`

  An unknown error occurred.

21.8.12 C API Threaded Function Descriptions

To create a threaded client, use the functions described in the following sections. See also Section 21.8.4.2, “Writing C API Threaded Client Programs”.

21.8.12.1 `my_init()`

```
void my_init(void)
```

Description

`my_init()` initializes some global variables that MySQL needs. It also calls `mysql_thread_init()` for this thread.

It is necessary for `my_init()` to be called early in the initialization phase of a program's use of the MySQL library. However, `my_init()` is automatically called by `mysql_init()`, `mysql_library_init()`, `mysql_server_init()`, and `mysql_connect()`. If you ensure that your program invokes one of those functions before any other MySQL calls, there is no need to invoke `my_init()` explicitly.

To access the prototype for `my_init()`, your program should include these header files:

```
#include <my_global.h>
#include <my_sys.h>
```

Return Values

None.

21.8.12.2 `mysql_thread_end()`

```
void mysql_thread_end(void)
```

Description

This function needs to be called before calling `pthread_exit()` to free memory allocated by `mysql_thread_init()`.

`mysql_thread_end()` *is not invoked automatically by the client library*. It must be called explicitly to avoid a memory leak.

Return Values

None.

21.8.12.3 `mysql_thread_init()`

```
my_bool mysql_thread_init(void)
```

Description

This function must be called early within each created thread to initialize thread-specific variables. However, you may not necessarily need to invoke it explicitly: `mysql_thread_init()` is automatically called by `my_init()`, which itself is automatically called by `mysql_init()`, `mysql_library_init()`, `mysql_server_init()`, and `mysql_connect()`. If you invoke any of those functions, `mysql_thread_init()` will be called for you.

Return Values

Zero for success. Nonzero if an error occurred.

21.8.12.4 `mysql_thread_safe()`

```
unsigned int mysql_thread_safe(void)
```

Description

This function indicates whether the client library is compiled as thread-safe.

Return Values

1 if the client library is thread-safe, 0 otherwise.

21.8.13 C API Embedded Server Function Descriptions

MySQL applications can be written to use an embedded server. See Section 21.7, “libmysqld, the Embedded MySQL Server Library”. To write such an application, you must link it against the `libmysqld` library by using the `-lmysqld` flag rather than linking it against the `libmysqlclient` client library by using the `-lmysqlclient` flag. However, the calls to initialize and finalize the library are the same whether you write a client application or one that uses the embedded server: Call `mysql_library_init()` to initialize the library and `mysql_library_end()` when you are done with it. See Section 21.8.6, “C API Function Overview”.

21.8.13.1 `mysql_server_init()`

```
int mysql_server_init(int argc, char **argv, char **groups)
```

Description

This function initializes the MySQL library, which must be done before you call any other MySQL function. However, `mysql_server_init()` is deprecated and you should call `mysql_library_init()` instead. See Section 21.8.7.41, “`mysql_library_init()`”.

Return Values

Zero for success. Nonzero if an error occurred.

21.8.13.2 `mysql_server_end()`

```
void mysql_server_end(void)
```

Description

This function finalizes the MySQL library, which should be done when you are done using the library. However, `mysql_server_end()` is deprecated and `mysql_library_end()` should be used instead. See Section 21.8.7.40, “`mysql_library_end()`”.

Return Values

None.

21.8.14 C API Client Plugin Functions

This section describes functions used for the client-side plugin API. They enable management of client plugins. For a description of the `st_mysql_client_plugin` structure used by these functions, see Client Plugin Descriptors.

It is unlikely that a client program needs to call the functions in this section. For example, a client that supports the use of authentication plugins normally causes a plugin to be loaded by calling `mysql_options()` to set the `MYSQL_DEFAULT_AUTH` and `MYSQL_PLUGIN_DIR` options:

```
char *plugin_dir = "path_to_plugin_dir";
char *default_auth = "plugin_name";

/* ... process command-line options ... */

mysql_options(&mysql, MYSQL_PLUGIN_DIR, plugin_dir);
mysql_options(&mysql, MYSQL_DEFAULT_AUTH, default_auth);
```

Typically, the program will also accept `--plugin-dir` and `--default-auth` options that enable users to override the default values.

21.8.14.1 `mysql_client_find_plugin()`

`struct st_mysql_client_plugin *mysql_client_find_plugin(MYSQL *mysql, const char *name, int type)`

Description

Returns a pointer to a loaded plugin, loading the plugin first if necessary. An error occurs if the type is invalid or the plugin cannot be found or loaded.

Specify the parameters as follows:

- `mysql`: A pointer to a `MYSQL` structure. The plugin API does not require a connection to a MySQL server, but this structure must be properly initialized. The structure is used to obtain connection-related information.
- `name`: The plugin name.
- `type`: The plugin type.

Return Values

A pointer to the plugin for success. `NULL` if an error occurred.

Errors

To check for errors, call the `mysql_error()` or `mysql_errno()` function. See Section 21.8.7.15, “`mysql_error()`”, and Section 21.8.7.14, “`mysql_errno()`”.

Example

```
MYSQL mysql;
struct st_mysql_client_plugin *p;

if ((p = mysql_client_find_plugin(&mysql, "myplugin",
                                  MYSQL_CLIENT_AUTHENTICATION_PLUGIN, 0)))
{
  printf("Plugin version: %d.%d.%d\n", p->version[0], p->version[1], p->version[2]);
}
```

21.8.14.2 `mysql_client_register_plugin()`

`struct st_mysql_client_plugin *mysql_client_register_plugin(MYSQL *mysql, struct st_mysql_client_plugin *plugin)`

Description

Adds a plugin structure to the list of loaded plugins. An error occurs if the plugin is already loaded.

Specify the parameters as follows:

- `mysql`: A pointer to a `MYSQL` structure. The plugin API does not require a connection to a MySQL server, but this structure must be properly initialized. The structure is used to obtain connection-related information.

- `plugin`: A pointer to the plugin structure.

Return Values

A pointer to the plugin for success. `NULL` if an error occurred.

Errors

To check for errors, call the `mysql_error()` or `mysql_errno()` function. See Section 21.8.7.15, “`mysql_error()`”, and Section 21.8.7.14, “`mysql_errno()`”.

21.8.14.3 `mysql_load_plugin()`

```
struct st_mysql_client_plugin *mysql_load_plugin(MYSQL *mysql, const char
*name, int type, int argc, ...)
```

Description

Loads a MySQL client plugin, specified by name and type. An error occurs if the type is invalid or the plugin cannot be loaded.

It is not possible to load multiple plugins of the same type. An error occurs if you try to load a plugin of a type already loaded.

Specify the parameters as follows:

- `mysql`: A pointer to a `MYSQL` structure. The plugin API does not require a connection to a MySQL server, but this structure must be properly initialized. The structure is used to obtain connection-related information.

- `name`: The name of the plugin to load.

- `type`: The type of plugin to load, or –1 to disable type checking. If type is not –1, only plugins matching the type are considered for loading.

- `argc`: The number of following arguments (0 if there are none). Interpretation of any following arguments depends on the plugin type.

Another way to cause plugins to be loaded is to set the `LIBMYSQL_PLUGINS` environment variable to a semicolon-separated list of plugin names. For example:

```
shell> export LIBMYSQL_PLUGINS="myplugin1;myplugin2"
```

Plugins named by `LIBMYSQL_PLUGINS` are loaded when the client program calls `mysql_library_init()`. No error is reported if problems occur loading these plugins.

As of MySQL 5.7.1, the `LIBMYSQL_PLUGIN_DIR` environment variable can be set to the path name of the directory in which to look for client plugins. This variable is used in two ways:

- During client plugin preloading, the value of the `--plugin-dir` option is not available, so client plugin loading fails unless the plugins are located in the hardwired default directory. If the plugins are located elsewhere, LIBMYSQL_PLUGIN_DIR environment variable can be set to the proper directory to enable plugin preloading to succeed.

- For explicit client plugin loading, the mysql_load_plugin() and mysql_load_plugin_v() C API functions use the LIBMYSQL_PLUGIN_DIR value if it exists and the --plugin-dir option was not given. If --plugin-dir is given, mysql_load_plugin() and mysql_load_plugin_v() ignore LIBMYSQL_PLUGIN_DIR.

Return Values

A pointer to the plugin if it was loaded successfully. NULL if an error occurred.

Errors

To check for errors, call the mysql_error() or mysql_errno() function. See Section 21.8.7.15, “mysql_error()”, and Section 21.8.7.14, “mysql_errno()”.

Example

```
MYSQL mysql;

if(!mysql_load_plugin(&mysql, "myplugin",
                      MYSQL_CLIENT_AUTHENTICATION_PLUGIN, 0))
{
    fprintf(stderr, "Error: %s\n", mysql_error(&mysql));
    exit(-1);
}
```

See Also

See also Section 21.8.14.3, “mysql_load_plugin()”, Section 21.8.7.15, “mysql_error()”, Section 21.8.7.14, “mysql_errno()”.

21.8.14.4 mysql_load_plugin_v()

```
struct st_mysql_client_plugin *mysql_load_plugin_v(MYSQL *mysql, const char
*name, int type, int argc, va_list args)
```

Description

This function is equivalent to mysql_load_plugin(), but it accepts a va_list instead of a variable list of parameters.

See Also

See also Section 21.8.14.3, “mysql_load_plugin()”.

21.8.14.5 mysql_plugin_options()

```
int mysql_plugin_options(struct st_mysql_client_plugin *plugin, const char
*option, const void *value)
```

Description

Passes an option type and value to a plugin. This function can be called multiple times to set several options. If the plugin does not have an option handler, an error occurs.

Specify the parameters as follows:

- `plugin`: A pointer to the plugin structure.
- `option`: The option to be set.
- `value`: A pointer to the option value.

Return Values

Zero for success, 1 if an error occurred. If the plugin has an option handler, that handler should also return zero for success and 1 if an error occurred.

21.8.15 Common Questions and Problems When Using the C API

21.8.15.1 Why `mysql_store_result()` Sometimes Returns `NULL` After `mysql_query()` Returns Success

It is possible for `mysql_store_result()` to return `NULL` following a successful call to `mysql_query()`. When this happens, it means one of the following conditions occurred:

- There was a `malloc()` failure (for example, if the result set was too large).
- The data could not be read (an error occurred on the connection).
- The query returned no data (for example, it was an `INSERT`, `UPDATE`, or `DELETE`).

You can always check whether the statement should have produced a nonempty result by calling `mysql_field_count()`. If `mysql_field_count()` returns zero, the result is empty and the last query was a statement that does not return values (for example, an `INSERT` or a `DELETE`). If `mysql_field_count()` returns a nonzero value, the statement should have produced a nonempty result. See the description of the `mysql_field_count()` function for an example.

You can test for an error by calling `mysql_error()` or `mysql_errno()`.

21.8.15.2 What Results You Can Get from a Query

In addition to the result set returned by a query, you can also get the following information:

- `mysql_affected_rows()` returns the number of rows affected by the last query when doing an `INSERT`, `UPDATE`, or `DELETE`.

  For a fast re-create, use `TRUNCATE TABLE`.

- `mysql_num_rows()` returns the number of rows in a result set. With `mysql_store_result()`, `mysql_num_rows()` may be called as soon as `mysql_store_result()` returns. With `mysql_use_result()`, `mysql_num_rows()` may be called only after you have fetched all the rows with `mysql_fetch_row()`.

- `mysql_insert_id()` returns the ID generated by the last query that inserted a row into a table with an `AUTO_INCREMENT` index. See Section 21.8.7.38, “`mysql_insert_id()`”.

- Some queries (`LOAD DATA INFILE ...`, `INSERT INTO ... SELECT ...`, `UPDATE`) return additional information. The result is returned by `mysql_info()`. See the description for `mysql_info()` for the format of the string that it returns. `mysql_info()` returns a `NULL` pointer if there is no additional information.

21.8.15.3 How to Get the Unique ID for the Last Inserted Row

If you insert a record into a table that contains an `AUTO_INCREMENT` column, you can obtain the value stored into that column by calling the `mysql_insert_id()` function.

You can check from your C applications whether a value was stored in an `AUTO_INCREMENT` column by executing the following code (which assumes that you've checked that the statement succeeded). It determines whether the query was an `INSERT` with an `AUTO_INCREMENT` index:

```
if ((result = mysql_store_result(&mysql)) == 0 &&
    mysql_field_count(&mysql) == 0 &&
    mysql_insert_id(&mysql) != 0)
{
    used_id = mysql_insert_id(&mysql);
}
```

When a new `AUTO_INCREMENT` value has been generated, you can also obtain it by executing a `SELECT LAST_INSERT_ID()` statement with `mysql_query()` and retrieving the value from the result set returned by the statement.

When inserting multiple values, the last automatically incremented value is returned.

For `LAST_INSERT_ID()`, the most recently generated ID is maintained in the server on a per-connection basis. It is not changed by another client. It is not even changed if you update another `AUTO_INCREMENT` column with a nonmagic value (that is, a value that is not `NULL` and not `0`). Using `LAST_INSERT_ID()` and `AUTO_INCREMENT` columns simultaneously from multiple clients is perfectly valid. Each client will receive the last inserted ID for the last statement *that* client executed.

If you want to use the ID that was generated for one table and insert it into a second table, you can use SQL statements like this:

```
INSERT INTO foo (auto,text)
    VALUES(NULL,'text');         # generate ID by inserting NULL
INSERT INTO foo2 (id,text)
    VALUES(LAST_INSERT_ID(),'text');  # use ID in second table
```

Note that `mysql_insert_id()` returns the value stored into an `AUTO_INCREMENT` column, whether that value is automatically generated by storing `NULL` or `0` or was specified as an explicit value. `LAST_INSERT_ID()` returns only automatically generated `AUTO_INCREMENT` values. If you store an explicit value other than `NULL` or `0`, it does not affect the value returned by `LAST_INSERT_ID()`.

For more information on obtaining the last ID in an `AUTO_INCREMENT` column:

- For information on `LAST_INSERT_ID()`, which can be used within an SQL statement, see Section 12.14, “Information Functions”.
- For information on `mysql_insert_id()`, the function you use from within the C API, see Section 21.8.7.38, “`mysql_insert_id()`”.
- For information on obtaining the auto-incremented value when using Connector/J, see Retrieving `AUTO_INCREMENT` Column Values through JDBC.
- For information on obtaining the auto-incremented value when using Connector/ODBC, see Obtaining Auto-Increment Values.

21.8.16 Controlling Automatic Reconnection Behavior

The MySQL client library can perform an automatic reconnection to the server if it finds that the connection is down when you attempt to send a statement to the server to be executed. If auto-reconnect is enabled, the library tries once to reconnect to the server and send the statement again.

In MySQL 5.7, auto-reconnect is disabled by default.

If it is important for your application to know that the connection has been dropped (so that is can exit or take action to adjust for the loss of state information), be sure that auto-reconnect is disabled. To ensure this, call `mysql_options()` with the `MYSQL_OPT_RECONNECT` option:

```
my_bool reconnect = 0;
mysql_options(&mysql, MYSQL_OPT_RECONNECT, &reconnect);
```

If the connection has gone down, the effect of `mysql_ping()` depends on the auto-reconnect state. If auto-reconnect is enabled, `mysql_ping()` performs a reconnect. Otherwise, it returns an error.

Some client programs might provide the capability of controlling automatic reconnection. For example, `mysql` reconnects by default, but the `--skip-reconnect` option can be used to suppress this behavior.

If an automatic reconnection does occur (for example, as a result of calling `mysql_ping()`), there is no explicit indication of it. To check for reconnection, call `mysql_thread_id()` to get the original connection identifier before calling `mysql_ping()`, then call `mysql_thread_id()` again to see whether the identifier changed.

Automatic reconnection can be convenient because you need not implement your own reconnect code, but if a reconnection does occur, several aspects of the connection state are reset on the server side and your application will not be notified.

The connection-related state is affected as follows:

- Any active transactions are rolled back and autocommit mode is reset.
- All table locks are released.
- All `TEMPORARY` tables are closed (and dropped).
- Session system variables are reinitialized to the values of the corresponding global system variables, including system variables that are set implicitly by statements such as `SET NAMES`.
- User variable settings are lost.
- Prepared statements are released.
- `HANDLER` variables are closed.
- The value of `LAST_INSERT_ID()` is reset to 0.
- Locks acquired with `GET_LOCK()` are released.
- The association of the client with the Performance Schema `threads` table row that determines connection thread instrumentation is lost. If the client reconnects after a disconnect, the session is associated with a new row in the `threads` table and the thread monitoring state may be different. See Section 20.9.13.3, “The `threads` Table”.

If the connection drops, it is possible that the session associated with the connection on the server side will still be running if the server has not yet detected that the client is no longer connected. In this case, any locks held by the original connection still belong to that session, so you may want to kill it by calling `mysql_kill()`.

21.8.17 C API Support for Multiple Statement Execution

By default, `mysql_query()` and `mysql_real_query()` interpret their statement string argument as a single statement to be executed, and you process the result according to whether the statement produces a result set (a set of rows, as for `SELECT`) or an affected-rows count (as for `INSERT`, `UPDATE`, and so forth).

MySQL 5.7 also supports the execution of a string containing multiple statements separated by semicolon ("`;`") characters. This capability is enabled by special options that are specified either when you connect to the server with `mysql_real_connect()` or after connecting by calling` `mysql_set_server_option()`.

Executing a multiple-statement string can produce multiple result sets or row-count indicators. Processing these results involves a different approach than for the single-statement case: After handling the result from the first statement, it is necessary to check whether more results exist and process them in turn if so. To support multiple-result processing, the C API includes the `mysql_more_results()` and `mysql_next_result()` functions. These functions are used at the end of a loop that iterates as long as more results are available. *Failure to process the result this way may result in a dropped connection to the server.*

Multiple-result processing also is required if you execute `CALL` statements for stored procedures. Results from a stored procedure have these characteristics:

- Statements within the procedure may produce result sets (for example, if it executes `SELECT` statements). These result sets are returned in the order that they are produced as the procedure executes.

  In general, the caller cannot know how many result sets a procedure will return. Procedure execution may depend on loops or conditional statements that cause the execution path to differ from one call to the next. Therefore, you must be prepared to retrieve multiple results.

- The final result from the procedure is a status result that includes no result set. The status indicates whether the procedure succeeded or an error occurred.

The multiple statement and result capabilities can be used only with `mysql_query()` or `mysql_real_query()`. They cannot be used with the prepared statement interface. Prepared statement handles are defined to work only with strings that contain a single statement. See Section 21.8.8, "C API Prepared Statements".

To enable multiple-statement execution and result processing, the following options may be used:

- The `mysql_real_connect()` function has a `flags` argument for which two option values are relevant:

  - `CLIENT_MULTI_RESULTS` enables the client program to process multiple results. This option *must* be enabled if you execute `CALL` statements for stored procedures that produce result sets. Otherwise, such procedures result in an error `Error 1312 (0A000): PROCEDURE` *`proc_name`* `can't return a result set in the given context`. In MySQL 5.7, `CLIENT_MULTI_RESULTS` is enabled by default.

  - `CLIENT_MULTI_STATEMENTS` enables `mysql_query()` and `mysql_real_query()` to execute statement strings containing multiple statements separated by semicolons. This option also enables `CLIENT_MULTI_RESULTS` implicitly, so a `flags` argument of `CLIENT_MULTI_STATEMENTS` to `mysql_real_connect()` is equivalent to an argument of `CLIENT_MULTI_STATEMENTS | CLIENT_MULTI_RESULTS`. That is, `CLIENT_MULTI_STATEMENTS` is sufficient to enable multiple-statement execution and all multiple-result processing.

- After the connection to the server has been established, you can use the `mysql_set_server_option()` function to enable or disable multiple-statement execution by passing it an argument of `MYSQL_OPTION_MULTI_STATEMENTS_ON` or `MYSQL_OPTION_MULTI_STATEMENTS_OFF`. Enabling multiple-statement execution with this function also enables processing of “simple” results for a multiple-statement string where each statement produces a single result, but is *not* sufficient to permit processing of stored procedures that produce result sets.

The following procedure outlines a suggested strategy for handling multiple statements:

1. Pass `CLIENT_MULTI_STATEMENTS` to `mysql_real_connect()`, to fully enable multiple-statement execution and multiple-result processing.

2. After calling `mysql_query()` or `mysql_real_query()` and verifying that it succeeds, enter a loop within which you process statement results.

3. For each iteration of the loop, handle the current statement result, retrieving either a result set or an affected-rows count. If an error occurs, exit the loop.

4. At the end of the loop, call `mysql_next_result()` to check whether another result exists and initiate retrieval for it if so. If no more results are available, exit the loop.

One possible implementation of the preceding strategy is shown following. The final part of the loop can be reduced to a simple test of whether `mysql_next_result()` returns nonzero. The code as written distinguishes between no more results and an error, which enables a message to be printed for the latter occurrence.

```
/* connect to server with the CLIENT_MULTI_STATEMENTS option */
if (mysql_real_connect (mysql, host_name, user_name, password,
    db_name, port_num, socket_name, CLIENT_MULTI_STATEMENTS) == NULL)
{
  printf("mysql_real_connect() failed\n");
  mysql_close(mysql);
  exit(1);
}

/* execute multiple statements */
status = mysql_query(mysql,
                     "DROP TABLE IF EXISTS test_table;\
                      CREATE TABLE test_table(id INT);\
                      INSERT INTO test_table VALUES(10);\
                      UPDATE test_table SET id=20 WHERE id=10;\
                      SELECT * FROM test_table;\
                      DROP TABLE test_table");
if (status)
{
  printf("Could not execute statement(s)");
  mysql_close(mysql);
  exit(0);
}

/* process each statement result */
do {
  /* did current statement return data? */
  result = mysql_store_result(mysql);
  if (result)
  {
    /* yes; process rows and free the result set */
    process_result_set(mysql, result);
    mysql_free_result(result);
  }
  else          /* no result set or error */
  {
    if (mysql_field_count(mysql) == 0)
    {
      printf("%lld rows affected\n",
            mysql_affected_rows(mysql));
    }
    else  /* some error occurred */
    {
      printf("Could not retrieve result set\n");
      break;
    }
  }
  /* more results? -1 = no, >0 = error, 0 = yes (keep looping) */
  if ((status = mysql_next_result(mysql)) > 0)
    printf("Could not execute statement\n");
} while (status == 0);

mysql_close(mysql);
```

21.8.18 C API Prepared Statement Problems

Here follows a list of the currently known problems with prepared statements:

- `TIME`, `TIMESTAMP`, and `DATETIME` do not support parts of seconds (for example, from `DATE_FORMAT()`).
- When converting an integer to string, `ZEROFILL` is honored with prepared statements in some cases where the MySQL server does not print the leading zeros. (For example, with `MIN(`*`number-with-zerofill`*`)`).
- When converting a floating-point number to a string in the client, the rightmost digits of the converted value may differ slightly from those of the original value.
- Prepared statements use the query cache under the conditions described in Section 8.9.3.1, "How the Query Cache Operates".
- Prepared statements do not support multi-statements (that is, multiple statements within a single string separated by "`;`" characters).
- The capabilities of prepared `CALL` statements are described in Section 21.8.20, "C API Support for Prepared `CALL` Statements".

21.8.19 C API Prepared Statement Handling of Date and Time Values

The binary (prepared statement) protocol enables you to send and receive date and time values (`DATE`, `TIME`, `DATETIME`, and `TIMESTAMP`), using the `MYSQL_TIME` structure. The members of this structure are described in Section 21.8.9, "C API Prepared Statement Data Structures".

To send temporal data values, create a prepared statement using `mysql_stmt_prepare()`. Then, before calling `mysql_stmt_execute()` to execute the statement, use the following procedure to set up each temporal parameter:

1. In the `MYSQL_BIND` structure associated with the data value, set the `buffer_type` member to the type that indicates what kind of temporal value you're sending. For `DATE`, `TIME`, `DATETIME`, or `TIMESTAMP` values, set `buffer_type` to `MYSQL_TYPE_DATE`, `MYSQL_TYPE_TIME`, `MYSQL_TYPE_DATETIME`, or `MYSQL_TYPE_TIMESTAMP`, respectively.
2. Set the `buffer` member of the `MYSQL_BIND` structure to the address of the `MYSQL_TIME` structure in which you pass the temporal value.

3. Fill in the members of the MYSQL_TIME structure that are appropriate for the type of temporal value to be passed.

Use mysql_stmt_bind_param() to bind the parameter data to the statement. Then you can call mysql_stmt_execute().

To retrieve temporal values, the procedure is similar, except that you set the buffer_type member to the type of value you expect to receive, and the buffer member to the address of a MYSQL_TIME structure into which the returned value should be placed. Use mysql_stmt_bind_result() to bind the buffers to the statement after calling mysql_stmt_execute() and before fetching the results.

Here is a simple example that inserts DATE, TIME, and TIMESTAMP data. The mysql variable is assumed to be a valid connection handle.

```
MYSQL_TIME  ts;
MYSQL_BIND  bind[3];
MYSQL_STMT  *stmt;

strmov(query, "INSERT INTO test_table(date_field, time_field, \
                             timestamp_field) VALUES(?,?,?");

stmt = mysql_stmt_init(mysql);
if (!stmt)
{
  fprintf(stderr, " mysql_stmt_init(), out of memory\n");
  exit(0);
}
if (mysql_stmt_prepare(mysql, query, strlen(query)))
{
  fprintf(stderr, "\n mysql_stmt_prepare(), INSERT failed");
  fprintf(stderr, "\n %s", mysql_stmt_error(stmt));
  exit(0);
}

/* set up input buffers for all 3 parameters */
bind[0].buffer_type= MYSQL_TYPE_DATE;
bind[0].buffer= (char *)&ts;
bind[0].is_null= 0;
bind[0].length= 0;
...
bind[1]= bind[2]= bind[0];
...

mysql_stmt_bind_param(stmt, bind);

/* supply the data to be sent in the ts structure */
ts.year= 2002;
ts.month= 02;
ts.day= 03;

ts.hour= 10;
ts.minute= 45;
ts.second= 20;

mysql_stmt_execute(stmt);
..
```

21.8.20 C API Support for Prepared CALL Statements

This section describes prepared-statement support in the C API for stored procedures executed using CALL statements:

In MySQL 5.7, stored procedures executed using prepared CALL statements can be used in the following ways:

- A stored procedure can produce any number of result sets. The number of columns and the data types of the columns need not be the same for all result sets.
- The final values of OUT and INOUT parameters are available to the calling application after the procedure returns. These parameters are returned as an extra single-row result set following any result sets produced by the procedure itself. The row contains the values of the OUT and INOUT parameters in the order in which they are declared in the procedure parameter list.

The following discussion shows how to use these capabilities through the C API for prepared statements. To use prepared CALL statements through the PREPARE and EXECUTE statements, see Section 13.2.1, “CALL Syntax”.

If an application might be compiled or executed in a context where a version of MySQL older than 5.5.3 is used, prepared CALL capabilities for multiple result sets and OUT or INOUT parameters might not be available:

- For the client side, the application will not compile unless the libraries are from MySQL 5.5.3 or higher (the API function and symbols introduced in that version will not be present).
- To verify at runtime that the server is recent enough, a client can use this test:

  ```
  if (mysql_get_server_version(mysql) < 50503)
  {
    fprintf(stderr,
            "Server does not support required CALL capabilities\n");
    mysql_close(mysql);
    exit (1);
  }
  ```

An application that executes a prepared CALL statement should use a loop that fetches a result and then invokes mysql_stmt_next_result() to determine whether there are more results. The results consist of any result sets produced by the stored procedure followed by a final status value that indicates whether the procedure terminated successfully.

If the procedure has OUT or INOUT parameters, the result set preceding the final status value contains their values. To determine whether a result set contains parameter values, test whether the SERVER_PS_OUT_PARAMS bit is set in the server_status member of the MYSQL connection handler:

```
mysql->server_status & SERVER_PS_OUT_PARAMS
```

The following example uses a prepared CALL statement to execute a stored procedure that produces multiple result sets and that provides parameter values back to the caller by means of OUT and INOUT parameters. The procedure takes parameters of all three types (IN, OUT, INOUT), displays their initial values, assigns new values, displays the updated values, and returns. The expected return information from the procedure therefore consists of multiple result sets and a final status:

- One result set from a SELECT that displays the initial parameter values: 10, NULL, 30. (The OUT parameter is assigned a value by the caller, but this assignment is expected to be ineffective: OUT parameters are seen as NULL within a procedure until assigned a value within the procedure.)
- One result set from a SELECT that displays the modified parameter values: 100, 200, 300.
- One result set containing the final OUT and INOUT parameter values: 200, 300.

- A final status packet.

The code to execute the procedure:

```
MYSQL_STMT *stmt;
MYSQL_BIND ps_params[3];  /* input parameter buffers */
int        int_data[3];   /* input/output values */
my_bool    is_null[3];    /* output value nullability */
int        status;

/* set up stored procedure */
status = mysql_query(mysql, "DROP PROCEDURE IF EXISTS p1");
test_error(mysql, status);

status = mysql_query(mysql,
  "CREATE PROCEDURE p1("
  "  IN p_in INT, "
  "  OUT p_out INT, "
  "  INOUT p_inout INT) "
  "BEGIN "
  "  SELECT p_in, p_out, p_inout; "
  "  SET p_in = 100, p_out = 200, p_inout = 300; "
  "  SELECT p_in, p_out, p_inout; "
  "END");
test_error(mysql, status);

/* initialize and prepare CALL statement with parameter placeholders */
stmt = mysql_stmt_init(mysql);
if (!stmt)
{
  printf("Could not initialize statement\n");
  exit(1);
}
status = mysql_stmt_prepare(stmt, "CALL p1(?, ?, ?)", 16);
test_stmt_error(stmt, status);

/* initialize parameters: p_in, p_out, p_inout (all INT) */
memset(ps_params, 0, sizeof (ps_params));

ps_params[0].buffer_type = MYSQL_TYPE_LONG;
ps_params[0].buffer = (char *) &int_data[0];
ps_params[0].length = 0;
ps_params[0].is_null = 0;

ps_params[1].buffer_type = MYSQL_TYPE_LONG;
ps_params[1].buffer = (char *) &int_data[1];
ps_params[1].length = 0;
ps_params[1].is_null = 0;

ps_params[2].buffer_type = MYSQL_TYPE_LONG;
ps_params[2].buffer = (char *) &int_data[2];
ps_params[2].length = 0;
ps_params[2].is_null = 0;

/* bind parameters */
status = mysql_stmt_bind_param(stmt, ps_params);
test_stmt_error(stmt, status);

/* assign values to parameters and execute statement */
int_data[0]= 10;  /* p_in */
int_data[1]= 20;  /* p_out */
int_data[2]= 30;  /* p_inout */

status = mysql_stmt_execute(stmt);
test_stmt_error(stmt, status);
```

```
/* process results until there are no more */
do {
  int i;
  int num_fields;       /* number of columns in result */
  MYSQL_FIELD *fields;  /* for result set metadata */
  MYSQL_BIND *rs_bind;  /* for output buffers */

  /* the column count is > 0 if there is a result set */
  /* 0 if the result is only the final status packet */
  num_fields = mysql_stmt_field_count(stmt);

  if (num_fields > 0)
  {
    /* there is a result set to fetch */
    printf("Number of columns in result: %d\n", (int) num_fields);

    /* what kind of result set is this? */
    printf("Data: ");
    if(mysql->server_status & SERVER_PS_OUT_PARAMS)
      printf("this result set contains OUT/INOUT parameters\n");
    else
      printf("this result set is produced by the procedure\n");

    MYSQL_RES *rs_metadata = mysql_stmt_result_metadata(stmt);
    test_stmt_error(stmt, rs_metadata == NULL);

    fields = mysql_fetch_fields(rs_metadata);

    rs_bind = (MYSQL_BIND *) malloc(sizeof (MYSQL_BIND) * num_fields);
    if (!rs_bind)
    {
      printf("Cannot allocate output buffers\n");
      exit(1);
    }
    memset(rs_bind, 0, sizeof (MYSQL_BIND) * num_fields);

    /* set up and bind result set output buffers */
    for (i = 0; i < num_fields; ++i)
    {
      rs_bind[i].buffer_type = fields[i].type;
      rs_bind[i].is_null = &is_null[i];

      switch (fields[i].type)
      {
        case MYSQL_TYPE_LONG:
          rs_bind[i].buffer = (char *) &(int_data[i]);
          rs_bind[i].buffer_length = sizeof (int_data);
          break;

        default:
          fprintf(stderr, "ERROR: unexpected type: %d.\n", fields[i].type);
          exit(1);
      }
    }

    status = mysql_stmt_bind_result(stmt, rs_bind);
    test_stmt_error(stmt, status);

    /* fetch and display result set rows */
    while (1)
    {
      status = mysql_stmt_fetch(stmt);

      if (status == 1 || status == MYSQL_NO_DATA)
        break;

      for (i = 0; i < num_fields; ++i)
      {
        switch (rs_bind[i].buffer_type)
        {
          case MYSQL_TYPE_LONG:
            if (*rs_bind[i].is_null)
              printf(" val[%d] = NULL;", i);
            else
              printf(" val[%d] = %ld;",
                     i, (long) *((int *) rs_bind[i].buffer));
            break;

          default:
            printf("  unexpected type (%d)\n",
              rs_bind[i].buffer_type);
        }
      }
      printf("\n");
    }

    mysql_free_result(rs_metadata); /* free metadata */
    free(rs_bind);                  /* free output buffers */
  }
  else
  {
    /* no columns = final status packet */
    printf("End of procedure output\n");
  }

  /* more results? -1 = no, >0 = error, 0 = yes (keep looking) */
  status = mysql_stmt_next_result(stmt);
  if (status > 0)
    test_stmt_error(stmt, status);
} while (status == 0);

mysql_stmt_close(stmt);
```

Execution of the procedure should produce the following output:

```
Number of columns in result: 3
Data: this result set is produced by the procedure
 val[0] = 10; val[1] = NULL; val[2] = 30;
Number of columns in result: 3
Data: this result set is produced by the procedure
 val[0] = 100; val[1] = 200; val[2] = 300;
Number of columns in result: 2
Data: this result set contains OUT/INOUT parameters
 val[0] = 200; val[1] = 300;
End of procedure output
```

The code uses two utility routines, `test_error()` and `test_stmt_error()`, to check for errors and terminate after printing diagnostic information if an error occurred:

```
static void test_error(MYSQL *mysql, int status)
{
  if (status)
  {
    fprintf(stderr, "Error: %s (errno: %d)\n",
            mysql_error(mysql), mysql_errno(mysql));
    exit(1);
  }
}

static void test_stmt_error(MYSQL_STMT *stmt, int status)
{
  if (status)
  {
    fprintf(stderr, "Error: %s (errno: %d)\n",
            mysql_stmt_error(stmt), mysql_stmt_errno(stmt));
    exit(1);
  }
}
```

21.9 MySQL PHP API

The MySQL PHP API manual is now published in standalone form, not as part of the MySQL Reference Manual. See MySQL and PHP.

21.10 MySQL Perl API

The Perl DBI module provides a generic interface for database access. You can write a DBI script that works with many different database engines without change. To use DBI with MySQL, install the following:

1. The DBI module.
2. The DBD::mysql module. This is the DataBase Driver (DBD) module for Perl.
3. Optionally, the DBD module for any other type of database server you want to access.

Perl DBI is the recommended Perl interface. It replaces an older interface called mysqlperl, which should be considered obsolete.

These sections contain information about using Perl with MySQL and writing MySQL applications in Perl:

- For installation instructions for Perl DBI support, see Section 2.12, “Perl Installation Notes”.
- For an example of reading options from option files, see Section 5.3.4, “Using Client Programs in a Multiple-Server Environment”.
- For secure coding tips, see Section 6.1.1, “Security Guidelines”.
- For debugging tips, see Section 22.4.1.4, “Debugging mysqld under gdb”.
- For some Perl-specific environment variables, see Section 2.11, “Environment Variables”.
- For considerations for running on Mac OS X, see Section 2.4.5, “Using the Bundled MySQL on Mac OS X Server”.
- For ways to quote string literals, see Section 9.1.1, “String Literals”.

DBI information is available at the command line, online, or in printed form:

- Once you have the DBI and DBD::mysql modules installed, you can get information about them at the command line with the perldoc command:

  ```
  shell> perldoc DBI
  shell> perldoc DBI::FAQ
  shell> perldoc DBD::mysql
  ```

  You can also use pod2man, pod2html, and so on to translate this information into other formats.

- For online information about Perl DBI, visit the DBI Web site, http://dbi.perl.org/. That site hosts a general DBI mailing list. Oracle Corporation hosts a list specifically about DBD::mysql; see Section 1.6.1, “MySQL Mailing Lists”.

- For printed information, the official DBI book is *Programming the Perl DBI* (Alligator Descartes and Tim Bunce, O'Reilly & Associates, 2000). Information about the book is available at the DBI Web site, http://dbi.perl.org/.

  For information that focuses specifically on using DBI with MySQL, see *MySQL and Perl for the Web* (Paul DuBois, New Riders, 2001). This book's Web site is http://www.kitebird.com/mysql-perl/.

21.11 MySQL Python API

`MySQLdb` is a third-party driver that provides MySQL support for Python, compliant with the Python DB API version 2.0. It can be found at http://sourceforge.net/projects/mysql-python/.

The new MySQL Connector/Python component provides an interface to the same Python API, and is built into the MySQL Server and supported by Oracle. See MySQL Connector/Python Developer Guide for details on the Connector, as well as coding guidelines for Python applications and sample Python code.

21.12 MySQL Ruby APIs

Two APIs are available for Ruby programmers developing MySQL applications:

- The MySQL/Ruby API is based on the `libmysqlclient` API library. For information on installing and using the MySQL/Ruby API, see Section 21.12.1, “The MySQL/Ruby API”.
- The Ruby/MySQL API is written to use the native MySQL network protocol (a native driver). For information on installing and using the Ruby/MySQL API, see Section 21.12.2, “The Ruby/MySQL API”.

For background and syntax information about the Ruby language, see Ruby Programming Language.

21.12.1 The MySQL/Ruby API

The MySQL/Ruby module provides access to MySQL databases using Ruby through `libmysqlclient`.

For information on installing the module, and the functions exposed, see MySQL/Ruby.

21.12.2 The Ruby/MySQL API

The Ruby/MySQL module provides access to MySQL databases using Ruby through a native driver interface using the MySQL network protocol.

For information on installing the module, and the functions exposed, see Ruby/MySQL.

21.13 MySQL Tcl API

`MySQLtcl` is a simple API for accessing a MySQL database server from the Tcl programming language. It can be found at http://www.xdobry.de/mysqltcl/.

21.14 MySQL Eiffel Wrapper

Eiffel MySQL is an interface to the MySQL database server using the Eiffel programming language, written by Michael Ravits. It can be found at http://efsa.sourceforge.net/archive/ravits/mysql.htm.

Chapter 22 Extending MySQL

Table of Contents

22.1 MySQL Internals

This chapter describes a lot of things that you need to know when working on the MySQL code. To track or contribute to MySQL development, follow the instructions in Section 2.8.3, “Installing MySQL Using a Development Source Tree”. If you are interested in MySQL internals, you should also subscribe to our `internals` mailing list. This list has relatively low traffic. For details on how to subscribe, please see Section 1.6.1, “MySQL Mailing Lists”. Many MySQL developers at Oracle Corporation are on the `internals` list and we help other people who are working on the MySQL code. Feel free to use this list both to ask questions about the code and to send patches that you would like to contribute to the MySQL project!

22.1.1 MySQL Threads

The MySQL server creates the following threads:

- Connection manager threads handle client connection requests on the network interfaces that the server listens to. On all platforms, one manager thread handles TCP/IP connection requests. On Unix, this manager thread also handles Unix socket file connection requests. On Windows, a manager thread handles shared-memory connection requests, and another handles named-pipe connection requests. The server does not create threads to handle interfaces that it does not listen to. For example, a Windows server that does not have support for named-pipe connections enabled does not create a thread to handle them.

- Connection manager threads associate each client connection with a thread dedicated to it that handles authentication and request processing for that connection. Manager threads create a new thread when necessary but try to avoid doing so by consulting the thread cache first to see whether it contains a thread that can be used for the connection. When a connection ends, its thread is returned to the thread cache if the cache is not full.

  For information about tuning the parameters that control thread resources, see Section 8.11.5.1, “How MySQL Uses Threads for Client Connections”.

- On a master replication server, connections from slave servers are handled like client connections: There is one thread per connected slave.
- On a slave replication server, an I/O thread is started to connect to the master server and read updates from it. An SQL thread is started to apply updates read from the master. These two threads run independently and can be started and stopped independently.
- A signal thread handles all signals. This thread also normally handles alarms and calls process_alarm() to force timeouts on connections that have been idle too long.
- If InnoDB is used, there will be additional read and write threads by default. The number of these are controlled by the innodb_read_io_threads and innodb_write_io_threads parameters. See Section 14.2.13, "InnoDB Startup Options and System Variables".
- If the server is started with the --flush_time=*val* option, a dedicated thread is created to flush all tables every *val* seconds.
- If the event scheduler is active, there is one thread for the scheduler, and a thread for each event currently running. See Section 18.4.1, "Event Scheduler Overview".

mysqladmin processlist only shows the connection, replication, and event threads.

22.1.2 The MySQL Test Suite

The test system that is included in Unix source and binary distributions makes it possible for users and developers to perform regression tests on the MySQL code. These tests can be run on Unix.

You can also write your own test cases. For information about the MySQL Test Framework, including system requirements, see the manual available at http://dev.mysql.com/doc/mysqltest/2.0/en/.

The current set of test cases doesn't test everything in MySQL, but it should catch most obvious bugs in the SQL processing code, operating system or library issues, and is quite thorough in testing replication. Our goal is to have the tests cover 100% of the code. We welcome contributions to our test suite. You may especially want to contribute tests that examine the functionality critical to your system because this ensures that all future MySQL releases work well with your applications.

The test system consists of a test language interpreter (mysqltest), a Perl script to run all tests (mysql-test-run.pl), the actual test cases written in a special test language, and their expected results. To run the test suite on your system after a build, type make test from the source root directory, or change location to the mysql-test directory and type ./mysql-test-run.pl. If you have installed a binary distribution, change location to the mysql-test directory under the installation root directory (for example, /usr/local/mysql/mysql-test), and run ./mysql-test-run.pl. All tests should succeed. If any do not, feel free to try to find out why and report the problem if it indicates a bug in MySQL. See Section 1.7, "How to Report Bugs or Problems".

If one test fails, you should run mysql-test-run.pl with the --force option to check whether any other tests fail.

If you have a copy of mysqld running on the machine where you want to run the test suite, you do not have to stop it, as long as it is not using ports 9306 or 9307. If either of those ports is taken, you should set the MTR_BUILD_THREAD environment variable to an appropriate value, and the test suite will use a different set of ports for master, slave, and NDB). For example:

```
shell> export MTR_BUILD_THREAD=31
shell> ./mysql-test-run.pl [options] [test_name]
```

In the mysql-test directory, you can run an individual test case with ./mysql-test-run.pl *test_name*.

If you have a question about the test suite, or have a test case to contribute, send an email message to the MySQL internals mailing list. See Section 1.6.1, “MySQL Mailing Lists”.

22.2 The MySQL Plugin API

MySQL supports a plugin API that enables creation of server components. Plugins can be loaded at server startup, or loaded and unloaded at runtime without restarting the server. The API is generic and does not specify what plugins can do. The components supported by this interface include, but are not limited to, storage engines, full-text parser plugins, and server extensions.

For example, full-text parser plugins can be used to replace or augment the built-in full-text parser. A plugin can parse text into words using rules that differ from those used by the built-in parser. This can be useful if you need to parse text with characteristics different from those expected by the built-in parser.

The plugin interface is more general than the older user-defined function (UDF) interface.

The plugin interface uses the plugin table in the mysql database to record information about plugins that have been installed permanently with the INSTALL PLUGIN statement. This table is created as part of the MySQL installation process. Plugins can also be installed for a single server invocation with the --plugin-load option. Plugins installed this way are not recorded in the plugin table. See Section 5.1.8.1, “Installing and Uninstalling Plugins”.

MySQL 5.7 supports an API for client plugins in addition to that for server plugins. This is used, for example, by authentication plugins where a server-side plugin and a client-side plugin cooperate to enable clients to connect to the server through a variety of authentication methods.

Additional Resources

The book *MySQL 5.1 Plugin Development* by Sergei Golubchik and Andrew Hutchings provides a wealth of detail about the plugin API. Despite the fact that the book's title refers to MySQL Server 5.1, most of the information in it applies to later versions as well.

22.2.1 Plugin API Characteristics

The server plugin API has these characteristics:

- All plugins have several things in common.

  Each plugin has a name that it can be referred to in SQL statements, as well as other metadata such as an author and a description that provide other information. This information can be examined in the INFORMATION_SCHEMA.PLUGINS table or using the SHOW PLUGINS statement.

- The plugin framework is extendable to accommodate different kinds of plugins.

  Although some aspects of the plugin API are common to all types of plugins, the API also permits type-specific interface elements so that different types of plugins can be created. A plugin with one purpose can have an interface most appropriate to its own requirements and not the requirements of some other plugin type.

  Interfaces for several types of plugins exist, such as storage engines, full-text parser, and INFORMATION_SCHEMA tables. Others can be added.

- Plugins can expose information to users.

  A plugin can implement system and status variables that are available through the SHOW VARIABLES and SHOW STATUS statements.

- The plugin API includes versioning information.

  The version information included in the plugin API enables a plugin library and each plugin that it contains to be self-identifying with respect to the API version that was used to build the library. If the API changes over time, the version numbers will change, but a server can examine a given plugin library's version information to determine whether it supports the plugins in the library.

  There are two types of version numbers. The first is the version for the general plugin framework itself. Each plugin library includes this kind of version number. The second type of version applies to individual plugins. Each specific type of plugin has a version for its interface, so each plugin in a library has a type-specific version number. For example, a library containing a full-text parser plugin has a general plugin API version number, and the plugin has a version number specific to the full-text plugin interface.

- The plugin API implements security restrictions.

  A plugin library must be installed in a specific dedicated directory for which the location is controlled by the server and cannot be changed at runtime. Also, the library must contain specific symbols that identify it as a plugin library. The server will not load something as a plugin if it was not built as a plugin.

- Plugins have access to server services.

  The services interface exposes server functionality that plugins can access using ordinary function calls. For details, see Section 22.2.5, “MySQL Services for Plugins”.

In some respects, the server plugin API is similar to the older user-defined function (UDF) API that it supersedes, but the plugin API has several advantages over the older interface. For example, UDFs had no versioning information. Also, the newer plugin interface eliminates the security issues of the older UDF interface. The older interface for writing nonplugin UDFs permitted libraries to be loaded from any directory searched by the system's dynamic linker, and the symbols that identified the UDF library were relatively nonspecific.

The client plugin API has similar architectural characteristics, but client plugins have no direct access to the server the way server plugins do.

22.2.2 Plugin API Components

The server plugin implementation comprises several components.

SQL statements:

- INSTALL PLUGIN registers a plugin in the mysql.plugin table and loads the plugin code.
- UNINSTALL PLUGIN unregisters a plugin from the mysql.plugin table and unloads the plugin code.
- The WITH PARSER clause for full-text index creation associates a full-text parser plugin with a given FULLTEXT index.
- SHOW PLUGINS displays information about server plugins.

Command-line options and system variables:

- The --plugin-load option enables plugins to be loaded at server startup time.

- The plugin_dir system variable indicates the location of the directory where all plugins must be installed. The value of this variable can be specified at server startup with a --plugin_dir=*path* option. mysql_config --plugindir displays the default plugin directory path name.

For additional information about plugin loading, see Section 5.1.8.1, “Installing and Uninstalling Plugins”.

Plugin-related tables:

- The INFORMATION_SCHEMA.PLUGINS table contains plugin information.
- The mysql.plugin table lists each plugin that was installed with INSTALL PLUGIN and is required for plugin use. For new MySQL installations, this table is created during the installation process.

The client plugin implementation is simpler:

- For the mysql_options() C API function, the MYSQL_DEFAULT_AUTH and MYSQL_PLUGIN_DIR options enable client programs to load authentication plugins.
- There are C API functions that enable management of client plugins.

To examine how MySQL implements plugins, consult the following source files in a MySQL source distribution:

- In the include/mysql directory, plugin.h exposes the public plugin API. This file should be examined by anyone who wants to write a plugin library. plugin_*xxx*.h files provide additional information that pertains to specific types of plugins. client_plugin.h contains information specific to client plugins.
- In the sql directory, sql_plugin.h and sql_plugin.cc comprise the internal plugin implementation. sql_acl.cc is where the server uses authentication plugins. These files need not be consulted by plugin developers. They may be of interest for those who want to know more about how the server handles plugins.
- In the sql-common directory, client_plugin.h implements the C API client plugin functions, and client.c implements client authentication support. These files need not be consulted by plugin developers. They may be of interest for those who want to know more about how the server handles plugins.

22.2.3 Types of Plugins

The plugin API enables creation of plugins that implement several capabilities:

- Storage engines
- Full-text parsers
- Daemons
- INFORMATION_SCHEMA tables
- Semisynchronous replication
- Auditing
- Authentication

The following sections provide an overview of these plugin types.

22.2.3.1 Storage Engine Plugins

The pluggable storage engine architecture used by MySQL Server enables storage engines to be written as plugins and loaded into and unloaded from a running server. For a description of this architecture, see Section 14.12, “Overview of MySQL Storage Engine Architecture”.

For information on how to use the plugin API to write storage engines, see MySQL Internals: Writing a Custom Storage Engine.

22.2.3.2 Full-Text Parser Plugins

MySQL has a built-in parser that it uses by default for full-text operations (parsing text to be indexed, or parsing a query string to determine the terms to be used for a search). For full-text processing, “parsing” means extracting words from text or a query string based on rules that define which character sequences make up a word and where word boundaries lie.

When parsing for indexing purposes, the parser passes each word to the server, which adds it to a full-text index. When parsing a query string, the parser passes each word to the server, which accumulates the words for use in a search.

The parsing properties of the built-in full-text parser are described in Section 12.9, “Full-Text Search Functions”. These properties include rules for determining how to extract words from text. The parser is influenced by certain system variables such as `innodb_ft_min_token_size` and `innodb_ft_max_token_size` for `InnoDB` or `ft_min_word_len` and `ft_max_word_len` for `MyISAM` that cause words shorter or longer to be excluded, and by the stopword list that identifies common words to be ignored.

The plugin API enables you to provide a full-text parser of your own so that you have control over the basic duties of a parser. A parser plugin can operate in either of two roles:

- The plugin can replace the built-in parser. In this role, the plugin reads the input to be parsed, splits it up into words, and passes the words to the server (either for indexing or for word accumulation).

  One reason to use a parser this way is that you need to use different rules from those of the built-in parser for determining how to split up input into words. For example, the built-in parser considers the text “case-sensitive” to consist of two words “case” and “sensitive,” whereas an application might need to treat the text as a single word.

- The plugin can act in conjunction with the built-in parser by serving as a front end for it. In this role, the plugin extracts text from the input and passes the text to the parser, which splits up the text into words using its normal parsing rules. In particular, this parsing will be affected by the `innodb_ft_xxx` or `ft_xxx` system variables and the stopword list.

  One reason to use a parser this way is that you need to index content such as PDF documents, XML documents, or `.doc` files. The built-in parser is not intended for those types of input but a plugin can pull out the text from these input sources and pass it to the built-in parser.

It is also possible for a parser plugin to operate in both roles. That is, it could extract text from nonplaintext input (the front end role), and also parse the text into words (thus replacing the built-in parser).

A full-text plugin is associated with full-text indexes on a per-index basis. That is, when you install a parser plugin initially, that does not cause it to be used for any full-text operations. It simply becomes available. For example, a full-text parser plugin becomes available to be named in a `WITH PARSER` clause when creating individual `FULLTEXT` indexes. To create such an index at table-creation time, do this:

```
CREATE TABLE t
(
  doc CHAR(255),
  FULLTEXT INDEX (doc) WITH PARSER my_parser
) ENGINE=InnoDB;
```

Or you can add the index after the table has been created:

```
ALTER TABLE t ADD FULLTEXT INDEX (doc) WITH PARSER my_parser;
```

The only SQL change for associating the parser with the index is the `WITH PARSER` clause. Searches are specified as before, with no changes needed for queries.

When you associate a parser plugin with a `FULLTEXT` index, the plugin is required for using the index. If the parser plugin is dropped, any index associated with it becomes unusable. Any attempt to use a table for which a plugin is not available results in an error, although `DROP TABLE` is still possible.

For more information about full-text plugins, see Section 22.2.4.4, “Writing Full-Text Parser Plugins”. MySQL 5.7 supports full-text plugins with `MyISAM`. As of MySQL 5.7.3, full-text plugins are also supported with `InnoDB`.

22.2.3.3 Daemon Plugins

A daemon plugin is a simple type of plugin used for code that should be run by the server but that does not communicate with it. MySQL distributions include an example daemon plugin that writes periodic heartbeat messages to a file.

For more information about daemon plugins, see Section 22.2.4.5, “Writing Daemon Plugins”.

22.2.3.4 `INFORMATION_SCHEMA` Plugins

`INFORMATION_SCHEMA` plugins enable the creation of tables containing server metadata that are exposed to users through the `INFORMATION_SCHEMA` database. For example, `InnoDB` uses `INFORMATION_SCHEMA` plugins to provide tables that contain information about current transactions and locks.

For more information about `INFORMATION_SCHEMA` plugins, see Section 22.2.4.6, “Writing `INFORMATION_SCHEMA` Plugins”.

22.2.3.5 Semisynchronous Replication Plugins

MySQL replication is asynchronous by default. With semisynchronous replication, a commit performed on the master side blocks before returning to the session that performed the transaction until at least one slave acknowledges that it has received and logged the events for the transaction. Semisynchronous replication is implemented through complementary master and client plugins. See Section 16.3.8, “Semisynchronous Replication”.

For more information about semisynchronous replication plugins, see Section 22.2.4.7, “Writing Semisynchronous Replication Plugins”.

22.2.3.6 Audit Plugins

In MySQL 5.7, the server provides a pluggable audit interface that enables information about server operations to be reported to interested parties. Currently, audit notification occurs for these operations (although the interface is general and the server could be modified to report others):

- Write a message to the general query log (if the log is enabled)

- Write a message to the error log

- Send a query result to a client

Audit plugins may register with the audit interface to receive notification about server operations. When an auditable event occurs within the server, the server determines whether notification is needed. For each registered audit plugin, the server checks the event against those event classes in which the plugin is interested and passes the event to the plugin if there is a match.

This interface enables audit plugins to receive notifications only about operations in event classes they consider significant and to ignore others. The interface provides for categorization of operations into event classes and further division into event subclasses within each class.

When an audit plugin is notified of an auditable event, it receives a pointer to the current THD structure and a pointer to a structure that contains information about the event. The plugin can examine the event and perform whatever auditing actions are appropriate. For example, the plugin can see what statement produced a result set or was logged, the number of rows in a result, who the current user was for an operation, or the error code for failed operations.

For more information about audit plugins, see Section 22.2.4.8, “Writing Audit Plugins”.

22.2.3.7 Authentication Plugins

MySQL 5.7 supports pluggable authentication. Authentication plugins exist on both the server and client sides. Plugins on the server side implement authentication methods for use by clients when they connect to the server. A plugin on the client side communicates with a server-side plugin to provide the authentication information that it requires. A client-side plugin may interact with the user, performing tasks such as soliciting a password or other authentication credentials to be sent to the server. See Section 6.3.8, “Pluggable Authentication”.

Pluggable authentication also enables proxy user capability, in which one user takes the identity of another user. A server-side authentication plugin can return to the server the name of the user whose identity the connecting user should have. See Section 6.3.10, “Proxy Users”.

For more information about authentication plugins, see Section 22.2.4.9, “Writing Authentication Plugins”.

22.2.3.8 Password-Validation Plugins

In MySQL 5.7, the server provides an interface for writing plugins that test passwords. Such a plugin implements two capabilities:

- Rejection of too-weak passwords in statements that assign passwords (such as `CREATE USER`, `GRANT`, and `SET PASSWORD` statements), and passwords given as arguments to the `PASSWORD()` and `OLD_PASSWORD()` functions.

- Assessing the strength of potential passwords for the `VALIDATE_PASSWORD_STRENGTH()` SQL function.

For information about writing this type of plugin, see Section 22.2.4.10, “Writing Password-Validation Plugins”.

22.2.3.9 Protocol Trace Plugins

MySQL 5.7 supports the use of protocol trace plugins: client-side plugins that implement tracing of communication between a client and the server that takes place using the client/server protocol. This capability can be used in MySQL 5.7.2 and up.

For more information about protocol trace plugins, see Section 22.2.4.11, “Writing Protocol Trace Plugins”.

22.2.4 Writing Plugins

To create a plugin library, you must provide the required descriptor information that indicates what plugins the library file contains, and write the interface functions for each plugin.

Every server plugin must have a general descriptor that provides information to the plugin API, and a type-specific descriptor that provides information about the plugin interface for a given type of plugin. The structure of the general descriptor is the same for all plugin types. The structure of the type-specific descriptor varies among plugin types and is determined by the requirements of what the plugin needs to do. The server plugin interface also enables plugins to expose status and system variables. These variables become visible through the `SHOW STATUS` and `SHOW VARIABLES` statements and the corresponding `INFORMATION_SCHEMA` tables.

For client-side plugins, the architecture is a bit different. Each plugin must have a descriptor, but there is no division into separate general and type-specific descriptors. Instead, the descriptor begins with a fixed set of members common to all client plugin types, and the common members are followed by any additional members required to implement the specific plugin type.

You can write plugins in C or C++ (or another language that can use C calling conventions). Plugins are loaded and unloaded dynamically, so your operating system must support dynamic loading and you must have compiled the calling application dynamically (not statically). For server plugins, this means that `mysqld` must be compiled dynamically.

A server plugin contains code that becomes part of the running server, so when you write the plugin, you are bound by any and all constraints that otherwise apply to writing server code. For example, you may have problems if you attempt to use functions from the `libstdc++` library. These constraints may change in future versions of the server, so it is possible that server upgrades will require revisions to plugins originally written for older servers. For information about these constraints, see Section 2.8.4, “MySQL Source-Configuration Options”, and Section 2.8.5, “Dealing with Problems Compiling MySQL”.

Client plugin writers should avoid dependencies on what symbols the calling application has because you cannot be sure what applications will use the plugin.

22.2.4.1 Overview of Plugin Writing

The following procedure provides an overview of the steps needed to create a plugin library. The next sections provide additional details on setting plugin data structures and writing specific types of plugins.

1. In the plugin source file, include the header files that the plugin library needs. The `plugin.h` file is required, and the library might require other files as well. For example:

```
#include <stdlib.h>
#include <ctype.h>
#include <mysql/plugin.h>
```

2. Set up the descriptor information for the plugin library file. For server plugins, write the library descriptor, which must contain the general plugin descriptor for each server plugin in the file. For more information, see Server Plugin Library and Plugin Descriptors. In addition, set up the type-specific descriptor for each server plugin in the library. Each plugin's general descriptor points to its type-specific descriptor.

   For client plugins, write the client descriptor. For more information, see Client Plugin Descriptors.

3. Write the plugin interface functions for each plugin. For example, each plugin's general plugin descriptor points to the initialization and deinitialization functions that the server should invoke when it loads and unloads the plugin. The plugin's type-specific description may also point to interface functions.

4. For server plugins, set up the status and system variables, if there are any.
5. Compile the plugin library as a shared library and install it in the plugin directory. For more information, see Section 22.2.4.3, “Compiling and Installing Plugin Libraries”.
6. For server plugins, register the plugin with the server. For more information, see Section 5.1.8.1, “Installing and Uninstalling Plugins”.
7. Test the plugin to verify that it works properly.

22.2.4.2 Plugin Data Structures

A plugin library file includes descriptor information to indicate what plugins it contains.

If the plugin library contains any server plugins, it must include the following descriptor information:

- A library descriptor indicates the general server plugin API version number used by the library and contains a general plugin descriptor for each server plugin in the library. To provide the framework for this descriptor, invoke two macros from the `plugin.h` header file:

  ```
  mysql_declare_plugin(name)
   ... one or more server plugin descriptors here ...
  mysql_declare_plugin_end;
  ```

  The macros expand to provide a declaration for the API version automatically. You must provide the plugin descriptors.

- Within the library descriptor, each general server plugin is described by a `st_mysql_plugin` structure. This plugin descriptor structure contains information that is common to every type of server plugin: A value that indicates the plugin type; the plugin name, author, description, and license type; pointers to the initialization and deinitialization functions that the server invokes when it loads and unloads the plugin, and pointers to any status or system variables the plugin implements.

- Each general server plugin descriptor within the library descriptor also contains a pointer to a type-specific plugin descriptor. The structure of the type-specific descriptors varies from one plugin type to another because each type of plugin can have its own API. A type-specific plugin descriptor contains a type-specific API version number and pointers to the functions that are needed to implement that plugin type. For example, a full-text parser plugin has initialization and deinitialization functions, and a main parsing function. The server invokes these functions when it uses the plugin to parse text.

The plugin library also contains the interface functions that are referenced by the general and type-specific descriptors for each plugin in the library.

If the plugin library contains a client plugin, it must include a descriptor for the plugin. The descriptor begins with a fixed set of members common to all client plugins, followed by any members specific to the plugin type. To provide the descriptor framework, invoke two macros from the `client_plugin.h` header file:

```
mysql_declare_client_plugin(plugin_type)
   ... members common to all client plugins ...
   ... type-specific extra members ...
mysql_end_client_plugin;
```

The plugin library also contains any interface functions referenced by the client descriptor.

The `mysql_declare_plugin()` and `mysql_declare_client_plugin()` macros differ somewhat in how they can be invoked, which has implications for the contents of plugin libraries. The following guidelines summarize the rules:

- mysql_declare_plugin() and mysql_declare_client_plugin() can both be used in the same source file, which means that a plugin library can contain both server and client plugins. However, each of mysql_declare_plugin() and mysql_declare_client_plugin() can be used at most once.
- mysql_declare_plugin() permits multiple server plugin declarations, so a plugin library can contain multiple server plugins.
- mysql_declare_client_plugin() permits only a single client plugin declaration. To create multiple client plugins, separate plugin libraries must be used.

When a client program looks for a client plugin that is in a plugin library and not built into libmysqlclient, it looks for a file with a basename that is the same as the plugin name. For example, if a program needs to use a client authentication plugin named auth_xxx on a system that uses .so as the library suffix, it looks in the file named auth_xxx.so. (On Mac OS X, the program looks first for auth_xxx.dylib, then for auth_xxx.so.) For this reason, if a plugin library contains a client plugin, the library must have the same basename as that plugin.

The same is not true for a library that contains server plugins. The --plugin-load option and the INSTALL PLUGIN statement provide the library file name explicitly, so there need be no explicit relationship between the library name and the name of any server plugins it contains.

Server Plugin Library and Plugin Descriptors

Every plugin library that contains server plugins must include a library descriptor that contains the general plugin descriptor for each server plugin in the file. This section discusses how to write the library and general descriptors for server plugins.

The library descriptor must define two symbols:

- _mysql_plugin_interface_version_ specifies the version number of the general plugin framework. This is given by the MYSQL_PLUGIN_INTERFACE_VERSION symbol, which is defined in the plugin.h file.
- _mysql_plugin_declarations_ defines an array of plugin declarations, terminated by a declaration with all members set to 0. Each declaration is an instance of the st_mysql_plugin structure (also defined in plugin.h). There must be one of these for each server plugin in the library.

If the server does not find those two symbols in a library, it does not accept it as a legal plugin library and rejects it with an error. This prevents use of a library for plugin purposes unless it was built specifically as a plugin library.

The conventional way to define the two required symbols is by using the mysql_declare_plugin() and mysql_declare_plugin_end macros from the plugin.h file:

```
mysql_declare_plugin(name)
 ... one or more server plugin descriptors here ...
mysql_declare_plugin_end;
```

Each server plugin must have a general descriptor that provides information to the server plugin API. The general descriptor has the same structure for all plugin types. The st_mysql_plugin structure in the plugin.h file defines this descriptor:

```
struct st_mysql_plugin
{
  int type;             /* the plugin type (a MYSQL_XXX_PLUGIN value)   */
  void *info;           /* pointer to type-specific plugin descriptor   */
  const char *name;     /* plugin name                                 */
  const char *author;   /* plugin author (for I_S.PLUGINS)             */
  const char *descr;    /* general descriptive text (for I_S.PLUGINS)   */
  int license;          /* the plugin license (PLUGIN_LICENSE_XXX)      */
  int (*init)(void *);  /* the function to invoke when plugin is loaded */
  int (*deinit)(void *);/* the function to invoke when plugin is unloaded */
  unsigned int version; /* plugin version (for I_S.PLUGINS)             */
  struct st_mysql_show_var *status_vars;
  struct st_mysql_sys_var **system_vars;
  void * __reserved1;   /* reserved for dependency checking             */
  unsigned long flags;  /* flags for plugin */
};
```

The `st_mysql_plugin` descriptor structure members are used as follows. `char *` members should be specified as null-terminated strings.

- `type`: The plugin type. This must be one of the plugin-type values from `plugin.h`:

  ```
  /*
    The allowable types of plugins
  */
  #define MYSQL_UDF_PLUGIN             0  /* User-defined function        */
  #define MYSQL_STORAGE_ENGINE_PLUGIN  1  /* Storage Engine               */
  #define MYSQL_FTPARSER_PLUGIN        2  /* Full-text parser plugin      */
  #define MYSQL_DAEMON_PLUGIN          3  /* The daemon/raw plugin type */
  #define MYSQL_INFORMATION_SCHEMA_PLUGIN  4  /* The I_S plugin type */
  #define MYSQL_AUDIT_PLUGIN           5  /* The Audit plugin type        */
  #define MYSQL_REPLICATION_PLUGIN     6  /* The replication plugin type */
  #define MYSQL_AUTHENTICATION_PLUGIN  7  /* The authentication plugin type */
  ...
  ```

  For example, for a full-text parser plugin, the `type` value is `MYSQL_FTPARSER_PLUGIN`.

- `info`: A pointer to the type-specific descriptor for the plugin. This descriptor's structure depends on the particular type of plugin, unlike that of the general plugin descriptor structure. For version-control purposes, the first member of the type-specific descriptor for every plugin type is expected to be the interface version for the type. This enables the server to check the type-specific version for every plugin no matter its type. Following the version number, the descriptor includes any other members needed, such as callback functions and other information needed by the server to invoke the plugin properly. Later sections on writing particular types of server plugins describe the structure of their type-specific descriptors.

- `name`: A string that gives the plugin name. This is the name that will be listed in the `mysql.plugin` table and by which you refer to the plugin in SQL statements such as `INSTALL PLUGIN` and `UNINSTALL PLUGIN`, or with the `--plugin-load` option. The name is also visible in the `INFORMATION_SCHEMA.PLUGINS` table or the output from `SHOW PLUGINS`.

  The plugin name should not begin with the name of any server option. If it does, the server will fail to initialize it. For example, the server has a `--socket` option, so you should not use a plugin name such as `socket`, `socket_plugin`, and so forth.

- `author`: A string naming the plugin author. This can be whatever you like.

- `desc`: A string that provides a general description of the plugin. This can be whatever you like.

- `license`: The plugin license type. The value can be one of `PLUGIN_LICENSE_PROPRIETARY`, `PLUGIN_LICENSE_GPL`, or `PLUGIN_LICENSE_BSD`.

- `init`: A once-only initialization function, or `NULL` if there is no such function. The server executes this function when it loads the plugin, which happens for `INSTALL PLUGIN` or, for plugins listed in the mysql.plugin table, at server startup. The function takes one argument that points to the internal structure used to identify the plugin. It returns zero for success and nonzero for failure.

- deinit: A once-only deinitialization function, or NULL if there is no such function. The server executes this function when it unloads the plugin, which happens for UNINSTALL PLUGIN or, for plugins listed in the mysql.plugin table, at server shutdown. The function takes one argument that points to the internal structure used to identify the plugin It returns zero for success and nonzero for failure.

- version: The plugin version number. When the plugin is installed, this value can be retrieved from the INFORMATION_SCHEMA.PLUGINS table. The value includes major and minor numbers. If you write the value as a hex constant, the format is 0x*MMNN*, where *MM* and NN are the major and minor numbers, respectively. For example, 0x0302 represents version 3.2.

- status_vars: A pointer to a structure for status variables associated with the plugin, or NULL if there are no such variables. When the plugin is installed, these variables are displayed in the output of the SHOW STATUS statement.

  The status_vars member, if not NULL, points to an array of st_mysql_show_var structures that describe status variables. See Server Plugin Status and System Variables.

- system_vars: A pointer to a structure for system variables associated with the plugin, or NULL if there are no such variables. These options and system variables can be used to help initialize variables within the plugin.

  The system_vars member, if not NULL, points to an array of st_mysql_sys_var structures that describe system variables. See Server Plugin Status and System Variables.

- __reserved1: A placeholder for the future. Currently, it should be set to NULL.

- flags: Plugin flags. Individual bits correspond to different flags. The value should be set to the OR of the applicable flags. These flags are available:

  ```
  #define PLUGIN_OPT_NO_INSTALL   1UL   /* Not dynamically loadable */
  #define PLUGIN_OPT_NO_UNINSTALL 2UL   /* Not dynamically unloadable */
  ```

  PLUGIN_OPT_NO_INSTALL indicates that the plugin cannot be loaded at runtime with the INSTALL PLUGIN statement. This is appropriate for plugins that must be loaded at server startup with the --plugin-load option. PLUGIN_OPT_NO_UNINSTALL indicates that the plugin cannot be unloaded at runtime with the UNINSTALL PLUGIN statement.

The server invokes the init and deinit functions in the general plugin descriptor only when loading and unloading the plugin. They have nothing to do with use of the plugin such as happens when an SQL statement causes the plugin to be invoked.

For example, the descriptor information for a library that contains a single full-text parser plugin named simple_parser looks like this:

```
mysql_declare_plugin(ftexample)
{
  MYSQL_FTPARSER_PLUGIN,      /* type                            */
  &simple_parser_descriptor,  /* descriptor                      */
  "simple_parser",            /* name                            */
  "Oracle Corporation",       /* author                          */
  "Simple Full-Text Parser",  /* description                     */
  PLUGIN_LICENSE_GPL,         /* plugin license                  */
  simple_parser_plugin_init,  /* init function (when loaded)     */
  simple_parser_plugin_deinit,/* deinit function (when unloaded) */
  0x0001,                     /* version                         */
```

```
  simple_status,              /* status variables                */
  simple_system_variables,    /* system variables                */
  NULL,
  0
}
mysql_declare_plugin_end;
```

For a full-text parser plugin, the type must be MYSQL_FTPARSER_PLUGIN. This is the value that identifies the plugin as being legal for use in a WITH PARSER clause when creating a FULLTEXT index. (No other plugin type is legal for this clause.)

plugin.h defines the mysql_declare_plugin() and mysql_declare_plugin_end macros like this:

```
#ifndef MYSQL_DYNAMIC_PLUGIN
#define __MYSQL_DECLARE_PLUGIN(NAME, VERSION, PSIZE, DECLS) \
MYSQL_PLUGIN_EXPORT int VERSION= MYSQL_PLUGIN_INTERFACE_VERSION; \
MYSQL_PLUGIN_EXPORT int PSIZE= sizeof(struct st_mysql_plugin); \
MYSQL_PLUGIN_EXPORT struct st_mysql_plugin DECLS[]= {
#else
#define __MYSQL_DECLARE_PLUGIN(NAME, VERSION, PSIZE, DECLS) \
MYSQL_PLUGIN_EXPORT int _mysql_plugin_interface_version_= MYSQL_PLUGIN_INTERFACE_VERSION; \
MYSQL_PLUGIN_EXPORT int _mysql_sizeof_struct_st_plugin_= sizeof(struct st_mysql_plugin); \
MYSQL_PLUGIN_EXPORT struct st_mysql_plugin _mysql_plugin_declarations_[]= {
#endif

#define mysql_declare_plugin(NAME) \
__MYSQL_DECLARE_PLUGIN(NAME, \
                 builtin_ ## NAME ## _plugin_interface_version, \
                 builtin_ ## NAME ## _sizeof_struct_st_plugin, \
                 builtin_ ## NAME ## _plugin)

#define mysql_declare_plugin_end ,{0,0,0,0,0,0,0,0,0,0,0,0,0}}
```

Note

Those declarations define the _mysql_plugin_interface_version_ symbol only if the MYSQL_DYNAMIC_PLUGIN symbol is defined. This means that -DMYSQL_DYNAMIC_PLUGIN must be provided as part of the compilation command to build the plugin as a shared library.

When the macros are used as just shown, they expand to the following code, which defines both of the required symbols (_mysql_plugin_interface_version_ and _mysql_plugin_declarations_):

```
int _mysql_plugin_interface_version_= MYSQL_PLUGIN_INTERFACE_VERSION;
int _mysql_sizeof_struct_st_plugin_= sizeof(struct st_mysql_plugin);
struct st_mysql_plugin _mysql_plugin_declarations_[]= {
{
  MYSQL_FTPARSER_PLUGIN,      /* type                            */
  &simple_parser_descriptor,  /* descriptor                      */
  "simple_parser",            /* name                            */
  "Oracle Corporation",       /* author                          */
  "Simple Full-Text Parser",  /* description                     */
  PLUGIN_LICENSE_GPL,         /* plugin license                  */
  simple_parser_plugin_init,  /* init function (when loaded)     */
  simple_parser_plugin_deinit,/* deinit function (when unloaded) */
  0x0001,                     /* version                         */
  simple_status,              /* status variables                */
  simple_system_variables,    /* system variables                */
  NULL,
  0
}
```

```
  ,{0,0,0,0,0,0,0,0,0,0,0,0}}
};
```

The preceding example declares a single plugin in the general descriptor, but it is possible to declare multiple plugins. List the declarations one after the other between `mysql_declare_plugin()` and `mysql_declare_plugin_end`, separated by commas.

MySQL server plugins can be written in C or C++ (or another language that can use C calling conventions). If you write a C++ plugin, one C++ feature that you should not use is nonconstant variables to initialize global structures. Members of structures such as the `st_mysql_plugin` structure should be initialized only with constant variables. The `simple_parser` descriptor shown earlier is permissible in a C++ plugin because it satisfies that requirement:

```
mysql_declare_plugin(ftexample)
{
  MYSQL_FTPARSER_PLUGIN,      /* type                            */
  &simple_parser_descriptor,  /* descriptor                      */
  "simple_parser",            /* name                            */
  "Oracle Corporation",       /* author                          */
  "Simple Full-Text Parser",  /* description                     */
  PLUGIN_LICENSE_GPL,         /* plugin license                  */
  simple_parser_plugin_init,  /* init function (when loaded)     */
  simple_parser_plugin_deinit,/* deinit function (when unloaded) */
  0x0001,                     /* version                         */
  simple_status,              /* status variables                */
  simple_system_variables,    /* system variables                */
  NULL,
  0
}
mysql_declare_plugin_end;
```

Here is another valid way to write the general descriptor. It uses constant variables to indicate the plugin name, author, and description:

```
const char *simple_parser_name = "simple_parser";
const char *simple_parser_author = "Oracle Corporation";
const char *simple_parser_description = "Simple Full-Text Parser";

mysql_declare_plugin(ftexample)
{
  MYSQL_FTPARSER_PLUGIN,      /* type                            */
  &simple_parser_descriptor,  /* descriptor                      */
  simple_parser_name,         /* name                            */
  simple_parser_author,       /* author                          */
  simple_parser_description,  /* description                     */
  PLUGIN_LICENSE_GPL,         /* plugin license                  */
  simple_parser_plugin_init,  /* init function (when loaded)     */
  simple_parser_plugin_deinit,/* deinit function (when unloaded) */
  0x0001,                     /* version                         */
  simple_status,              /* status variables                */
  simple_system_variables,    /* system variables                */
  NULL,
  0
}
mysql_declare_plugin_end;
```

However, the following general descriptor is invalid. It uses structure members to indicate the plugin name, author, and description, but structures are not considered constant initializers in C++:

```
typedef struct
{
  const char *name;
  const char *author;
  const char *description;
} plugin_info;

plugin_info parser_info = {
  "simple_parser",
  "Oracle Corporation",
  "Simple Full-Text Parser"
};

mysql_declare_plugin(ftexample)
{
  MYSQL_FTPARSER_PLUGIN,      /* type                            */
  &simple_parser_descriptor,  /* descriptor                      */
  parser_info.name,           /* name                            */
  parser_info.author,         /* author                          */
  parser_info.description,    /* description                     */
  PLUGIN_LICENSE_GPL,         /* plugin license                  */
  simple_parser_plugin_init,  /* init function (when loaded)     */
  simple_parser_plugin_deinit,/* deinit function (when unloaded) */
  0x0001,                     /* version                         */
  simple_status,              /* status variables                */
  simple_system_variables,    /* system variables                */
  NULL,
  0
}
mysql_declare_plugin_end;
```

Server Plugin Status and System Variables

The server plugin interface enables plugins to expose status and system variables using the status_vars and system_vars members of the general plugin descriptor.

The status_vars member of the general plugin descriptor, if not 0, points to an array of st_mysql_show_var structures, each of which describes one status variable, followed by a structure with all members set to 0. The st_mysql_show_var structure has this definition:

```
struct st_mysql_show_var {
  const char *name;
  char *value;
  enum enum_mysql_show_type type;
};
```

When the plugin is installed, the plugin name and the name value are joined with an underscore to form the name displayed by SHOW STATUS.

The following table shows the permissible status variable type values and what the corresponding variable should be.

Table 22.1 Server Plugin Status Variable Types

Variable Type	Meaning
SHOW_BOOL	Pointer to a boolean variable
SHOW_INT	Pointer to an integer variable
SHOW_LONG	Pointer to a long integer variable
SHOW_LONGLONG	Pointer to a longlong integer variable
SHOW_CHAR	A string
SHOW_CHAR_PTR	Pointer to a string
SHOW_ARRAY	Pointer to another st_mysql_show_var array

Variable Type	Meaning
SHOW_FUNC	Pointer to a function
SHOW_DOUBLE	Pointer to a double

For the SHOW_FUNC type, the function is called and fills in its out parameter, which then provides information about the variable to be displayed. The function has this signature:

```
#define SHOW_VAR_FUNC_BUFF_SIZE 1024

typedef int (*mysql_show_var_func) (void *thd,
                                    struct st_mysql_show_var *out,
                                    char *buf);
```

The system_vars member, if not 0, points to an array of st_mysql_sys_var structures, each of which describes one system variable (which can also be set from the command-line or configuration file), followed by a structure with all members set to 0. The st_mysql_sys_var structure is defined as follows:

```
struct st_mysql_sys_var {
 int flags;
 const char *name, *comment;
 int (*check)(THD*, struct st_mysql_sys_var *, void*, st_mysql_value*);
 void (*update)(THD*, struct st_mysql_sys_var *, void*, const void*);
};
```

Additional fields are append as required depending upon the flags.

For convenience, a number of macros are defined that make creating new system variables within a plugin much simpler.

Throughout the macros, the following fields are available:

- name: An unquoted identifier for the system variable.
- varname: The identifier for the static variable. Where not available, it is the same as the name field.
- opt: Additional use flags for the system variable. The following table shows the permissible flags.

  Table 22.2 Server Plugin System Variable Flags

Flag Value	Description
PLUGIN_VAR_READONLY	The system variable is read only
PLUGIN_VAR_NOSYSVAR	The system variable is not user visible at runtime
PLUGIN_VAR_NOCMDOPT	The system variable is not configurable from the command line
PLUGIN_VAR_NOCMDARG	No argument is required at the command line (typically used for boolean variables)
PLUGIN_VAR_RQCMDARG	An argument is required at the command line (this is the default)
PLUGIN_VAR_OPCMDARG	An argument is optional at the command line
PLUGIN_VAR_MEMALLOC	Used for string variables; indicates that memory is to be allocated for storage of the string

- comment: A descriptive comment to be displayed in the server help message. NULL if this variable is to be hidden.
- check: The check function, NULL for default.

- `update`: The update function, `NULL` for default.
- `default`: The variable default value.
- `minimum`: The variable minimum value.
- `maximum`: The variable maximum value.
- `blocksize`: The variable block size. When the value is set, it is rounded to the nearest multiple of `blocksize`.

A system variable may be accessed either by using the static variable directly or by using the `SYSVAR()`accessor macro. The `SYSVAR()` macro is provided for completeness. Usually it should be used only when the code cannot directly access the underlying variable.

For example:

```
static int my_foo;
static MYSQL_SYSVAR_INT(foo_var, my_foo,
                        PLUGIN_VAR_RQCMDARG, "foo comment",
                        NULL, NULL, 0, 0, INT_MAX, 0);
 ...
   SYSVAR(foo_var)= value;
   value= SYSVAR(foo_var);
   my_foo= value;
   value= my_foo;
```

Session variables may be accessed only through the `THDVAR()` accessor macro. For example:

```
static MYSQL_THDVAR_BOOL(some_flag,
                         PLUGIN_VAR_NOCMDARG, "flag comment",
                         NULL, NULL, FALSE);
 ...
   if (THDVAR(thd, some_flag))
   {
     do_something();
     THDVAR(thd, some_flag)= FALSE;
   }
```

All global and session system variables must be published to `mysqld` before use. This is done by constructing a `NULL`-terminated array of the variables and linking to it in the plugin public interface. For example:

```
static struct st_mysql_sys_var *my_plugin_vars[]= {
  MYSQL_SYSVAR(foo_var),
  MYSQL_SYSVAR(some_flag),
  NULL
};
mysql_declare_plugin(fooplug)
{
  MYSQL_..._PLUGIN,
  &plugin_data,
  "fooplug",
  "foo author",
  "This does foo!",
  PLUGIN_LICENSE_GPL,
  foo_init,
  foo_fini,
  0x0001,
  NULL,
  my_plugin_vars,
  NULL,
  0
}
mysql_declare_plugin_end;
```

The following convenience macros enable you to declare different types of system variables:

- Boolean system variables of type `my_bool`, which is a 1-byte boolean. (0 = FALSE, 1 = TRUE)

  ```
  MYSQL_THDVAR_BOOL(name, opt, comment, check, update, default)
  MYSQL_SYSVAR_BOOL(name, varname, opt, comment, check, update, default)
  ```

- String system variables of type `char*`, which is a pointer to a null-terminated string.

  ```
  MYSQL_THDVAR_STR(name, opt, comment, check, update, default)
  MYSQL_SYSVAR_STR(name, varname, opt, comment, check, update, default)
  ```

- Integer system variables, of which there are several varieties.

  - An `int` system variable, which is typically a 4-byte signed word.

    ```
    MYSQL_THDVAR_INT(name, opt, comment, check, update, default, min, max, blk)
    MYSQL_SYSVAR_INT(name, varname, opt, comment, check, update, default,
                   minimum, maximum, blocksize)
    ```

  - An `unsigned int` system variable, which is typically a 4-byte unsigned word.

    ```
    MYSQL_THDVAR_UINT(name, opt, comment, check, update, default, min, max, blk)
    MYSQL_SYSVAR_UINT(name, varname, opt, comment, check, update, default,
                    minimum, maximum, blocksize)
    ```

  - A `long` system variable, which is typically either a 4- or 8-byte signed word.

    ```
    MYSQL_THDVAR_LONG(name, opt, comment, check, update, default, min, max, blk)
    MYSQL_SYSVAR_LONG(name, varname, opt, comment, check, update, default,
                    minimum, maximum, blocksize)
    ```

  - An `unsigned long` system variable, which is typically either a 4- or 8-byte unsigned word.

    ```
    MYSQL_THDVAR_ULONG(name, opt, comment, check, update, default, min, max, blk)
    MYSQL_SYSVAR_ULONG(name, varname, opt, comment, check, update, default,
                     minimum, maximum, blocksize)
    ```

  - A `long long` system variable, which is typically an 8-byte signed word.

    ```
    MYSQL_THDVAR_LONGLONG(name, opt, comment, check, update,
                        default, minimum, maximum, blocksize)
    MYSQL_SYSVAR_LONGLONG(name, varname, opt, comment, check, update,
                        default, minimum, maximum, blocksize)
    ```

  - An `unsigned long long` system variable, which is typically an 8-byte unsigned word.

    ```
    MYSQL_THDVAR_ULONGLONG(name, opt, comment, check, update,
                         default, minimum, maximum, blocksize)
    MYSQL_SYSVAR_ULONGLONG(name, varname, opt, comment, check, update,
                         default, minimum, maximum, blocksize)
    ```

- A double system variable, which is typically an 8-byte signed word. These accessor macros were added in MySQL 5.7.2.

```
MYSQL_THDVAR_DOUBLE(name, opt, comment, check, update,
                     default, minimum, maximum, blocksize)
MYSQL_SYSVAR_DOUBLE(name, varname, opt, comment, check, update,
                     default, minimum, maximum, blocksize)
```

- An unsigned long system variable, which is typically either a 4- or 8-byte unsigned word. The range of possible values is an ordinal of the number of elements in the typelib, starting from 0.

```
MYSQL_THDVAR_ENUM(name, opt, comment, check, update, default, typelib)
MYSQL_SYSVAR_ENUM(name, varname, opt, comment, check, update,
                default, typelib)
```

- An unsigned long long system variable, which is typically an 8-byte unsigned word. Each bit represents an element in the typelib.

```
MYSQL_THDVAR_SET(name, opt, comment, check, update, default, typelib)
MYSQL_SYSVAR_SET(name, varname, opt, comment, check, update,
               default, typelib)
```

Internally, all mutable and plugin system variables are stored in a HASH structure.

Display of the server command-line help text is handled by compiling a DYNAMIC_ARRAY of all variables relevant to command-line options, sorting them, and then iterating through them to display each option.

When a command-line option has been handled, it is then removed from the argv by the handle_option() function (my_getopt.c); in effect, it is consumed.

The server processes command-line options during the plugin installation process, immediately after the plugin has been successfully loaded but before the plugin initialization function has been called

Plugins loaded at runtime do not benefit from any configuration options and must have usable defaults. Once they are installed, they are loaded at mysqld initialization time and configuration options can be set at the command line or within my.cnf.

Plugins should consider the thd parameter to be read only.

Client Plugin Descriptors

Each client plugin must have a descriptor that provides information to the client plugin API. The descriptor structure begins with a fixed set of members common to all client plugins, followed by any members specific to the plugin type.

The st_mysql_client_plugin structure in the client_plugin.h file defines a “generic” descriptor that contains the common members:

```
struct st_mysql_client_plugin
{
  int type;
  unsigned int interface_version;
  const char *name;
  const char *author;
  const char *desc;
  unsigned int version[3];
  const char *license;
  void *mysql_api;
  int (*init)(char *, size_t, int, va_list);
  int (*deinit)();
  int (*options)(const char *option, const void *);
};
```

The common st_mysql_client_plugin descriptor structure members are used as follows. char * members should be specified as null-terminated strings.

- type: The plugin type. This must be one of the plugin-type values from client_plugin.h, such as MYSQL_CLIENT_AUTHENTICATION_PLUGIN.
- interface_version: The plugin interface version. For example, this is MYSQL_CLIENT_AUTHENTICATION_PLUGIN_INTERFACE_VERSION for an authentication plugin.
- name: A string that gives the plugin name. This is the name by which you refer to the plugin when you call mysql_options() with the MYSQL_DEFAULT_AUTH option or specify the --default-auth option to a MySQL client program.
- author: A string naming the plugin author. This can be whatever you like.
- desc: A string that provides a general description of the plugin. This can be whatever you like.
- version: The plugin version as an array of three integers indicating the major, minor, and teeny versions. For example, {1,2,3} indicates version 1.2.3.
- license: A string that specifies the license type.
- mysql_api: For internal use. Specify it as NULL in the plugin descriptor.
- init: A once-only initialization function, or NULL if there is no such function. The client library executes this function when it loads the plugin. The function returns zero for success and nonzero for failure.

  The init function uses its first two arguments to return an error message if an error occurs. The first argument is a pointer to a char buffer, and the second argument indicates the buffer length. Any message returned by the init function must be null-terminated, so the maximum message length is the buffer length minus one. The next arguments are passed to mysql_load_plugin(). The first indicates how many more arguments there are (0 if none), followed by any remaining arguments.
- deinit: A once-only deinitialization function, or NULL if there is no such function. The client library executes this function when it unloads the plugin. The function takes no arguments. It returns zero for success and nonzero for failure.
- options: A function for handling options passed to the plugin, or NULL if there is no such function. The function takes two arguments representing the option name and a pointer to its value. The function returns zero for success and nonzero for failure.

For a given client plugin type, the common descriptor members may be followed by additional members necessary to implement plugins of that type. For example, the st_mysql_client_plugin_AUTHENTICATION structure for authentication plugins has a function at the end that the client library calls to perform authentication.

To declare a plugin, use the mysql_declare_client_plugin() and mysql_end_client_plugin macros:

```

mysql_declare_client_plugin(plugin_type)
   ... members common to all client plugins ...
   ... type-specific extra members ...
```

```
mysql_end_client_plugin;
```

Do not specify the `type` or `interface_version` member explicitly. The `mysql_declare_client_plugin()` macro uses the *`plugin_type`* argument to generate their values automatically. For example, declare an authentication client plugin like this:

```
mysql_declare_client_plugin(AUTHENTICATION)
  "my_auth_plugin",
  "Author Name",
  "My Client Authentication Plugin",
  {1,0,0},
  "GPL",
  NULL,
  my_auth_init,
  my_auth_deinit,
  my_auth_options,
  my_auth_main
mysql_end_client_plugin;
```

This declaration uses the `AUTHENTICATION` argument to set the `type` and `interface_version` members to `MYSQL_CLIENT_AUTHENTICATION_PLUGIN` and `MYSQL_CLIENT_AUTHENTICATION_PLUGIN_INTERFACE_VERSION`.

Depending on the plugin type, the descriptor may have other members following the common members. For example, for an authentication plugin, there is a function (`my_auth_main()` in the descriptor just shown) that handles communication with the server. See Section 22.2.4.9, “Writing Authentication Plugins”.

Normally, a client program that supports the use of authentication plugins causes a plugin to be loaded by calling `mysql_options()` to set the `MYSQL_DEFAULT_AUTH` and `MYSQL_PLUGIN_DIR` options:

```
char *plugin_dir = "path_to_plugin_dir";
char *default_auth = "plugin_name";

/* ... process command-line options ... */

mysql_options(&mysql, MYSQL_PLUGIN_DIR, plugin_dir);
mysql_options(&mysql, MYSQL_DEFAULT_AUTH, default_auth);
```

Typically, the program will also accept `--plugin-dir` and `--default-auth` options that enable users to override the default values.

Should a client program require lower-level plugin management, the client library contains functions that take an `st_mysql_client_plugin` argument. See Section 21.8.14, “C API Client Plugin Functions”.

22.2.4.3 Compiling and Installing Plugin Libraries

After your plugin is written, you must compile it and install it. The procedure for compiling shared objects varies from system to system. If you build your library using `CMake`, it should be able to generate the correct compilation commands for your system. If the library is named `somepluglib`, you should end up with a shared object file that has a name something like `somepluglib.so`. (The file name might have a different suffix on your system.)

To use `CMake`, you'll need to set up the configuration files to enable the plugin to be compiled and installed. Use the plugin examples under the `plugin` directory of a MySQL source distribution as a guide.

Create `CMakeLists.txt`, which should look something like this:

```
MYSQL_ADD_PLUGIN(somepluglib somepluglib.c
  MODULE_ONLY MODULE_OUTPUT_NAME "somepluglib")
```

When CMake generates the Makefile, it should take care of passing to the compilation command the -DMYSQL_DYNAMIC_PLUGIN flag, and passing to the linker the -lmysqlservices flag, which is needed to link in any functions from services provided through the plugin services interface. See Section 22.2.5, “MySQL Services for Plugins”.

Run CMake, then run make:

```
shell> cmake .
shell> make
```

If you need to specify configuration options to CMake, see Section 2.8.4, “MySQL Source-Configuration Options”, for a list. For example, you might want to specify CMAKE_INSTALL_PREFIX to indicate the MySQL base directory under which the plugin should be installed. You can see what value to use for this option with SHOW VARIABLES:

```
mysql> SHOW VARIABLES LIKE 'basedir';
+---------------+------------------+
| Variable_name | Value            |
+---------------+------------------+
| base          | /usr/local/mysql |
+---------------+------------------+
```

The location of the plugin directory where you should install the library is given by the plugin_dir system variable. For example:

```
mysql> SHOW VARIABLES LIKE 'plugin_dir';
+---------------+-----------------------------------+
| Variable_name | Value                             |
+---------------+-----------------------------------+
| plugin_dir    | /usr/local/mysql/lib/mysql/plugin |
+---------------+-----------------------------------+
```

To install the plugin library, use make:

```
shell> make install
```

Verify that make install installed the plugin library in the proper directory. After installing it, make sure that the library permissions permit it to be executed by the server.

22.2.4.4 Writing Full-Text Parser Plugins

MySQL 5.7 supports full-text parser plugins with MyISAM. Full-text parser plugins are supported with InnoDB as of MySQL 5.7.3. For introductory information about full-text parser plugins, see Section 22.2.3.2, “Full-Text Parser Plugins”.

A full-text parser server plugin can be used to replace or modify the built-in full-text parser. This section describes how to write a full-text parser plugin named simple_parser. This plugin performs parsing based on simpler rules than those used by the MySQL built-in full-text parser: Words are nonempty runs of whitespace characters.

The instructions use the source code in the plugin/fulltext directory of MySQL source distributions, so change location into that directory. The following procedure describes how the plugin library is created:

1. To write a full-text parser plugin, include the following header file in the plugin source file. Other MySQL or general header files might also be needed, depending on the plugin capabilities and requirements.

```
#include <mysql/plugin.h>
```

plugin.h defines the MYSQL_FTPARSER_PLUGIN server plugin type and the data structures needed to declare the plugin.

2. Set up the library descriptor for the plugin library file.

   This descriptor contains the general plugin descriptor for the server plugin. For a full-text parser plugin, the type must be MYSQL_FTPARSER_PLUGIN. This is the value that identifies the plugin as being legal for use in a WITH PARSER clause when creating a FULLTEXT index. (No other plugin type is legal for this clause.)

   For example, the library descriptor for a library that contains a single full-text parser plugin named simple_parser looks like this:

```
mysql_declare_plugin(ftexample)
{
  MYSQL_FTPARSER_PLUGIN,      /* type                            */
  &simple_parser_descriptor,  /* descriptor                      */
  "simple_parser",            /* name                            */
  "Oracle Corporation",       /* author                          */
  "Simple Full-Text Parser",  /* description                     */
  PLUGIN_LICENSE_GPL,         /* plugin license                  */
  simple_parser_plugin_init,  /* init function (when loaded)     */
  simple_parser_plugin_deinit,/* deinit function (when unloaded) */
  0x0001,                     /* version                         */
  simple_status,              /* status variables                */
  simple_system_variables,    /* system variables                */
  NULL,
  0
}
mysql_declare_plugin_end;
```

   The name member (simple_parser) indicates the name to use for references to the plugin in statements such as INSTALL PLUGIN or UNINSTALL PLUGIN. This is also the name displayed by SHOW PLUGINS or INFORMATION_SCHEMA.PLUGINS.

   For more information, see Server Plugin Library and Plugin Descriptors.

3. Set up the type-specific plugin descriptor.

   Each general plugin descriptor in the library descriptor points to a type-specific descriptor. For a full-text parser plugin, the type-specific descriptor is an instance of the st_mysql_ftparser structure in the plugin.h file:

```
struct st_mysql_ftparser
{
  int interface_version;
  int (*parse)(MYSQL_FTPARSER_PARAM *param);
  int (*init)(MYSQL_FTPARSER_PARAM *param);
  int (*deinit)(MYSQL_FTPARSER_PARAM *param);
};
```

   As shown by the structure definition, the descriptor has an interface version number and contains pointers to three functions.

   The interface version number is specified using a symbol, which is in the form: MYSQL_*xxx*_INTERFACE_VERSION. For full-text parser plugins, the symbol is

"MYSQL_FTPARSER_INTERFACE_VERSION". In the source code, you will find the actual interface version number for the full-text parser plugin defined in include/mysql/plugin_ftparser.h. With the introduction of full-text parser plugin support for InnoDB, the interface version number has been incremented in MySQL 5.7.3 from 0x0100 to 0x0101.

The init and deinit members should point to a function or be set to 0 if the function is not needed. The parse member must point to the function that performs the parsing.

In the simple_parser declaration, that descriptor is indicated by &simple_parser_descriptor. The descriptor specifies the version number for the full-text plugin interface (as given by MYSQL_FTPARSER_INTERFACE_VERSION), and the plugin's parsing, initialization, and deinitialization functions:

```
static struct st_mysql_ftparser simple_parser_descriptor=
{
  MYSQL_FTPARSER_INTERFACE_VERSION, /* interface version      */
  simple_parser_parse,              /* parsing function       */
  simple_parser_init,               /* parser init function   */
  simple_parser_deinit              /* parser deinit function */
};
```

A full-text parser plugin is used in two different contexts, indexing and searching. In both contexts, the server calls the initialization and deinitialization functions at the beginning and end of processing each SQL statement that causes the plugin to be invoked. However, during statement processing, the server calls the main parsing function in context-specific fashion:

- For indexing, the server calls the parser for each column value to be indexed.

- For searching, the server calls the parser to parse the search string. The parser might also be called for rows processed by the statement. In natural language mode, there is no need for the server to call the parser. For boolean mode phrase searches or natural language searches with query expansion, the parser is used to parse column values for information that is not in the index. Also, if a boolean mode search is done for a column that has no FULLTEXT index, the built-in parser will be called. (Plugins are associated with specific indexes. If there is no index, no plugin is used.)

The plugin declaration in the general plugin descriptor has init and deinit members that point initialization and deinitialization functions, and so does the type-specific plugin descriptor to which it points. However, these pairs of functions have different purposes and are invoked for different reasons:

- For the plugin declaration in the general plugin descriptor, the initialization and deinitialization functions are invoked when the plugin is loaded and unloaded.

- For the type-specific plugin descriptor, the initialization and deinitialization functions are invoked per SQL statement for which the plugin is used.

Each interface function named in the plugin descriptor should return zero for success or nonzero for failure, and each of them receives an argument that points to a MYSQL_FTPARSER_PARAM structure containing the parsing context. The structure has this definition:

```
typedef struct st_mysql_ftparser_param
{
  int (*mysql_parse)(struct st_mysql_ftparser_param *,
                     char *doc, int doc_len);
  int (*mysql_add_word)(struct st_mysql_ftparser_param *,
                        char *word, int word_len,
                        MYSQL_FTPARSER_BOOLEAN_INFO *boolean_info);
  void *ftparser_state;
  void *mysql_ftparam;
  struct charset_info_st *cs;
  char *doc;
  int length;
  int flags;
  enum enum_ftparser_mode mode;
} MYSQL_FTPARSER_PARAM;
```

The structure members are used as follows:

- `mysql_parse`: A pointer to a callback function that invokes the server's built-in parser. Use this callback when the plugin acts as a front end to the built-in parser. That is, when the plugin parsing function is called, it should process the input to extract the text and pass the text to the `mysql_parse` callback.

  The first parameter for this callback function should be the `param` value itself:

  ```
  param->mysql_parse(param, ...);
  ```

  A front end plugin can extract text and pass it all at once to the built-in parser, or it can extract and pass text to the built-in parser a piece at a time. However, in this case, the built-in parser treats the pieces of text as though there are implicit word breaks between them.

- `mysql_add_word`: A pointer to a callback function that adds a word to a full-text index or to the list of search terms. Use this callback when the parser plugin replaces the built-in parser. That is, when the plugin parsing function is called, it should parse the input into words and invoke the `mysql_add_word` callback for each word.

  The first parameter for this callback function should be the `param` value itself:

  ```
  param->mysql_add_word(param, ...);
  ```

- `ftparser_state`: This is a generic pointer. The plugin can set it to point to information to be used internally for its own purposes.

- `mysql_ftparam`: This is set by the server. It is passed as the first argument to the `mysql_parse` or `mysql_add_word` callback.

- `cs`: A pointer to information about the character set of the text, or 0 if no information is available.

- `doc`: A pointer to the text to be parsed.

- `length`: The length of the text to be parsed, in bytes.

- `flags`: Parser flags. This is zero if there are no special flags. Currently, the only nonzero flag is `MYSQL_FTFLAGS_NEED_COPY`, which means that `mysql_add_word()` must save a copy of the word (that is, it cannot use a pointer to the word because the word is in a buffer that will be overwritten.) This member was added in MySQL 5.1.12.

  This flag might be set or reset by MySQL before calling the parser plugin, by the parser plugin itself, or by the `mysql_parse()` function.

- `mode`: The parsing mode. This value will be one of the following constants:

  - `MYSQL_FTPARSER_SIMPLE_MODE`: Parse in fast and simple mode, which is used for indexing and for natural language queries. The parser should pass to the server only those words that should be indexed. If the parser uses length limits or a stopword list to determine which words to ignore, it should not pass such words to the server.

- MYSQL_FTPARSER_WITH_STOPWORDS: Parse in stopword mode. This is used in boolean searches for phrase matching. The parser should pass all words to the server, even stopwords or words that are outside any normal length limits.

- MYSQL_FTPARSER_FULL_BOOLEAN_INFO: Parse in boolean mode. This is used for parsing boolean query strings. The parser should recognize not only words but also boolean-mode operators and pass them to the server as tokens using the mysql_add_word callback. To tell the server what kind of token is being passed, the plugin needs to fill in a MYSQL_FTPARSER_BOOLEAN_INFO structure and pass a pointer to it.

Note

For MyISAM, the stopword list and ft_min_word_len and ft_max_word_len are checked inside the tokenizer. For InnoDB, the stopword list and equivalent word length variable settings (innodb_ft_min_token_size and innodb_ft_max_token_size) are checked outside of the tokenizer. As a result, InnoDB plugin parsers do not need to check the stopword list, innodb_ft_min_token_size, or innodb_ft_max_token_size. Instead, it is recommended that all words be returned to InnoDB. However, if you want to check stopwords within your plugin parser, use MYSQL_FTPARSER_SIMPLE_MODE, which is for full-text search index and natural language search. For MYSQL_FTPARSER_WITH_STOPWORDS and MYSQL_FTPARSER_FULL_BOOLEAN_INFO modes, it is recommended that all words be returned to InnoDB including stopwords, in case of phrase searches.

If the parser is called in boolean mode, the param->mode value will be MYSQL_FTPARSER_FULL_BOOLEAN_INFO. The MYSQL_FTPARSER_BOOLEAN_INFO structure that the parser uses for passing token information to the server looks like this:

```
typedef struct st_mysql_ftparser_boolean_info
{
  enum enum_ft_token_type type;
  int yesno;
  int weight_adjust;
  char wasign;
  char trunc;
  int position;
  /* These are parser state and must be removed. */
  char prev;
  char *quot;
} MYSQL_FTPARSER_BOOLEAN_INFO;
```

The parser should fill in the structure members as follows:

- type: The token type. The following table shows the permissible types.

Table 22.3 Full-Text Parser Token Types

Token Value	Meaning
FT_TOKEN_EOF	End of data
FT_TOKEN_WORD	A regular word
FT_TOKEN_LEFT_PAREN	The beginning of a group or subexpression

Token Value	Meaning
FT_TOKEN_RIGHT_PAREN	The end of a group or subexpression
FT_TOKEN_STOPWORD	A stopword

- yesno: Whether the word must be present for a match to occur. 0 means that the word is optional but increases the match relevance if it is present. Values larger than 0 mean that the word must be present. Values smaller than 0 mean that the word must not be present.

- weight_adjust: A weighting factor that determines how much a match for the word counts. It can be used to increase or decrease the word's importance in relevance calculations. A value of zero indicates no weight adjustment. Values greater than or less than zero mean higher or lower weight, respectively. The examples at Section 12.9.2, “Boolean Full-Text Searches”, that use the < and > operators illustrate how weighting works.

- wasign: The sign of the weighting factor. A negative value acts like the ~ boolean-search operator, which causes the word's contribution to the relevance to be negative.

- trunc: Whether matching should be done as if the boolean-mode * truncation operator had been given.

- position: Start position of the word in the document, in bytes. Used by InnoDB full-text search (FTS). The position member is new as of MySQL 5.7.3. For existing plugins that are called in boolean mode, support must be added for the position member.

Plugins should not use the prev and quot members of the MYSQL_FTPARSER_BOOLEAN_INFO structure.

Note

The boolean operator, @ distance, is not supported by the current plugin parser framework. For information about boolean full-text search operators, see Section 12.9.2, “Boolean Full-Text Searches”.

4. Set up the plugin interface functions.

   The general plugin descriptor in the library descriptor names the initialization and deinitialization functions that the server should invoke when it loads and unloads the plugin. For simple_parser, these functions do nothing but return zero to indicate that they succeeded:

```
static int simple_parser_plugin_init(void *arg __attribute__((unused)))
{
  return(0);
}

static int simple_parser_plugin_deinit(void *arg __attribute__((unused)))
{
  return(0);
}
```

   Because those functions do not actually do anything, you could omit them and specify 0 for each of them in the plugin declaration.

   The type-specific plugin descriptor for simple_parser names the initialization, deinitialization, and parsing functions that the server invokes when the plugin is used. For simple_parser, the initialization and deinitialization functions do nothing:

```
static int simple_parser_init(MYSQL_FTPARSER_PARAM *param
                              __attribute__((unused)))
{
  return(0);
}

static int simple_parser_deinit(MYSQL_FTPARSER_PARAM *param
                                __attribute__((unused)))
{
  return(0);
}
```

Here too, because those functions do nothing, you could omit them and specify 0 for each of them in the plugin descriptor.

The main parsing function, `simple_parser_parse()`, acts as a replacement for the built-in full-text parser, so it needs to split text into words and pass each word to the server. The parsing function's first argument is a pointer to a structure that contains the parsing context. This structure has a `doc` member that points to the text to be parsed, and a `length` member that indicates how long the text is. The simple parsing done by the plugin considers nonempty runs of whitespace characters to be words, so it identifies words like this:

```
static int simple_parser_parse(MYSQL_FTPARSER_PARAM *param)
{
  char *end, *start, *docend= param->doc + param->length;

  for (end= start= param->doc;; end++)
  {
    if (end == docend)
    {
      if (end > start)
        add_word(param, start, end - start);
      break;
    }
    else if (isspace(*end))
    {
      if (end > start)
        add_word(param, start, end - start);
      start= end + 1;
    }
  }
  return(0);
}
```

As the parser finds each word, it invokes a function `add_word()` to pass the word to the server. `add_word()` is a helper function only; it is not part of the plugin interface. The parser passes the parsing context pointer to `add_word()`, as well as a pointer to the word and a length value:

```
static void add_word(MYSQL_FTPARSER_PARAM *param, char *word, size_t len)
{
  MYSQL_FTPARSER_BOOLEAN_INFO bool_info=
    { FT_TOKEN_WORD, 0, 0, 0, 0, ' ', 0 };

  param->mysql_add_word(param, word, len, &bool_info);
}
```

For boolean-mode parsing, `add_word()` fills in the members of the `bool_info` structure as described earlier in the discussion of the `st_mysql_ftparser_boolean_info` structure.

5. Set up the status variables. For the simple_parser plugin, the following status variable array sets up one status variable with a value that is static text, and another with a value that is stored in a long integer variable:

   ```
   long number_of_calls= 0;

   struct st_mysql_show_var simple_status[]=
   {
     {"static", (char *)"just a static text", SHOW_CHAR},
     {"called", (char *)&number_of_calls,     SHOW_LONG},
     {0,0,0}
   };
   ```

   When the plugin is installed, the plugin name and the name value are joined with an underscore to form the name displayed by SHOW STATUS. For the array just shown, the resulting status variable names are simple_parser_static and simple_parser_called. This convention means that you can easily display the variables for a plugin using its name:

   ```
   mysql> SHOW STATUS LIKE 'simple_parser%';
   +----------------------+--------------------+
   | Variable_name        | Value              |
   +----------------------+--------------------+
   | simple_parser_static | just a static text |
   | simple_parser_called | 0                  |
   +----------------------+--------------------+
   ```

6. To compile and install a plugin library object file, use the instructions in Section 22.2.4.3, “Compiling and Installing Plugin Libraries”. To use the library file, it must be installed in the plugin directory (the directory named by the plugin_dir system variable). For the simple_parser plugin, it is compiled and installed when you build MySQL from source. It is also included in binary distributions. The build process produces a shared object library with a name of mypluglib.so (the suffix might differ depending on your platform).

7. To use the plugin, register it with the server. For example, to register the plugin at runtime, use this statement (changing the suffix as necessary):

   ```
   mysql> INSTALL PLUGIN simple_parser SONAME 'mypluglib.so';
   ```

   For additional information about plugin loading, see Section 5.1.8.1, “Installing and Uninstalling Plugins”.

8. To verify plugin installation, examine the INFORMATION_SCHEMA.PLUGINS table or use the SHOW PLUGINS statement.

9. Test the plugin to verify that it works properly.

   Create a table that contains a string column and associate the parser plugin with a FULLTEXT index on the column:

   ```
   mysql> CREATE TABLE t (c VARCHAR(255),
       ->   FULLTEXT (c) WITH PARSER simple_parser
       -> ) ENGINE=MyISAM;
   Query OK, 0 rows affected (0.01 sec)
   ```

   Insert some text into the table and try some searches. These should verify that the parser plugin treats all nonwhitespace characters as word characters:

```
mysql> INSERT INTO t VALUES
    ->   ('latin1_general_cs is a case-sensitive collation'),
    ->   ('I\'d like a case of oranges'),
    ->   ('this is sensitive information'),
    ->   ('another row'),
    ->   ('yet another row');
Query OK, 5 rows affected (0.02 sec)
Records: 5  Duplicates: 0  Warnings: 0

mysql> SELECT c FROM t;
+-------------------------------------------------+
| c                                               |
+-------------------------------------------------+
| latin1_general_cs is a case-sensitive collation |
| I'd like a case of oranges                      |
| this is sensitive information                   |
| another row                                     |
| yet another row                                 |
+-------------------------------------------------+
5 rows in set (0.00 sec)

mysql> SELECT MATCH(c) AGAINST('case') FROM t;
+--------------------------+
| MATCH(c) AGAINST('case') |
+--------------------------+
|                        0 |
|          1.2968142032623 |
|                        0 |
|                        0 |
|                        0 |
+--------------------------+
5 rows in set (0.00 sec)

mysql> SELECT MATCH(c) AGAINST('sensitive') FROM t;
+-------------------------------+
| MATCH(c) AGAINST('sensitive') |
+-------------------------------+
|                             0 |
|                             0 |
|               1.3253291845322 |
|                             0 |
|                             0 |
+-------------------------------+
5 rows in set (0.01 sec)

mysql> SELECT MATCH(c) AGAINST('case-sensitive') FROM t;
+------------------------------------+
| MATCH(c) AGAINST('case-sensitive') |
+------------------------------------+
|                    1.3109166622162 |
|                                  0 |
|                                  0 |
|                                  0 |
|                                  0 |
+------------------------------------+
5 rows in set (0.01 sec)

mysql> SELECT MATCH(c) AGAINST('I\'d') FROM t;
+--------------------------+
| MATCH(c) AGAINST('I\'d') |
+--------------------------+
|                        0 |
|          1.2968142032623 |
|                        0 |
|                        0 |
|                        0 |
+--------------------------+
5 rows in set (0.01 sec)
```

Note how neither “case” nor “insensitive” match “case-insensitive” the way that they would for the built-in parser.

22.2.4.5 Writing Daemon Plugins

A daemon plugin is a simple type of plugin used for code that should be run by the server but that does not communicate with it. This section describes how to write a daemon server plugin, using the example plugin found in the `plugin/daemon_example` directory of MySQL source distributions. That directory contains the `daemon_example.cc` source file for a daemon plugin named `daemon_example` that writes a heartbeat string at regular intervals to a file named `mysql-heartbeat.log` in the data directory.

To write a daemon plugin, include the following header file in the plugin source file. Other MySQL or general header files might also be needed, depending on the plugin capabilities and requirements.

```
#include <mysql/plugin.h>
```

`plugin.h` defines the `MYSQL_DAEMON_PLUGIN` server plugin type and the data structures needed to declare the plugin.

The `daemon_example.cc` file sets up the library descriptor as follows. The library descriptor includes a single general server plugin descriptor.

```
mysql_declare_plugin(daemon_example)
{
  MYSQL_DAEMON_PLUGIN,
  &daemon_example_plugin,
  "daemon_example",
  "Brian Aker",
  "Daemon example, creates a heartbeat beat file in mysql-heartbeat.log",
  PLUGIN_LICENSE_GPL,
  daemon_example_plugin_init, /* Plugin Init */
  daemon_example_plugin_deinit, /* Plugin Deinit */
  0x0100 /* 1.0 */,
  NULL,                       /* status variables                */
  NULL,                       /* system variables                */
  NULL,                       /* config options                  */
  0,                          /* flags                           */
}
mysql_declare_plugin_end;
```

The `name` member (`daemon_example`) indicates the name to use for references to the plugin in statements such as `INSTALL PLUGIN` or `UNINSTALL PLUGIN`. This is also the name displayed by `SHOW PLUGINS` or `INFORMATION_SCHEMA.PLUGINS`.

The second member of the plugin descriptor, `daemon_example_plugin`, points to the type-specific daemon plugin descriptor. This structure consists only of the type-specific API version number:

```
struct st_mysql_daemon daemon_example_plugin=
{ MYSQL_DAEMON_INTERFACE_VERSION  };
```

The type-specific structure has no interface functions. There is no communication between the server and the plugin, except that the server calls the initialization and deinitialization functions from the general plugin descriptor to start and stop the plugin:

- daemon_example_plugin_init() opens the heartbeat file and spawns a thread that wakes up periodically and writes the next message to the file.
- daemon_example_plugin_deinit() closes the file and performs other cleanup.

To compile and install a plugin library object file, use the instructions in Section 22.2.4.3, “Compiling and Installing Plugin Libraries”. To use the library file, it must be installed in the plugin directory (the directory named by the plugin_dir system variable). For the daemon_example plugin, it is compiled and installed when you build MySQL from source. It is also included in binary distributions. The build process produces a shared object library with a name of libdaemon_example.so (the suffix might differ depending on your platform).

To use the plugin, register it with the server. For example, to register the plugin at runtime, use this statement (change the suffix as necessary):

```
mysql> INSTALL PLUGIN daemon_example SONAME 'libdaemon_example.so';
```

For additional information about plugin loading, see Section 5.1.8.1, “Installing and Uninstalling Plugins”.

To verify plugin installation, examine the INFORMATION_SCHEMA.PLUGINS table or use the SHOW PLUGINS statement.

While the plugin is loaded, it writes a heartbeat string at regular intervals to a file named mysql-heartbeat.log in the data directory. This file grows without limit, so after you have satistifed yourself that the plugin operates correctly, unload it:

```
mysql> UNINSTALL PLUGIN daemon_example;
```

22.2.4.6 Writing INFORMATION_SCHEMA Plugins

This section describes how to write an INFORMATION_SCHEMA table server plugin. For example code that implements such plugins, see the sql/sql_show.cc file of a MySQL source distribution. You can also look at the example plugins found in the InnoDB source. See the handler/i_s.cc and handler/ha_innodb.cc files within the InnoDB source tree (in the storage/innobase directory).

To write an INFORMATION_SCHEMA table plugin, include the following header files in the plugin source file. Other MySQL or general header files might also be needed, depending on the plugin capabilities and requirements.

```
#include <sql_class.h>
#include <table.h>
```

These header files are located in the sql directory of MySQL source distributions. They contain C++ structures, so the source file for an INFORMATION_SCHEMA plugin must be compiled as C++ (not C) code.

The source file for the example plugin developed here is named simple_i_s_table.cc. It creates a simple INFORMATION_SCHEMA table named SIMPLE_I_S_TABLE that has two columns named NAME and VALUE. The general descriptor for a plugin library that implements the table looks like this:

```
mysql_declare_plugin(simple_i_s_library)
{
  MYSQL_INFORMATION_SCHEMA_PLUGIN,
  &simple_table_info,                /* type-specific descriptor */
  "SIMPLE_I_S_TABLE",                /* table name */
  "Author Name",                     /* author */
  "Simple INFORMATION_SCHEMA table", /* description */
  PLUGIN_LICENSE_GPL,                /* license type */
  simple_table_init,                 /* init function */
  NULL,
  0x0100,                            /* version = 1.0 */
  NULL,                              /* no status variables */
  NULL,                              /* no system variables */
  NULL,                              /* no reserved information */
  0                                  /* no flags */
}
mysql_declare_plugin_end;
```

The name member (SIMPLE_I_S_TABLE) indicates the name to use for references to the plugin in statements such as INSTALL PLUGIN or UNINSTALL PLUGIN. This is also the name displayed by SHOW PLUGINS or INFORMATION_SCHEMA.PLUGINS.

The simple_table_info member of the general descriptor points to the type-specific descriptor, which consists only of the type-specific API version number:

```
static struct st_mysql_information_schema simple_table_info =
{ MYSQL_INFORMATION_SCHEMA_INTERFACE_VERSION };
```

The general descriptor points to the initialization and deinitialization functions:

- The initialization function provides information about the table structure and a function that populates the table.

- The deinitialization function performs any required cleanup. If no cleanup is needed, this descriptor member can be NULL (as in the example shown).

The initialization function should return 0 for success, 1 if an error occurs. The function receives a generic pointer, which it should interpret as a pointer to the table structure:

```
static int table_init(void *ptr)
{
  ST_SCHEMA_TABLE *schema_table= (ST_SCHEMA_TABLE*)ptr;

  schema_table->fields_info= simple_table_fields;
  schema_table->fill_table= simple_fill_table;
  return 0;
}
```

The function should set these two members of the table structure:

- fields_info: An array of ST_FIELD_INFO structures that contain information about each column.

- fill_table: A function that populates the table.

The array pointed to by fields_info should contain one element per column of the INFORMATION_SCHEMA plus a terminating element. The following simple_table_fields array for the example plugin indicates that SIMPLE_I_S_TABLE has two columns. NAME is string-valued with a length of 10 and VALUE is integer-valued with a display width of 20. The last structure marks the end of the array.

```
static ST_FIELD_INFO simple_table_fields[]=
{
  {"NAME", 10, MYSQL_TYPE_STRING, 0, 0 0, 0},
  {"VALUE", 6, MYSQL_TYPE_LONG, 0, MY_I_S_UNSIGNED, 0, 0},
  {0, 0, MYSQL_TYPE_NULL, 0, 0, 0, 0}
};
```

For more information about the column information structure, see the definition of ST_FIELD_INFO in the table.h header file. The permissible MYSQL_TYPE_*xxx* type values are those used in the C API; see Section 21.8.5, “C API Data Structures”.

The fill_table member should be set to a function that populates the table and returns 0 for success, 1 if an error occurs. For the example plugin, the simple_fill_table() function looks like this:

```
static int simple_fill_table(THD *thd, TABLE_LIST *tables, Item *cond)
{
  TABLE *table= tables->table;

  table->field[0]->store("Name 1", 6, system_charset_info);
  table->field[1]->store(1);
  if (schema_table_store_record(thd, table))
    return 1;
  table->field[0]->store("Name 2", 6, system_charset_info);
  table->field[1]->store(2);
  if (schema_table_store_record(thd, table))
    return 1;
  return 0;
}
```

For each row of the INFORMATION_SCHEMA table, this function initializes each column, then calls schema_table_store_record() to install the row. The store() method arguments depend on the type of value to be stored. For column 0 (NAME, a string), store() takes a pointer to a string, its length, and information about the character set of the string:

```
store(const char *to, uint length, CHARSET_INFO *cs);
```

For column 1 (VALUE, an integer), store() takes the value and a flag indicating whether it is unsigned:

```
store(longlong nr, bool unsigned_value);
```

For other examples of how to populate INFORMATION_SCHEMA tables, search for instances of schema_table_store_record() in sql_show.cc.

To compile and install a plugin library object file, see the instructions in Section 22.2.4.3, “Compiling and Installing Plugin Libraries”. To use the library file, it must be installed in the plugin directory (the directory named by the plugin_dir system variable).

To test the plugin, install it:

```
mysql> INSTALL PLUGIN SIMPLE_I_S_TABLE SONAME 'simple_i_s_table.so';
```

Verify that the table is present:

```
mysql> SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES
    -> WHERE TABLE_NAME = 'SIMPLE_I_S_TABLE';
+------------------+
| TABLE_NAME       |
+------------------+
| SIMPLE_I_S_TABLE |
+------------------+
```

Try to select from it:

```
mysql> SELECT * FROM INFORMATION_SCHEMA.SIMPLE_I_S_TABLE;
+--------+-------+
| NAME   | VALUE |
+--------+-------+
| Name 1 |     1 |
| Name 2 |     2 |
+--------+-------+
```

Uninstall it:

```
mysql> UNINSTALL PLUGIN SIMPLE_I_S_TABLE;
```

22.2.4.7 Writing Semisynchronous Replication Plugins

This section describes how to write semisynchronous replication server plugins, using the example plugins found in the `plugin/semisync` directory of MySQL source distributions. That directory contains the source files for master and slave plugins named `rpl_semi_sync_master` and `rpl_semi_sync_slave`. The information here covers only how to set up the plugin framework. For details about how the plugins implement replication functions, see the source.

To write a semisynchronous replication plugin, include the following header file in the plugin source file. Other MySQL or general header files might also be needed, depending on the plugin capabilities and requirements.

```
#include <mysql/plugin.h>
```

`plugin.h` defines the `MYSQL_REPLICATION_PLUGIN` server plugin type and the data structures needed to declare the plugin.

For the master side, `semisync_master_plugin.cc` contains this general descriptor for a plugin named `rpl_semi_sync_master`:

```
mysql_declare_plugin(semi_sync_master)
{
  MYSQL_REPLICATION_PLUGIN,
  &semi_sync_master_plugin,
  "rpl_semi_sync_master",
  "He Zhenxing",
  "Semi-synchronous replication master",
  PLUGIN_LICENSE_GPL,
  semi_sync_master_plugin_init, /* Plugin Init */
  semi_sync_master_plugin_deinit, /* Plugin Deinit */
  0x0100 /* 1.0 */,
  semi_sync_master_status_vars, /* status variables */
  semi_sync_master_system_vars, /* system variables */
  NULL,                         /* config options */
  0,                            /* flags */
}
mysql_declare_plugin_end;
```

For the slave side, `semisync_slave_plugin.cc` contains this general descriptor for a plugin named `rpl_semi_sync_slave`:

```
mysql_declare_plugin(semi_sync_slave)
{
  MYSQL_REPLICATION_PLUGIN,
  &semi_sync_slave_plugin,
  "rpl_semi_sync_slave",
  "He Zhenxing",
  "Semi-synchronous replication slave",
  PLUGIN_LICENSE_GPL,
  semi_sync_slave_plugin_init, /* Plugin Init */
  semi_sync_slave_plugin_deinit, /* Plugin Deinit */
  0x0100 /* 1.0 */,
  semi_sync_slave_status_vars,  /* status variables */
  semi_sync_slave_system_vars,  /* system variables */
  NULL,                         /* config options */
  0,                            /* flags */
}
mysql_declare_plugin_end;
```

For both the master and slave plugins, the general descriptor has pointers to the type-specific descriptor, the initialization and deinitialization functions, and to the status and system variables implemented by the plugin. For information about variable setup, see Server Plugin Status and System Variables. The following remarks discuss the type-specific descriptor and the initialization and deinitialization functions for the master plugin but apply similarly to the slave plugin.

The semi_sync_master_plugin member of the master general descriptor points to the type-specific descriptor, which consists only of the type-specific API version number:

```
struct Mysql_replication semi_sync_master_plugin= {
  MYSQL_REPLICATION_INTERFACE_VERSION
};
```

The initialization and deinitialization function declarations look like this:

```
static int semi_sync_master_plugin_init(void *p);
static int semi_sync_master_plugin_deinit(void *p);
```

The initialization function uses the pointer to register transaction and binary logging “observers” with the server. After successful initialization, the server takes care of invoking the observers at the appropriate times. (For details on the observers, see the source files.) The deinitialization function cleans up by deregistering the observers. Each function returns 0 for success or 1 if an error occurs.

To compile and install a plugin library object file, use the instructions in Section 22.2.4.3, “Compiling and Installing Plugin Libraries”. To use the library files, they must be installed in the plugin directory (the directory named by the plugin_dir system variable). For the rpl_semi_sync_master and rpl_semi_sync_slave plugins, they are compiled and installed when you build MySQL from source. They are also included in binary distributions. The build process produces shared object libraries with names of semisync_master.so and semisync_slave.so (the suffix might differ depending on your platform).

22.2.4.8 Writing Audit Plugins

This section describes how to write an audit server plugin, using the example plugin found in the plugin/audit_null directory of MySQL source distributions. The audit_null.c source file in that directory implements a simple example audit plugin named NULL_AUDIT.

Within the server, the pluggable audit interface is implemented in the sql_audit.h and sql_audit.cc files in the sql directory of MySQL source distributions. Additionally, several places in the server are modified to call the audit interface when an auditable event occurs, so that registered audit plugins can be notified about the event if necessary. To see where such calls occur, look for invocations of functions with names of the form mysql_audit_*xxx*(). Audit notification occurs for server operations such as these:

- Writing a message to the general query log (if the log is enabled)
- Writing a message to the error log
- Sending a query result to a client
- Client connect and disconnect events

To write an audit plugin, include the following header file in the plugin source file. Other MySQL or general header files might also be needed, depending on the plugin capabilities and requirements.

```
#include <mysql/plugin_audit.h>
```

`plugin_audit.h` includes `plugin.h`, so you need not include the latter file explicitly. `plugin.h` defines the `MYSQL_AUDIT_PLUGIN` server plugin type and the data structures needed to declare the plugin. `plugin_audit.h` defines data structures specific to audit plugins.

An audit plugin, like any MySQL server plugin, has a general plugin descriptor (see Server Plugin Library and Plugin Descriptors). In `audit_null.c`, the general descriptor looks like this:

```
mysql_declare_plugin(audit_null)
{
  MYSQL_AUDIT_PLUGIN,         /* type                            */
  &audit_null_descriptor,     /* descriptor                      */
  "NULL_AUDIT",               /* name                            */
  "Oracle Corp",              /* author                          */
  "Simple NULL Audit",        /* description                     */
  PLUGIN_LICENSE_GPL,
  audit_null_plugin_init,     /* init function (when loaded)     */
  audit_null_plugin_deinit,   /* deinit function (when unloaded) */
  0x0003,                     /* version                         */
  simple_status,              /* status variables                */
  NULL,                       /* system variables                */
  NULL,
  0,
}
mysql_declare_plugin_end;
```

The `name` member (`NULL_AUDIT`) indicates the name to use for references to the plugin in statements such as `INSTALL PLUGIN` or `UNINSTALL PLUGIN`. This is also the name displayed by `INFORMATION_SCHEMA.PLUGINS` or `SHOW PLUGINS`.

The general descriptor also refers to `simple_status`, a structure that exposes several status variables to the `SHOW STATUS` statement:

```
static struct st_mysql_show_var simple_status[]=
{
  { "Audit_null_called",
    (char *) &number_of_calls,
    SHOW_INT },
  { "Audit_null_general_log",
    (char *) &number_of_calls_general_log,
    SHOW_INT },
  { "Audit_null_general_error",
    (char *) &number_of_calls_general_error,
    SHOW_INT },
  { "Audit_null_general_result",
    (char *) &number_of_calls_general_result,
    SHOW_INT },
  { "Audit_null_general_status",
    (char *) &number_of_calls_general_status,
    SHOW_INT },
  { "Audit_null_connection_connect",
    (char *) &number_of_calls_connection_connect,
    SHOW_INT },
  { "Audit_null_connection_disconnect",
    (char *) &number_of_calls_connection_disconnect,
    SHOW_INT },
  { "Audit_null_connection_change_user",
    (char *) &number_of_calls_connection_change_user,
    SHOW_INT },
  { 0, 0, 0}
};
```

The audit_null_plugin_init initialization function sets the status variables to zero when the plugin is loaded. The audit_null_plugin_deinit function performs cleanup with the plugin is unloaded. During operation, the plugin increments the first status variable for each notification it receives. It also increments the others according to the event class and subclass. In effect, the first variable is the aggregate of the counts for the event subclasses.

The audit_null_descriptor value in the general descriptor points to the type-specific descriptor. For audit plugins, this descriptor has the following structure:

```
struct st_mysql_audit
{
  int interface_version;
  void (*release_thd)(MYSQL_THD);
  void (*event_notify)(MYSQL_THD, unsigned int, const void *);
  unsigned long class_mask[MYSQL_AUDIT_CLASS_MASK_SIZE];
};
```

The type-specific descriptor has these members:

- interface_version: By convention, type-specific plugin descriptors begin with the interface version for the given plugin type. The server checks interface_version when it loads the plugin to see whether the plugin is compatible with it. For audit plugins, the value of the interface_version member is MYSQL_AUDIT_INTERFACE_VERSION (defined in plugin_audit.h).

- release_thd: A function that the server calls to inform the plugin that it is being dissociated from its thread context. This should be NULL if there is no such function.

- event_notify: A function that the server calls to notify the plugin that an auditable event has occurred. This function should not be NULL; that would not make sense because no auditing would occur.

- class_mask: A bit mask that indicates the event classes for which the plugin wants to receive notification. If this value is 0, the server passes no events to the plugin.

The server uses the event_notify and release_thd functions together. They are called within the context of a specific thread, and a thread might perform an activity that produces several event notifications. The first time the server calls event_notify for a thread, it creates a binding of the plugin to the thread. The plugin cannot be uninstalled while this binding exists. When no more events for the thread will occur, the server informs the plugin of this by calling the release_thd function, and then destroys the binding. For example, when a client issues a statement, the thread processing the statement might notify audit plugins about the result set produced by the statement and about the statement being logged. After these notifications occur, the server releases the plugin before putting the thread to sleep until the client issues another statement.

This design enables the plugin to allocate resources needed for a given thread in the first call to the event_notify function and release them in the release_thd function:

```
event_notify function:
  if memory is needed to service the thread
    allocate memory
  ... rest of notification processing ...

release_thd function:
  if memory was allocated
    release memory
  ... rest of release processing ...
```

That is more efficient than allocating and releasing memory repeatedly in the notification function.

For the NULL_AUDIT example audit plugin, the type-specific descriptor looks like this:

```
static struct st_mysql_audit audit_null_descriptor=
{
  MYSQL_AUDIT_INTERFACE_VERSION,                    /* interface version    */
  NULL,                                             /* release_thd function */
  audit_null_notify,                                /* notify function      */
  { (unsigned long) MYSQL_AUDIT_GENERAL_CLASSMASK |
                    MYSQL_AUDIT_CONNECTION_CLASSMASK } /* class mask        */
};
```

The server calls audit_null_notify to pass audit event information to the plugin. There is no release_thd function.

The event class mask indicates an interest in all events of the “general” and “connection” classes. plugin_audit.h defines symbols for these classes and their corresponding class masks:

```
#define MYSQL_AUDIT_GENERAL_CLASS 0
#define MYSQL_AUDIT_GENERAL_CLASSMASK (1 << MYSQL_AUDIT_GENERAL_CLASS)

#define MYSQL_AUDIT_CONNECTION_CLASS 1
#define MYSQL_AUDIT_CONNECTION_CLASSMASK (1 << MYSQL_AUDIT_CONNECTION_CLASS)
```

In the type-specific descriptor, the second and third parameters of the event_notify function prototype represent the event class and a generic pointer to an event structure:

```
void (*event_notify)(MYSQL_THD, unsigned int, const void *);
```

Events in different classes may have different structures, so the notification function should use the event class value to determine how to interpret the pointer to the event structure.

If the server calls the notification function with an event class of MYSQL_AUDIT_GENERAL_CLASS, it passes the event structure as a pointer to a mysql_event_general structure:

```
struct mysql_event_general
{
  unsigned int event_subclass;
  int general_error_code;
  unsigned long general_thread_id;
  const char *general_user;
  unsigned int general_user_length;
  const char *general_command;
  unsigned int general_command_length;
  const char *general_query;
  unsigned int general_query_length;
  struct charset_info_st *general_charset;
  unsigned long long general_time;
  unsigned long long general_rows;
  MYSQL_LEX_STRING general_host;
  MYSQL_LEX_STRING general_sql_command;
  MYSQL_LEX_STRING general_external_user;
  MYSQL_LEX_STRING general_ip;
};
```

Audit plugins can interpret mysql_event_general members as follows:

- event_subclass: The event subclass, one of the following values:

```
#define MYSQL_AUDIT_GENERAL_LOG 0
#define MYSQL_AUDIT_GENERAL_ERROR 1
#define MYSQL_AUDIT_GENERAL_RESULT 2
#define MYSQL_AUDIT_GENERAL_STATUS 3
```

- general_error_code: The error code. This is a value like that returned by the mysql_errno() C API function; 0 means "no error."
- general_thread_id: The ID of the thread for which the event occurred.
- general_user: The current user for the event.
- general_user_length: The length of general_user, in bytes.
- general_command: For general query log events, the type of operation. Examples: Connect, Query, Shutdown. For error log events, the error message. This is a value like that returned by the mysql_error() C API function; an empty string means "no error." For result events, this is empty.
- general_command_length: The length of general_command, in bytes.
- general_query: The SQL statement that was logged or produced a result.
- general_query_length: The length of general_query, in bytes.
- general_charset: Character set information for the event.
- general_time: A TIMESTAMP value indicating the time just before the notification function was called.
- general_rows: For general query log events, zero. For error log events, the row number at which an error occurred. For result events, the number of rows in the result plus one. For statements that produce no result set, the value is 0. This encoding enables statements that produce no result set to be distinguished from those that produce an empty result set. For example, for a DELETE statement, this value is 0. For a SELECT, the result is always 1 or more, where 1 represents an empty result set.
- general_host: For general query log events, a string representing the client host name.
- general_sql_command: For general query log events, a string that indicates the type of action performed, such as connect or drop_table.
- general_external_user: For general query log events, a string representing the external user (empty if none).
- general_ip: For general query log events, a string representing the client IP address.

The general_host, general_sql_command, general_external_user, and general_ip members are MYSQL_LEX_STRING structures that pair a string and its length. For example, if event_general is a pointer to a general event, you can access the members of the general_host value as follows:

```
event_general->general_host.length
event_general->general_host.str
```

If the server calls the notification function with an event class of MYSQL_AUDIT_CONNECTION_CLASS, it passes the event structure as a pointer to a mysql_event_connection structure, which is similar to and interpreted much the same way as the mysql_event_general structure.

The NULL_AUDIT plugin notification function is quite simple. It increments a global event counter, determines the event class, then looks at the event subclass to determine which subclass counter to increment:

```
static void audit_null_notify(MYSQL_THD thd __attribute__((unused)),
                              unsigned int event_class,
                              const void *event)
{
  /* prone to races, oh well */
  number_of_calls++;
  if (event_class == MYSQL_AUDIT_GENERAL_CLASS)
  {
    const struct mysql_event_general *event_general=
      (const struct mysql_event_general *) event;
    switch (event_general->event_subclass)
    {
    case MYSQL_AUDIT_GENERAL_LOG:
      number_of_calls_general_log++;
      break;
    case MYSQL_AUDIT_GENERAL_ERROR:
      number_of_calls_general_error++;
      break;
    case MYSQL_AUDIT_GENERAL_RESULT:
      number_of_calls_general_result++;
      break;
    case MYSQL_AUDIT_GENERAL_STATUS:
      number_of_calls_general_status++;
      break;
    default:
      break;
    }
  }
  else if (event_class == MYSQL_AUDIT_CONNECTION_CLASS)
  {
    const struct mysql_event_connection *event_connection=
      (const struct mysql_event_connection *) event;
    switch (event_connection->event_subclass)
    {
    case MYSQL_AUDIT_CONNECTION_CONNECT:
      number_of_calls_connection_connect++;
      break;
    case MYSQL_AUDIT_CONNECTION_DISCONNECT:
      number_of_calls_connection_disconnect++;
      break;
    case MYSQL_AUDIT_CONNECTION_CHANGE_USER:
      number_of_calls_connection_change_user++;
      break;
    default:
      break;
    }
  }
}
```

To compile and install a plugin library object file, use the instructions in Section 22.2.4.3, “Compiling and Installing Plugin Libraries”. To use the library file, it must be installed in the plugin directory (the directory named by the `plugin_dir` system variable). For the `AUDIT_NULL` plugin, it is compiled and installed when you build MySQL from source. It is also included in binary distributions. The build process produces a shared object library with a name of `adt_null.so` (the suffix might differ depending on your platform).

To register the plugin at runtime, use this statement (change the suffix as necessary):

```
mysql> INSTALL PLUGIN NULL_AUDIT SONAME 'adt_null.so';
```

For additional information about plugin loading, see Section 5.1.8.1, “Installing and Uninstalling Plugins”.

To verify plugin installation, examine the `INFORMATION_SCHEMA.PLUGINS` table or use the `SHOW PLUGINS` statement.

While the audit plugin is installed, it exposes status variables that indicate the events for which the plugin has been called:

```
mysql> SHOW STATUS LIKE 'Audit_null%';
+----------------------------------+-------+
| Variable_name                    | Value |
+----------------------------------+-------+
| Audit_null_called                | 1388  |
| Audit_null_connection_change_user | 0     |
| Audit_null_connection_connect    | 22    |
| Audit_null_connection_disconnect | 21    |
| Audit_null_general_error         | 1     |
| Audit_null_general_log           | 513   |
| Audit_null_general_result        | 415   |
| Audit_null_general_status        | 416   |
+----------------------------------+-------+
```

Audit_null_called counts all events, and the other variables count instances of event subclasses. For example, the preceding SHOW STATUS statement causes the server to send a result to the client and to write a message to the general query log if that log is enabled. Thus, a client that issues the statement repeatedly causes Audit_null_called and Audit_null_general_result to be incremented each time, and Audit_null_general_log to be incremented if the log is enabled.

To disable the plugin after testing it, use this statement to unload it:

```
mysql> UNINSTALL PLUGIN NULL_AUDIT;
```

22.2.4.9 Writing Authentication Plugins

MySQL supports pluggable authentication, in which plugins are invoked to authenticate client connections. Authentication plugins enable the use of authentication methods other than the built-in method of passwords stored in the mysql.user table. For example, plugins can be written to access external authentication methods. Also, authentication plugins can support the proxy user capability, such that the connecting user is a proxy for another user and is treated, for purposes of access control, as having the privileges of a different user. For more information, see Section 6.3.8, “Pluggable Authentication”, and Section 6.3.10, “Proxy Users”.

An authentication plugin can be written for the server side or the client side. Server-side plugins use the same plugin API that is used for the other server plugin types such as full-text parser or audit plugins (although with a different type-specific descriptor). Client-side plugins use the client plugin API.

Several header files contain information relevant to authentication plugins:

- plugin.h: Defines the MYSQL_AUTHENTICATION_PLUGIN server plugin type.
- client_plugin.h: Defines the API for client plugins. This includes the client plugin descriptor and function prototypes for client plugin C API calls (see Section 21.8.14, “C API Client Plugin Functions”).
- plugin_auth.h: Defines the part of the server plugin API specific to authentication plugins. This includes the type-specific descriptor for server-side authentication plugins and the MYSQL_SERVER_AUTH_INFO structure.
- plugin_auth_common.h: Contains common elements of client and server authentication plugins. This includes return value definitions and the MYSQL_PLUGIN_VIO structure.

To write an authentication plugin, include the following header files in the plugin source file. Other MySQL or general header files might also be needed, depending on the plugin capabilities and requirements.

- For a source file that implements a server authentication plugin, include this file:

```
#include <mysql/plugin_auth.h>
```

- For a source file that implements a client authentication plugin, or both client and server plugins, include these files:

```
#include <mysql/plugin_auth.h>
#include <mysql/client_plugin.h>
#include <mysql.h>
```

plugin_auth.h includes plugin.h and plugin_auth_common.h, so you need not include the latter files explicitly.

This section describes how to write a pair of simple server and client authentication plugins that work together.

Warning

These plugins accept any non-empty password and the password is sent in clear text. This is insecure, so the plugins *should not be used in production environments.*

The server-side and client-side plugins developed here both are named auth_simple. As described in Section 22.2.4.2, “Plugin Data Structures”, the plugin library file must have the same basename as the client plugin, so the source file name is auth_simple.c and produces a library named auth_simple.so (assuming that your system uses .so as the suffix for library files).

In MySQL source distributions, authentication plugin source is located in the plugin/auth directory and can be examined as a guide to writing other authentication plugins. Also, to see how the built-in authentication plugins are implemented, see sql/sql_acl.cc for plugins that are built in to the MySQL server and sql-common/client.c for plugins that are built in to the libmysqlclient client library. (For the built-in client plugins, note that the auth_plugin_t structures used there differ from the structures used with the usual client plugin declaration macros. In particular, the first two members are provided explicitly, not by declaration macros.)

Writing the Server-Side Authentication Plugin

Declare the server-side plugin with the usual general descriptor format that is used for all server plugin types (see Server Plugin Library and Plugin Descriptors). For the auth_simple plugin, the descriptor looks like this:

```
mysql_declare_plugin(auth_simple)
{
  MYSQL_AUTHENTICATION_PLUGIN,
  &auth_simple_handler,                 /* type-specific descriptor */
  "auth_simple",                        /* plugin name */
  "Author Name",                        /* author */
  "Any-password authentication plugin", /* description */
  PLUGIN_LICENSE_GPL,                   /* license type */
  NULL,                                 /* no init function */
  NULL,                                 /* no deinit function */
  0x0100,                               /* version = 1.0 */
  NULL,                                 /* no status variables */
  NULL,                                 /* no system variables */
  NULL,                                 /* no reserved information */
  0                                     /* no flags */
}
mysql_declare_plugin_end;
```

The name member (auth_simple) indicates the name to use for references to the plugin in statements such as INSTALL PLUGIN or UNINSTALL PLUGIN. This is also the name displayed by SHOW PLUGINS or INFORMATION_SCHEMA.PLUGINS.

The auth_simple_handler member of the general descriptor points to the type-specific descriptor. For an authentication plugin, the type-specific descriptor is an instance of the st_mysql_auth structure (defined in plugin_auth.h):

```
struct st_mysql_auth
{
  int interface_version;
  const char *client_auth_plugin;
  int (*authenticate_user)(MYSQL_PLUGIN_VIO *vio, MYSQL_SERVER_AUTH_INFO *info);
};
```

The st_mysql_auth structure has three members: The type-specific API version number, the client plugin name, and a pointer to the main plugin function that communicates with the client. The client_auth_plugin member should indicate the name of the client plugin if a specific plugin is required. A value of NULL means “any plugin.” In the latter case, whatever plugin the client uses will do. This is useful if the server plugin does not care about the client plugin or what user name or password it sends. For example, this might be true if the server plugin authenticates only local clients and uses some property of the operating system rather than the information sent by the client plugin.

For auth_simple, the type-specific descriptor looks like this:

```
static struct st_mysql_auth auth_simple_handler =
{
  MYSQL_AUTHENTICATION_INTERFACE_VERSION,
  "auth_simple",           /* required client-side plugin name */
  auth_simple_server       /* server-side plugin main function */
};
```

The main function takes two arguments representing an I/O structure and a MYSQL_SERVER_AUTH_INFO structure. The structure definition is found in plugin_auth.h. It looks like this:

```
typedef struct st_mysql_server_auth_info
{
  char *user_name;
  unsigned int user_name_length;
  const char *auth_string;
  unsigned long auth_string_length;
  char authenticated_as[MYSQL_USERNAME_LENGTH+1];
  char external_user[512];
  int  password_used;
  const char *host_or_ip;
  unsigned int host_or_ip_length;
} MYSQL_SERVER_AUTH_INFO;
```

The character set for string members is UTF-8. If there is a _length member associated with a string, it indicates the string length in bytes. Strings are also null-terminated.

When an authentication plugin is invoked by the server, it should interpret the MYSQL_SERVER_AUTH_INFO structure members as follows. Some of these are used to set the value of SQL functions or system variables within the client session, as indicated.

- user_name: The user name sent by the client. The value becomes the USER() function value.
- user_name_length: The length of user_name in bytes.

- auth_string: The value of the authentication_string column of the mysql.user table row for the matching account name (that is, the row that matches the client user name and host name and that the server uses to determine how to authenticate the client).

  Suppose that you create an account using the following statement:

  ```
  CREATE USER 'my_user'@'localhost'
    IDENTIFIED WITH my_plugin AS 'my_auth_string';
  ```

  When my_user connects from the local host, the server invokes my_plugin and passes 'my_auth_string' to it as the auth_string value.

- auth_string_length: The length of auth_string in bytes.

- authenticated_as: The server sets this to the user name (the value of user_name). The plugin can alter it to indicate that the client should have the privileges of a different user. For example, if the plugin supports proxy users, the initial value is the name of the connecting (proxy) user, and the plugin can change this member to the proxied user name. The server then treats the proxy user as having the privileges of the proxied user (assuming that the other conditions for proxy user support are satisfied; see Implementing Proxy User Support in Authentication Plugins). The value is represented as a string at most MYSQL_USER_NAME_LENGTH bytes long, plus a terminating null. The value becomes the CURRENT_USER() function value.

- external_user: The server sets this to the empty string (null terminated). Its value becomes the external_user system variable value. If the plugin wants that system variable to have a different value, it should set this member accordingly; for example, to the connecting user name. The value is represented as a string at most 511 bytes long, plus a terminating null.

- password_used: This member applies when authentication fails. The plugin can set it or ignore it. The value is used to construct the failure error message of Authentication fails. Password used: %s. The value of password_used determines how %s is handled, as shown in the following table.

password_used	%s Handling
0	NO
1	YES
2	There will be no %s

- host_or_ip: The name of the client host if it can be resolved, or the IP address otherwise.

- host_or_ip_length: The length of host_or_ip in bytes.

The auth_simple main function, auth_simple_server(), reads the password (a null-terminated string) from the client and succeeds if the password is nonempty (first byte not null):

```
static int auth_simple_server (MYSQL_PLUGIN_VIO *vio,
                               MYSQL_SERVER_AUTH_INFO *info)
{
  unsigned char *pkt;
  int pkt_len;

  /* read the password as null-terminated string, fail on error */
  if ((pkt_len= vio->read_packet(vio, &pkt)) < 0)
    return CR_ERROR;

  /* fail on empty password */
  if (!pkt_len || *pkt == '\0')
```

```
  {
    info->password_used= PASSWORD_USED_NO;
    return CR_ERROR;
  }

  /* accept any nonempty password */
  info->password_used= PASSWORD_USED_YES;

  return CR_OK;
}
```

The main function should return one of the error codes shown in the following table.

Error Code	Meaning
CR_OK	Success
CR_OK_HANDSHAKE_COMPLETE	Do not send a status packet back to client
CR_ERROR	Error
CR_AUTH_USER_CREDENTIALS	Authentication failure
CR_AUTH_HANDSHAKE	Authentication handshake failure
CR_AUTH_PLUGIN_ERROR	Internal plugin error

For an example of how the handshake works, see the `plugin/auth/dialog.c` source file.

The server counts plugin errors in the Performance Schema `host_cache` table.

`auth_simple_server_main()` is so basic that it does not use the authentication information structure except to set the member that indicates whether a password was received.

A plugin that supports proxy users must return to the server the name of the proxied user (the MySQL user whose privileges the client user should get). To do this, the plugin must set the `info->authenticated_as` member to the proxied user name. For information about proxying, see Section 6.3.10, “Proxy Users”, and Implementing Proxy User Support in Authentication Plugins.

Writing the Client-Side Authentication Plugin

Declare the client-side plugin descriptor with the `mysql_declare_client_plugin()` and `mysql_end_client_plugin` macros (see Client Plugin Descriptors). For the `auth_simple` plugin, the descriptor looks like this:

```
mysql_declare_client_plugin(AUTHENTICATION)
  "auth_simple",                        /* plugin name */
  "Author Name",                        /* author */
  "Any-password authentication plugin", /* description */
  {1,0,0},                              /* version = 1.0.0 */
  "GPL",                                /* license type */
  NULL,                                 /* for internal use */
  NULL,                                 /* no init function */
  NULL,                                 /* no deinit function */
  NULL,                                 /* no option-handling function */
  auth_simple_client                    /* main function */
mysql_end_client_plugin;
```

The descriptor members from the plugin name through the option-handling function are common to all client plugin types. (For descriptions, see Client Plugin Descriptors.) Following the common members, the descriptor has an additional member specific to authentication plugins. This is the “main” function, which handles communication with the server. The function takes two arguments representing an I/O structure and a connection handler. For our simple any-password plugin, the main function does nothing but write to the server the password provided by the user:

```
static int auth_simple_client (MYSQL_PLUGIN_VIO *vio, MYSQL *mysql)
{
  int res;

  /* send password as null-terminated string in clear text */
  res= vio->write_packet(vio, (const unsigned char *) mysql->passwd,
                         strlen(mysql->passwd) + 1);

  return res ? CR_ERROR : CR_OK;
}
```

The main function should return one of the error codes shown in the following table.

Error Code	Meaning
CR_OK	Success
CR_OK_HANDSHAKE_COMPLETE	Success, client done
CR_ERROR	Error

CR_OK_HANDSHAKE_COMPLETE indicates that the client has done its part successfully and has read the last packet. A client plugin may return CR_OK_HANDSHAKE_COMPLETE if the number of round trips in the authentication protocol is not known in advance and the plugin must read another packet to determine whether authentication is finished.

Using the Authentication Plugins

To compile and install a plugin library object file, see the instructions in Section 22.2.4.3, “Compiling and Installing Plugin Libraries”. To use the library file, it must be installed in the plugin directory (the directory named by the plugin_dir system variable).

Register the server-side plugin with the server. For example, to load the plugin at server startup, use a --plugin-load=auth_simple.so option (change the library suffix as necessary for your system).

Create a user for whom the server will use the auth_simple plugin for authentication:

```
mysql> CREATE USER 'x'@'localhost'
    -> IDENTIFIED WITH auth_simple;
```

Use a client program to connect to the server as user x. The server-side auth_simple plugin communicates with the client program that it should use the client-side auth_simple plugin, and the latter sends the password to the server. The server plugin should reject connections that send an empty password and accept connections that send a nonempty password. Invoke the client program each way to verify this:

```
shell> mysql --user=x --skip-password
ERROR 1045 (28000): Access denied for user 'x'@'localhost' (using password: NO)

shell> mysql --user=x --password=abc
mysql>
```

Because the server plugin accepts any nonempty password, it should be considered insecure. After testing the plugin to verify that it works, restart the server without the --plugin-load option so as not to indavertently leave the server running with an insecure authentication plugin loaded. Also, drop the user with DROP USER 'x'@'localhost'.

For additional information about loading and using authentication plugins, see Section 5.1.8.1, “Installing and Uninstalling Plugins”, and Section 6.3.8, “Pluggable Authentication”.

If you are writing a client program that supports the use of authentication plugins, normally such a program causes a plugin to be loaded by calling `mysql_options()` to set the `MYSQL_DEFAULT_AUTH` and `MYSQL_PLUGIN_DIR` options:

```
char *plugin_dir = "path_to_plugin_dir";
char *default_auth = "plugin_name";

/* ... process command-line options ... */

mysql_options(&mysql, MYSQL_PLUGIN_DIR, plugin_dir);
mysql_options(&mysql, MYSQL_DEFAULT_AUTH, default_auth);
```

Typically, the program will also accept `--plugin-dir` and `--default-auth` options that enable users to override the default values.

Should a client program require lower-level plugin management, the client library contains functions that take an `st_mysql_client_plugin` argument. See Section 21.8.14, “C API Client Plugin Functions”.

Implementing Proxy User Support in Authentication Plugins

One of the capabilities that pluggable authentication makes possible is proxy users (see Section 6.3.10, “Proxy Users”). For a server-side authentication plugin to participate in proxy user support, these conditions must be satisfied:

- When a connecting client should be treated as a proxy user, the plugin must return a different name in the `authenticated_as` member of the `MYSQL_SERVER_AUTH_INFO` structure, to indicate the proxied user name. It may also optionally set the `external_user` member, to set the value of the `external_user` system variable.
- Proxy user accounts must be set up to be authenticated by the plugin. Use the `CREATE USER` or `GRANT` statement to associate accounts with plugins.
- Proxy user accounts must have the `PROXY` privilege for the proxied accounts. Use the `GRANT` statement to grant this privilege.

In other words, the only aspect of proxy user support required of the plugin is that it set `authenticated_as` to the proxied user name. The rest is optional (setting `external_user`) or done by the DBA using SQL statements.

How does an authentication plugin determine which proxied user to return when the proxy user connects? That depends on the plugin. Typically, the plugin maps clients to proxied users based on the authentication string passed to it by the server. This string comes from the `AS` part of the `IDENTIFIED WITH` clause of the `CREATE USER` statement that specifies use of the plugin for authentication.

The plugin developer determines the syntax rules for the authentication string and implements the plugin according to those rules. Suppose that a plugin takes a comma-separated list of pairs that map external users to MySQL users. For example:

```
CREATE USER ''@'%.example.com'
  IDENTIFIED WITH my_plugin AS 'extuser1=mysqlusera, extuser2=mysqluserb'
CREATE USER ''@'%.example.org'
  IDENTIFIED WITH my_plugin AS 'extuser1=mysqluserc, extuser2=mysqluserd'
```

When the server invokes a plugin to authenticate a client, it passes the appropriate authentication string to the plugin. The plugin is responsible to:

1. Parse the string into its components to determine the mapping to use
2. Compare the client user name to the mapping
3. Return the proper MySQL user name

For example, if extuser2 connects from an example.com host, the server passes 'extuser1=mysqlusera, extuser2=mysqluserb' to the plugin, and the plugin should copy mysqluserb into authenticated_as, with a terminating null byte. If extuser2 connects from an example.org host, the server passes 'extuser1=mysqluserc, extuser2=mysqluserd', and the plugin should copy mysqluserd instead.

If there is no match in the mapping, the action depends on the plugin. If a match is required, the plugin likely will return an error. Or the plugin might simply return the client name; in this case, it should not change authenticated_as, and the server will not treat the client as a proxy.

The following example demonstrates how to handle proxy users using a plugin named auth_simple_proxy. Like the auth_simple plugin described earlier, auth_simple_proxy accepts any nonempty password as valid (and thus should not be used in production environments). In addition, it examines the auth_string authentication string member and uses these very simple rules for interpreting it:

- If the string is empty, the plugin returns the user name as given and no proxying occurs. That is, the plugin leaves the value of authenticated_as unchanged.
- If the string is nonempty, the plugin treats it as the name of the proxied user and copies it to authenticated_as so that proxying occurs.

For testing, set up one account that is not proxied according to the preceding rules, and one that is. This means that one account has no AS clause, and one includes an AS clause that names the proxied user:

```
CREATE USER 'plugin_user1'@'localhost'
  IDENTIFIED WITH auth_simple_proxy;
CREATE USER 'plugin_user2'@'localhost'
  IDENTIFIED WITH auth_simple_proxy AS 'proxied_user';
```

In addition, create an account for the proxied user and grant plugin_user2 the PROXY privilege for it:

```
CREATE USER 'proxied_user'@'localhost'
  IDENTIFIED BY 'proxied_user_pass';
GRANT PROXY
  ON 'proxied_user'@'localhost'
  TO 'plugin_user2'@'localhost';
```

Before the server invokes an authentication plugin, it sets authenticated_as to the client user name. To indicate that the user is a proxy, the plugin should set authenticated_as to the proxied user name. For auth_simple_proxy, this means that it must examine the auth_string value, and, if the value is nonempty, copy it to the authenticated_as member to return it as the name of the proxied user. In addition, when proxying occurs, the plugin sets the external_user member to the client user name; this becomes the value of the external_user system variable.

```
static int auth_simple_proxy_server (MYSQL_PLUGIN_VIO *vio,
                                     MYSQL_SERVER_AUTH_INFO *info)
{
  unsigned char *pkt;
  int pkt_len;
  /* read the password as null-terminated string, fail on error */
  if ((pkt_len= vio->read_packet(vio, &pkt)) < 0)
    return CR_ERROR;

  /* fail on empty password */
  if (!pkt_len || *pkt == '\0')
  {
    info->password_used= PASSWORD_USED_NO;
    return CR_ERROR;
  }

  /* accept any nonempty password */
  info->password_used= PASSWORD_USED_YES;

  /* if authentication string is nonempty, use as proxied user name */
  /* and use client name as external_user value */
  if (info->auth_string_length > 0)
  {
    strcpy (info->authenticated_as, info->auth_string);
    strcpy (info->external_user, info->user_name);
  }

  return CR_OK;
}
```

After a successful connection, the USER() function should indicate the connecting client user and host name, and CURRENT_USER() should indicate the account whose privileges apply during the session. The latter value should be the connecting user account if no proxying occurs or the proxied account if proxying does occur.

Compile and install the plugin, then test it. First, connect as plugin_user1:

```
shell> mysql --user=plugin_user1 --password=x
```

In this case, there should be no proxying:

```
mysql> SELECT USER(), CURRENT_USER(), @@proxy_user, @@external_user\G
*************************** 1. row ***************************
         USER(): plugin_user1@localhost
 CURRENT_USER(): plugin_user1@localhost
   @@proxy_user: NULL
@@external_user: NULL
```

Then connect as plugin_user2:

```
shell> mysql --user=plugin_user2 --password=x
```

In this case, plugin_user2 should be proxied to proxied_user:

```
mysql> SELECT USER(), CURRENT_USER(), @@proxy_user, @@external_user\G
*************************** 1. row ***************************
         USER(): plugin_user2@localhost
 CURRENT_USER(): proxied_user@localhost
   @@proxy_user: 'plugin_user2'@'localhost'
@@external_user: 'plugin_user2'@'localhost'
```

22.2.4.10 Writing Password-Validation Plugins

This section describes how to write a password-validation server plugin. The instructions are based on the source code in the plugin/password_validation directory of MySQL source distributions.

The validate_password.cc source file in that directory implements the the plugin named validate_password.

To write a password-validation plugin, include the following header file in the plugin source file. Other MySQL or general header files might also be needed, depending on the plugin capabilities and requirements.

```
#include <mysql/plugin_validate_password.h>
```

plugin_validate_password.h includes plugin.h, so you need not include the latter file explicitly. plugin.h defines the MYSQL_VALIDATE_PASSWORD_PLUGIN server plugin type and the data structures needed to declare the plugin. plugin_validate_password.h defines data structures specific to password-validation plugins.

A password-validation plugin, like any MySQL server plugin, has a general plugin descriptor (see Server Plugin Library and Plugin Descriptors). In validate_password.cc, the general descriptor looks like this:

```
mysql_declare_plugin(validate_password)
{
  MYSQL_VALIDATE_PASSWORD_PLUGIN,     /*   type                            */
  &validate_password_descriptor,      /*   descriptor                      */
  "validate_password",                /*   name                            */
  "Oracle Corporation",               /*   author                          */
  "check password strength",          /*   description                     */
  PLUGIN_LICENSE_GPL,
  validate_password_init,             /*   init function (when loaded)     */
  validate_password_deinit,           /*   deinit function (when unloaded) */
  0x0100,                             /*   version                         */
  NULL,
  validate_password_system_variables, /*   system variables                */
  NULL,
  0,
}
mysql_declare_plugin_end;
```

The name member (validate_password) indicates the name to use for references to the plugin in statements such as INSTALL PLUGIN or UNINSTALL PLUGIN. This is also the name displayed by INFORMATION_SCHEMA.PLUGINS or SHOW PLUGINS.

The general descriptor also refers to validate_password_system_variables, a structure that exposes several system variables to the SHOW VARIABLES statement:

```
static struct st_mysql_sys_var* validate_password_system_variables[]= {
  MYSQL_SYSVAR(length),
  MYSQL_SYSVAR(number_count),
  MYSQL_SYSVAR(mixed_case_count),
  MYSQL_SYSVAR(special_char_count),
  MYSQL_SYSVAR(policy),
  MYSQL_SYSVAR(dictionary_file),
  NULL
};
```

The validate_password_init initialization function reads the dictionary file if one was specified, and the validate_password_deinit function frees data structures associated with the file.

The validate_password_descriptor value in the general descriptor points to the type-specific descriptor. For password-validation plugins, this descriptor has the following structure:

```
struct st_mysql_validate_password
{
  int interface_version;
  /*
    This function returns TRUE for passwords which satisfy the password
    policy (as chosen by plugin variable) and FALSE for all other
    password
  */
  int (*validate_password)(mysql_string_handle password);
  /*
    This function returns the password strength (0-100) depending
    upon the policies
  */
  int (*get_password_strength)(mysql_string_handle password);
};
```

The type-specific descriptor has these members:

- `interface_version`: By convention, type-specific plugin descriptors begin with the interface version for the given plugin type. The server checks `interface_version` when it loads the plugin to see whether the plugin is compatible with it. For password-validation plugins, the value of the `interface_version` member is `MYSQL_VALIDATE_PASSWORD_INTERFACE_VERSION` (defined in `plugin_validate_password.h`).

- `validate_password`: A function that the server calls to test whether a password satisfies the current password policy. It returns 1 if the password is okay and 0 otherwise. The argument is the password, passed as a `mysql_string_handle` value. This data type is implemented by the `mysql_string` server service. For details, see the `string_service.h` and `string_service.cc` source files in the `sql` directory.

- `get_password_strength`: A function that the server calls to assess the strength of a password. It returns a value from 0 (weak) to 100 (strong). The argument is the password, passed as a `mysql_string_handle` value.

For the `validate_password` plugin, the type-specific descriptor looks like this:

```
static struct st_mysql_validate_password validate_password_descriptor=
{
  MYSQL_VALIDATE_PASSWORD_INTERFACE_VERSION,
  validate_password,                        /* validate function          */
  get_password_strength                     /* validate strength function */
};
```

To compile and install a plugin library object file, use the instructions in Section 22.2.4.3, “Compiling and Installing Plugin Libraries”. To use the library file, it must be installed in the plugin directory (the directory named by the `plugin_dir` system variable). For the `validate_password` plugin, it is compiled and installed when you build MySQL from source. It is also included in binary distributions. The build process produces a shared object library with a name of `validate_password.so` (the extension might be different depending on your platform).

To register the plugin at runtime, use this statement (change the extension as necessary):

```
mysql> INSTALL PLUGIN validate_password SONAME 'validate_password.so';
```

For additional information about plugin loading, see Section 5.1.8.1, “Installing and Uninstalling Plugins”.

To verify plugin installation, examine the `INFORMATION_SCHEMA.PLUGINS` table or use the `SHOW PLUGINS` statement.

While the validate_password plugin is installed, it exposes system variables that indicate the password-checking parameters:

```
mysql> SHOW VARIABLES LIKE 'validate_password%';
+--------------------------------------+--------+
| Variable_name                        | Value  |
+--------------------------------------+--------+
| validate_password_dictionary_file    |        |
| validate_password_length             | 8      |
| validate_password_mixed_case_count   | 1      |
| validate_password_number_count       | 1      |
| validate_password_policy             | MEDIUM |
| validate_password_special_char_count | 1      |
+--------------------------------------+--------+
```

For descriptions of these variables, see Password Validation Plugin Options and Variables.

To disable the plugin after testing it, use this statement to unload it:

```
mysql> UNINSTALL PLUGIN validate_password;
```

22.2.4.11 Writing Protocol Trace Plugins

MySQL supports the use of protocol trace plugins: client-side plugins that implement tracing of communication between a client and the server that takes place using the client/server protocol. This capability can be used in MySQL 5.7.2 and up.

Using the Test Protocol Trace Plugin

MySQL includes a test protocol trace plugin that serves to illustrate the information available from such plugins, and as a guide to writing other protocol trace plugins. To see how the test plugin works, use a MySQL source distribution; binary distributions are built with the test plugin disabled.

Enable the test protocol trace plugin by configuring MySQL with the WITH_TEST_TRACE_PLUGIN CMake option enabled. This causes the test trace plugin to be built and MySQL client programs to load it, but the plugin has no effect by default. Control the plugin using these environment variables:

- MYSQL_TEST_TRACE_DEBUG: Set this variable to a value other than 0 to cause the test plugin to produce diagnostic output on stderr.

- MYSQL_TRACE_TRACE_CRASH: Set this variable to a value other than 0 to cause the test plugin to abort the client program if it detects an invalid trace event.

⚠ **Caution**

Diagnostic output from the test protocol trace plugin can disclose passwords and other sensitive information.

Given a MySQL installation built from source with the test plugin enabled, you can see a trace of the communication between the mysql client and the MySQL server as follows:

```
shell> export MYSQL_TEST_TRACE_DEBUG=1
shqll> mysql
test_trace: Test trace plugin initialized
test_trace: Starting tracing in stage CONNECTING
test_trace: stage: CONNECTING, event: CONNECTING
test_trace: stage: CONNECTING, event: CONNECTED
```

```
test_trace: stage: WAIT_FOR_INIT_PACKET, event: READ_PACKET
test_trace: stage: WAIT_FOR_INIT_PACKET, event: PACKET_RECEIVED
test_trace: packet received: 87 bytes
  0A 35 2E 37 2E 33 2D 6D  31 33 2D 64 65 62 75 67   .5.7.3-m13-debug
  2D 6C 6F 67 00 04 00 00  00 2B 7C 4F 55 3F 79 67   -log.....+|OU?yg
test_trace: 004: stage: WAIT_FOR_INIT_PACKET, event: INIT_PACKET_RECEIVED
test_trace: 004: stage: AUTHENTICATE, event: AUTH_PLUGIN
test_trace: 004: Using authentication plugin: mysql_native_password
test_trace: 004: stage: AUTHENTICATE, event: SEND_AUTH_RESPONSE
test_trace: 004: sending packet: 188 bytes
  85 A6 7F 00 00 00 00 01  21 00 00 00 00 00 00 00   .?......!.......
  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
...
mysql> quit
test_trace: 008: stage: READY_FOR_COMMAND, event: SEND_COMMAND
test_trace: 008: QUIT
test_trace: 008: stage: READY_FOR_COMMAND, event: PACKET_SENT
test_trace: 008: packet sent: 0 bytes
test_trace: 008: stage: READY_FOR_COMMAND, event: DISCONNECTED
test_trace: 008: Connection  closed
test_trace: 008: Tracing connection has ended
Bye
test_trace: Test trace plugin de-initialized
```

To disable trace output, do this:

```
shell> MYSQL_TEST_TRACE_DEBUG=
```

Using Your Own Protocol Trace Plugins

Note

To use your own protocol trace plugins, you must configure MySQL with the `WITH_TEST_TRACE_PLUGIN` CMake option *disabled* because only one protocol trace plugin can be loaded at a time and an error occurs for attempts to load a second one. If you have already built MySQL with the test protocol trace plugin enabled to see how it works, you must rebuild MySQL without it before you can use your own plugins.

This section discusses how to write a basic protocol trace plugin named `simple_trace`. This plugin provides a framework showing how to set up the client plugin descriptor and create the trace-related callback functions. In `simple_trace`, these functions are rudimentary and do little other than illustrate the arguments required. To see in detail how a trace plugin can make use of trace event information, check the source file for the test protocol trace plugin (`test_trace_plugin.cc` in the `libmysql` directory of a MySQL source distribution). However, note that the `st_mysql_client_plugin_TRACE` structure used there differs from the structures used with the usual client plugin declaration macros. In particular, the first two members are defined explicitly, not implicitly by declaration macros.

Several header files contain information relevant to protocol trace plugins:

- `client_plugin.h`: Defines the API for client plugins. This includes the client plugin descriptor and function prototypes for client plugin C API calls (see Section 21.8.14, “C API Client Plugin Functions”).

- `plugin_trace.h`: Contains declarations for client-side plugins of type `MYSQL_CLIENT_TRACE_PLUGIN`. It also contains descriptions of the permitted protocol stages, transitions between stages, and the types of events permitted at each stage.

To write a protocol trace plugin, include the following header files in the plugin source file. Other MySQL or general header files might also be needed, depending on the plugin capabilities and requirements.

```
#include <mysql/plugin_trace.h>
#include <mysql.h>
```

plugin_trace.h includes client_plugin.h, so you need not include the latter file explicitly.

Declare the client-side plugin descriptor with the mysql_declare_client_plugin() and mysql_end_client_plugin macros (see Client Plugin Descriptors). For the simple_trace plugin, the descriptor looks like this:

```
mysql_declare_client_plugin(TRACE)
  "simple_trace",                 /* plugin name */
  "Author Name",                  /* author */
  "Simple protocol trace plugin", /* description */
  {1,0,0},                        /* version = 1.0.0 */
  "GPL",                          /* license type */
  NULL,                           /* for internal use */
  plugin_init,                    /* initialization function */
  plugin_deinit,                  /* deinitialization function */
  plugin_options,                 /* option-handling function */
  trace_start,                    /* start-trace function */
  trace_stop,                     /* stop-trace function */
  trace_event                     /* event-handling function */
mysql_end_client_plugin;
```

The descriptor members from the plugin name through the option-handling function are common to all client plugin types. The members following the common members implement trace event handling.

Function members for which the plugin needs no processing can be declared as NULL in the descriptor, in which case you need not write any corresponding function. For illustration purposes and to show the argument syntax, the following discussion implements all functions listed in the descriptor, even though some of them do nothing,

The initialization, deinitialization, and options functions common to all client plugins are declared as follows. For a description of the arguments and return values, see Client Plugin Descriptors.

```
static int
plugin_init(char *errbuf, size_t errbuf_len, int argc, va_list args)
{
  return 0;
}

static int
plugin_deinit()
{
  return 0;
}

static int
plugin_options(const char *option, const void *value)
{
  return 0;
}
```

The trace-specific members of the client plugin descriptor are callback functions. The following descriptions provide more detail on how they are used. Each has a first argument that is a pointer to the plugin instance in case your implementation needs to access it.

trace_start(): This function is called at the start of each traced connection (each connection that starts after the plugin is loaded). It is passed the connection handler and the protocol stage at which tracing starts. `trace_start()` allocates memory needed by the `trace_event()` function, if any, and returns a pointer to it. If no memory is needed, this function returns `NULL`.

```
static void*
trace_start(struct st_mysql_client_plugin_TRACE *self,
            MYSQL *conn,
            enum protocol_stage stage)
{
  struct st_trace_data *plugin_data= malloc(sizeof(struct st_trace_data));

  fprintf(stderr, "Initializing trace: stage %d\n", stage);
  if (plugin_data)
  {
    memset(plugin_data, 0, sizeof(struct st_trace_data));
    fprintf(stderr, "Trace initialized\n");
    return plugin_data;
  }
  fprintf(stderr, "Could not initialize trace\n");
  exit(1);
}
```

`trace_stop()`: This function is called when tracing of the connection ends. That usually happens when the connection is closed, but can happen earlier. For example, `trace_event()` can return a nonzero value at any time and that causes tracing of the connection to terminate. `trace_stop()` is then called even though the connection has not ended.

`trace_stop()` is passed the connection handler and a pointer to the memory allocated by `trace_start()` (`NULL` if none). If the pointer is non-`NULL`, `trace_stop()` should deallocate the memory. This function returns no value.

```
static void
trace_stop(struct st_mysql_client_plugin_TRACE *self,
           MYSQL *conn,
           void *plugin_data)
{
  fprintf(stderr, "Terminating trace\n");
  if (plugin_data)
    free(plugin_data);
}
```

`trace_event()`: This function is called for each event occurrence. It is passed a pointer to the memory allocated by `trace_start()` (`NULL` if none), the connection handler, the current protocol stage and event codes, and event data. This function returns 0 to continue tracing, nonzero if tracing should stop.

```
static int
trace_event(struct st_mysql_client_plugin_TRACE *self,
            void *plugin_data,
            MYSQL *conn,
            enum protocol_stage stage,
            enum trace_event event,
            struct st_trace_event_args args)
{
  fprintf(stderr, "Trace event received: stage %d, event %d\n", stage, event);
  if (event == TRACE_EVENT_DISCONNECTED)
    fprintf(stderr, "Connection closed\n");
  return 0;
}
```

The tracing framework shuts down tracing of the connection when the connection ends, so `trace_event()` should return nonzero only if you want to terminate tracing of the connection early.

Suppose that you want to trace only connections for a certain MySQL account. After authentication, you can check the user name for the connection and stop tracing if it is not the user in whom you are interested.

For each call to `trace_event()`, the `st_trace_event_args` structure contains the event data. It has this definition:

```
struct st_trace_event_args
{
  const char           *plugin_name;
  int                   cmd;
  const unsigned char  *hdr;
  size_t                hdr_len;
  const unsigned char  *pkt;
  size_t                pkt_len;
};
```

For different event types, the `st_trace_event_args` structure contains the information described following. All lengths are in bytes. Unused members are set to `0`/`NULL`.

`AUTH_PLUGIN` event:

```
plugin_name  The name of the plugin
```

`SEND_COMMAND` event:

```
cmd          The command code
hdr          Pointer to the command packet header
hdr_len      Length of the header
pkt          Pointer to the command arguments
pkt_len      Length of the arguments
```

Other `SEND_xxx` and `xxx_RECEIVED` events:

```
pkt          Pointer to the data sent or received
pkt_len      Length of the data
```

`PACKET_SENT` event:

```
pkt_len      Number of bytes sent
```

To compile and install a plugin library object file, see the instructions in Section 22.2.4.3, “Compiling and Installing Plugin Libraries”. To use the library file, it must be installed in the plugin directory (the directory named by the `plugin_dir` system variable).

After the plugin library file is compiled and installed in the plugin directory, you can test it easily by setting the `LIBMYSQL_PLUGINS` environment variable to the plugin name, which affects any client program that uses that variable. `mysql` is one such program:

```
shell> export LIBMYSQL_PLUGINS=simple_trace
shqll> mysql
Initializing trace: stage 0
Trace initialized
Trace event received: stage 0, event 1
Trace event received: stage 0, event 2
...
```

```
Welcome to the MySQL monitor.  Commands end with ; or \g.
Trace event received
Trace event received
...
mysql> SELECT 1;
Trace event received: stage 4, event 12
Trace event received: stage 4, event 16
...
Trace event received: stage 8, event 14
Trace event received: stage 8, event 15
+---+
| 1 |
+---+
| 1 |
+---+
1 row in set (0.00 sec)

mysql> quit
Trace event received: stage 4, event 12
Trace event received: stage 4, event 16
Trace event received: stage 4, event 3
Connection closed
Terminating trace
Bye
```

To stop the trace plugin from being loaded, do this:

```
shell> LIBMYSQL_PLUGINS=
```

It is also possible to write client programs that directly load the plugin. You can tell the client where the plugin directory is located by calling `mysql_options()` to set the `MYSQL_PLUGIN_DIR` option:

```
char *plugin_dir = "path_to_plugin_dir";

/* ... process command-line options ... */

mysql_options(&mysql, MYSQL_PLUGIN_DIR, plugin_dir);
```

Typically, the program will also accept a `--plugin-dir` option that enables users to override the default value.

Should a client program require lower-level plugin management, the client library contains functions that take an `st_mysql_client_plugin` argument. See Section 21.8.14, “C API Client Plugin Functions”.

22.2.5 MySQL Services for Plugins

MySQL server plugins have access to server “services.” The services interface exposes server functionality that plugins can call. It complements the plugin API and has these characteristics:

- Services enable plugins to access code inside the server using ordinary function calls.
- Services are portable and work on multiple platforms.
- The interface includes a versioning mechanism so that service versions supported by the server can be checked at load time against plugin versions. Versioning protects against incompatibilities between the version of a service that the server provides and the version of the service expected or required by a plugin.

Current services include the following, and others can be implemented:

- my_plugin_log_service: A service that enables plugins to report errors and specify error messages. The server writes the messages to the error log.
- my_snprintf: A string-formatting service that produces consistent results across platforms.
- my_thd_scheduler: A service for plugins to select a thread scheduler.
- mysql_string: A service for string manipulation.
- thd_alloc: A memory-allocation service.
- thd_wait: A service for plugins to report when they are going to sleep or stall.

The plugin services interface differs from the plugin API as follows:

- The plugin API enables plugins to be used by the server. The calling initiative lies with the server to invoke plugins. This enables plugins to extend server functionality or register to receive notifications about server processing.
- The plugin services interface enables plugins to call code inside the server. The calling initiative lies with plugins to invoke service functions. This enables functionality already implemented in the server to be used by many plugins; they need not individually implement it themselves.

For developers who wish to modify the server to add a new service, see MySQL Services for Plugins.

The remainder of this section describes how a plugin uses server functionality that is available as a service. See also the source for the “daemon” example plugin, which uses the my_snprintf service. Within a MySQL source distribution, that plugin is located in the plugin/daemon_example directory.

To determine what services exist and what functions they provide, look in the include/mysql directory of a MySQL source distribution. The relevant files are:

- plugin.h includes services.h.
- services.h is the “umbrella” header that includes all available service-specific header files.
- Service-specific headers have names like service_my_snprintf.h or service_thd_alloc.h.

Each service-specific header should contain comments that provide full usage documentation for a given service, including what service functions are available, their calling sequences, and return values.

To use a service or services from within a plugin, the plugin source file must include the plugin.h header file to access service-related information:

```
#include <mysql/plugin.h>
```

This does not represent any additional setup cost. A plugin must include that file anyway because it contains definitions and structures that every plugin needs.

To access a service, a plugin calls service functions like any other function. For example, to format a string into a buffer for printing, call the my_snprintf() function provided by the service of the same name:

```
char buffer[BUFFER_SIZE];

my_snprintf(buffer, sizeof(buffer), format_string, argument_to_format, ...);
```

To report an error that the server will write to the error log, first choose an error level. mysql/service_my_plugin_log.h defines these levels:

```
enum plugin_log_level
{
  MY_ERROR_LEVEL,
  MY_WARNING_LEVEL,
  MY_INFORMATION_LEVEL
};
```

Then invoke `my_plugin_log_message()`:

```
int my_plugin_log_message(MYSQL_PLUGIN *plugin, enum plugin_log_level level,
                          const char *format, ...);
```

For example:

```
my_plugin_log_message(plugin_ptr, MY_ERROR_LEVEL, "Cannot initialize plugin");
```

When you build your plugin, you must link in the `libmysqlservices` library. Use the `-lmysqlservices` flag at link time. For example, for `CMake`, put this in the top-level `CMakeLists.txt` file:

```
FIND_LIBRARY(MYSQLSERVICES_LIB mysqlservices
 PATHS "${MYSQL_SRCDIR}/libservices" NO_DEFAULT_PATH)
```

Put this in the `CMakeLists.txt` file in the directory containing the plugin source:

```
# the plugin needs the mysql services library for error logging
TARGET_LINK_LIBRARIES (your_plugin_library_name ${MYSQLSERVICES_LIB})
```

22.3 Adding New Functions to MySQL

There are three ways to add new functions to MySQL:

- You can add functions through the user-defined function (UDF) interface. User-defined functions are compiled as object files and then added to and removed from the server dynamically using the `CREATE FUNCTION` and `DROP FUNCTION` statements. See Section 13.7.3.1, “`CREATE FUNCTION` Syntax for User-Defined Functions”.

- You can add functions as native (built-in) MySQL functions. Native functions are compiled into the `mysqld` server and become available on a permanent basis.

- Another way to add functions is by creating stored functions. These are written using SQL statements rather than by compiling object code. The syntax for writing stored functions is not covered here. See Section 18.2, “Using Stored Routines (Procedures and Functions)”.

Each method of creating compiled functions has advantages and disadvantages:

- If you write user-defined functions, you must install object files in addition to the server itself. If you compile your function into the server, you don't need to do that.

- Native functions require you to modify a source distribution. UDFs do not. You can add UDFs to a binary MySQL distribution. No access to MySQL source is necessary.

- If you upgrade your MySQL distribution, you can continue to use your previously installed UDFs, unless you upgrade to a newer version for which the UDF interface changes. For native functions, you must repeat your modifications each time you upgrade.

Whichever method you use to add new functions, they can be invoked in SQL statements just like native functions such as ABS() or SOUNDEX().

See Section 9.2.4, "Function Name Parsing and Resolution", for the rules describing how the server interprets references to different kinds of functions.

The following sections describe features of the UDF interface, provide instructions for writing UDFs, discuss security precautions that MySQL takes to prevent UDF misuse, and describe how to add native MySQL functions.

For example source code that illustrates how to write UDFs, take a look at the sql/udf_example.c file that is provided in MySQL source distributions.

22.3.1 Features of the User-Defined Function Interface

The MySQL interface for user-defined functions provides the following features and capabilities:

- Functions can return string, integer, or real values and can accept arguments of those same types.
- You can define simple functions that operate on a single row at a time, or aggregate functions that operate on groups of rows.
- Information is provided to functions that enables them to check the number, types, and names of the arguments passed to them.
- You can tell MySQL to coerce arguments to a given type before passing them to a function.
- You can indicate that a function returns NULL or that an error occurred.

22.3.2 Adding a New User-Defined Function

For the UDF mechanism to work, functions must be written in C or C++ and your operating system must support dynamic loading. MySQL source distributions include a file sql/udf_example.c that defines five UDF functions. Consult this file to see how UDF calling conventions work. The include/mysql_com.h header file defines UDF-related symbols and data structures, although you need not include this header file directly; it is included by mysql.h.

A UDF contains code that becomes part of the running server, so when you write a UDF, you are bound by any and all constraints that apply to writing server code. For example, you may have problems if you attempt to use functions from the libstdc++ library. These constraints may change in future versions of the server, so it is possible that server upgrades will require revisions to UDFs that were originally written for older servers. For information about these constraints, see Section 2.8.4, "MySQL Source-Configuration Options", and Section 2.8.5, "Dealing with Problems Compiling MySQL".

To be able to use UDFs, you must link mysqld dynamically. If you want to use a UDF that needs to access symbols from mysqld (for example, the metaphone function in sql/udf_example.c uses default_charset_info), you must link the program with -rdynamic (see man dlopen).

For each function that you want to use in SQL statements, you should define corresponding C (or C ++) functions. In the following discussion, the name "xxx" is used for an example function name. To distinguish between SQL and C/C++ usage, XXX() (uppercase) indicates an SQL function call, and xxx() (lowercase) indicates a C/C++ function call.

Note

When using C++ you can encapsulate your C functions within:

```
extern "C" { ... }
```

This ensures that your C++ function names remain readable in the completed UDF.

The following list describes the C/C++ functions that you write to implement the interface for a function named XXX(). The main function, xxx(), is required. In addition, a UDF requires at least one of the other functions described here, for reasons discussed in Section 22.3.2.6, “User-Defined Function Security Precautions”.

- xxx()

  The main function. This is where the function result is computed. The correspondence between the SQL function data type and the return type of your C/C++ function is shown here.

SQL Type	C/C++ Type
STRING	char *
INTEGER	long long
REAL	double

  It is also possible to declare a DECIMAL function, but currently the value is returned as a string, so you should write the UDF as though it were a STRING function. ROW functions are not implemented.

- xxx_init()

  The initialization function for xxx(). If present, it can be used for the following purposes:

  - To check the number of arguments to XXX().
  - To verify that the arguments are of a required type or, alternatively, to tell MySQL to coerce arguments to the required types when the main function is called.
  - To allocate any memory required by the main function.
  - To specify the maximum length of the result.
  - To specify (for REAL functions) the maximum number of decimal places in the result.
  - To specify whether the result can be NULL.

- xxx_deinit()

  The deinitialization function for xxx(). If present, it should deallocate any memory allocated by the initialization function.

When an SQL statement invokes XXX(), MySQL calls the initialization function xxx_init() to let it perform any required setup, such as argument checking or memory allocation. If xxx_init() returns an error, MySQL aborts the SQL statement with an error message and does not call the main or deinitialization functions. Otherwise, MySQL calls the main function xxx() once for each row. After all rows have been processed, MySQL calls the deinitialization function xxx_deinit() so that it can perform any required cleanup.

For aggregate functions that work like SUM(), you must also provide the following functions:

- xxx_clear()

Reset the current aggregate value but do not insert the argument as the initial aggregate value for a new group.

- `xxx_add()`

  Add the argument to the current aggregate value.

MySQL handles aggregate UDFs as follows:

1. Call `xxx_init()` to let the aggregate function allocate any memory it needs for storing results.
2. Sort the table according to the `GROUP BY` expression.
3. Call `xxx_clear()` for the first row in each new group.
4. Call `xxx_add()` for each row that belongs in the same group.
5. Call `xxx()` to get the result for the aggregate when the group changes or after the last row has been processed.
6. Repeat steps 3 to 5 until all rows has been processed
7. Call `xxx_deinit()` to let the UDF free any memory it has allocated.

All functions must be thread-safe. This includes not just the main function, but the initialization and deinitialization functions as well, and also the additional functions required by aggregate functions. A consequence of this requirement is that you are not permitted to allocate any global or static variables that change! If you need memory, you should allocate it in `xxx_init()` and free it in `xxx_deinit()`.

22.3.2.1 UDF Calling Sequences for Simple Functions

This section describes the different functions that you need to define when you create a simple UDF. Section 22.3.2, “Adding a New User-Defined Function”, describes the order in which MySQL calls these functions.

The main `xxx()` function should be declared as shown in this section. Note that the return type and parameters differ, depending on whether you declare the SQL function `XXX()` to return `STRING`, `INTEGER`, or `REAL` in the `CREATE FUNCTION` statement:

For `STRING` functions:

```
char *xxx(UDF_INIT *initid, UDF_ARGS *args,
          char *result, unsigned long *length,
          char *is_null, char *error);
```

For `INTEGER` functions:

```
long long xxx(UDF_INIT *initid, UDF_ARGS *args,
              char *is_null, char *error);
```

For `REAL` functions:

```
double xxx(UDF_INIT *initid, UDF_ARGS *args,
              char *is_null, char *error);
```

DECIMAL functions return string values and should be declared the same way as STRING functions. ROW functions are not implemented.

The initialization and deinitialization functions are declared like this:

```
my_bool xxx_init(UDF_INIT *initid, UDF_ARGS *args, char *message);

void xxx_deinit(UDF_INIT *initid);
```

The initid parameter is passed to all three functions. It points to a UDF_INIT structure that is used to communicate information between functions. The UDF_INIT structure members follow. The initialization function should fill in any members that it wishes to change. (To use the default for a member, leave it unchanged.)

- my_bool maybe_null

  xxx_init() should set maybe_null to 1 if xxx() can return NULL. The default value is 1 if any of the arguments are declared maybe_null.

- unsigned int decimals

  The number of decimal digits to the right of the decimal point. The default value is the maximum number of decimal digits in the arguments passed to the main function. For example, if the function is passed 1.34, 1.345, and 1.3, the default would be 3, because 1.345 has 3 decimal digits.

  For arguments that have no fixed number of decimals, the decimals value is set to 31, which is 1 more than the maximum number of decimals permitted for the DECIMAL, FLOAT, and DOUBLE data types. In MySQL 5.7, this value is available as the constant NOT_FIXED_DEC in the mysql_com.h header file.

  A decimals value of 31 is used for arguments in cases such as a FLOAT or DOUBLE column declared without an explicit number of decimals (for example, FLOAT rather than FLOAT(10,3)) and for floating-point constants such as 1345E-3. It is also used for string and other nonnumber arguments that might be converted within the function to numeric form.

  The value to which the decimals member is initialized is only a default. It can be changed within the function to reflect the actual calculation performed. The default is determined such that the largest number of decimals of the arguments is used. If the number of decimals is NOT_FIXED_DEC for even one of the arguments, that is the value used for decimals.

- unsigned int max_length

  The maximum length of the result. The default max_length value differs depending on the result type of the function. For string functions, the default is the length of the longest argument. For integer functions, the default is 21 digits. For real functions, the default is 13 plus the number of decimal digits indicated by initid->decimals. (For numeric functions, the length includes any sign or decimal point characters.)

  If you want to return a blob value, you can set max_length to 65KB or 16MB. This memory is not allocated, but the value is used to decide which data type to use if there is a need to temporarily store the data.

- char *ptr

  A pointer that the function can use for its own purposes. For example, functions can use initid->ptr to communicate allocated memory among themselves. xxx_init() should allocate the memory and assign it to this pointer:

  ```
  initid->ptr = allocated_memory;
  ```

In xxx() and xxx_deinit(), refer to initid->ptr to use or deallocate the memory.

- my_bool const_item

  xxx_init() should set const_item to 1 if xxx() always returns the same value and to 0 otherwise.

22.3.2.2 UDF Calling Sequences for Aggregate Functions

This section describes the different functions that you need to define when you create an aggregate UDF. Section 22.3.2, "Adding a New User-Defined Function", describes the order in which MySQL calls these functions.

- xxx_reset()

  This function is called when MySQL finds the first row in a new group. It should reset any internal summary variables and then use the given UDF_ARGS argument as the first value in your internal summary value for the group. Declare xxx_reset() as follows:

  ```
  void xxx_reset(UDF_INIT *initid, UDF_ARGS *args,
                 char *is_null, char *error);
  ```

  xxx_reset() is not needed or used in MySQL 5.7, in which the UDF interface uses xxx_clear() instead. However, you can define both xxx_reset() and xxx_clear() if you want to have your UDF work with older versions of the server. (If you do include both functions, the xxx_reset() function in many cases can be implemented internally by calling xxx_clear() to reset all variables, and then calling xxx_add() to add the UDF_ARGS argument as the first value in the group.)

- xxx_clear()

  This function is called when MySQL needs to reset the summary results. It is called at the beginning for each new group but can also be called to reset the values for a query where there were no matching rows. Declare xxx_clear() as follows:

  ```
  void xxx_clear(UDF_INIT *initid, char *is_null, char *error);
  ```

  is_null is set to point to CHAR(0) before calling xxx_clear().

  If something went wrong, you can store a value in the variable to which the error argument points. error points to a single-byte variable, not to a string buffer.

  xxx_clear() is required by MySQL 5.7.

- xxx_add()

  This function is called for all rows that belong to the same group. You should use it to add the value in the UDF_ARGS argument to your internal summary variable.

  ```
  void xxx_add(UDF_INIT *initid, UDF_ARGS *args,
               char *is_null, char *error);
  ```

The xxx() function for an aggregate UDF should be declared the same way as for a nonaggregate UDF. See Section 22.3.2.1, "UDF Calling Sequences for Simple Functions".

For an aggregate UDF, MySQL calls the xxx() function after all rows in the group have been processed. You should normally never access its UDF_ARGS argument here but instead return a value based on your internal summary variables.

Return value handling in xxx() should be done the same way as for a nonaggregate UDF. See Section 22.3.2.4, “UDF Return Values and Error Handling”.

The xxx_reset() and xxx_add() functions handle their UDF_ARGS argument the same way as functions for nonaggregate UDFs. See Section 22.3.2.3, “UDF Argument Processing”.

The pointer arguments to is_null and error are the same for all calls to xxx_reset(), xxx_clear(), xxx_add() and xxx(). You can use this to remember that you got an error or whether the xxx() function should return NULL. You should not store a string into *error! error points to a single-byte variable, not to a string buffer.

*is_null is reset for each group (before calling xxx_clear()). *error is never reset.

If *is_null or *error are set when xxx() returns, MySQL returns NULL as the result for the group function.

22.3.2.3 UDF Argument Processing

The args parameter points to a UDF_ARGS structure that has the members listed here:

- unsigned int arg_count

  The number of arguments. Check this value in the initialization function if you require your function to be called with a particular number of arguments. For example:

```
if (args->arg_count != 2)
{
    strcpy(message,"XXX() requires two arguments");
    return 1;
}
```

  For other UDF_ARGS member values that are arrays, array references are zero-based. That is, refer to array members using index values from 0 to args->arg_count – 1.

- enum Item_result *arg_type

  A pointer to an array containing the types for each argument. The possible type values are STRING_RESULT, INT_RESULT, REAL_RESULT, and DECIMAL_RESULT.

  To make sure that arguments are of a given type and return an error if they are not, check the arg_type array in the initialization function. For example:

```
if (args->arg_type[0] != STRING_RESULT ||
    args->arg_type[1] != INT_RESULT)
{
    strcpy(message,"XXX() requires a string and an integer");
    return 1;
}
```

  Arguments of type DECIMAL_RESULT are passed as strings, so you should handle them the same way as STRING_RESULT values.

  As an alternative to requiring your function's arguments to be of particular types, you can use the initialization function to set the arg_type elements to the types you want. This causes MySQL to coerce arguments to those types for each call to xxx(). For example, to specify that the first two arguments should be coerced to string and integer, respectively, do this in xxx_init():

```
args->arg_type[0] = STRING_RESULT;
```

```
args->arg_type[1] = INT_RESULT;
```

Exact-value decimal arguments such as 1.3 or DECIMAL column values are passed with a type of DECIMAL_RESULT. However, the values are passed as strings. If you want to receive a number, use the initialization function to specify that the argument should be coerced to a REAL_RESULT value:

```
args->arg_type[2] = REAL_RESULT;
```

- char **args

  args->args communicates information to the initialization function about the general nature of the arguments passed to your function. For a constant argument i, args->args[i] points to the argument value. (See later for instructions on how to access the value properly.) For a nonconstant argument, args->args[i] is 0. A constant argument is an expression that uses only constants, such as 3 or 4*7-2 or SIN(3.14). A nonconstant argument is an expression that refers to values that may change from row to row, such as column names or functions that are called with nonconstant arguments.

  For each invocation of the main function, args->args contains the actual arguments that are passed for the row currently being processed.

  If argument i represents NULL, args->args[i] is a null pointer (0). If the argument is not NULL, functions can refer to it as follows:

  - An argument of type STRING_RESULT is given as a string pointer plus a length, to enable handling of binary data or data of arbitrary length. The string contents are available as args->args[i] and the string length is args->lengths[i]. Do not assume that the string is null-terminated.
  - For an argument of type INT_RESULT, you must cast args->args[i] to a long long value:

    ```
    long long int_val;
    int_val = *((long long*) args->args[i]);
    ```

  - For an argument of type REAL_RESULT, you must cast args->args[i] to a double value:

    ```
    double    real_val;
    real_val = *((double*) args->args[i]);
    ```

  - For an argument of type DECIMAL_RESULT, the value is passed as a string and should be handled like a STRING_RESULT value.
  - ROW_RESULT arguments are not implemented.

- unsigned long *lengths

  For the initialization function, the lengths array indicates the maximum string length for each argument. You should not change these. For each invocation of the main function, lengths contains the actual lengths of any string arguments that are passed for the row currently being processed. For arguments of types INT_RESULT or REAL_RESULT, lengths still contains the maximum length of the argument (as for the initialization function).

- char *maybe_null

  For the initialization function, the maybe_null array indicates for each argument whether the argument value might be null (0 if no, 1 if yes).

- char **attributes

args->attributes communicates information about the names of the UDF arguments. For argument i, the attribute name is available as a string in args->attributes[i] and the attribute length is args->attribute_lengths[i]. Do not assume that the string is null-terminated.

By default, the name of a UDF argument is the text of the expression used to specify the argument. For UDFs, an argument may also have an optional [AS] *alias_name* clause, in which case the argument name is *alias_name*. The attributes value for each argument thus depends on whether an alias was given.

Suppose that a UDF my_udf() is invoked as follows:

```
SELECT my_udf(expr1, expr2 AS alias1, expr3 alias2);
```

In this case, the attributes and attribute_lengths arrays will have these values:

```
args->attributes[0] = "expr1"
args->attribute_lengths[0] = 5

args->attributes[1] = "alias1"
args->attribute_lengths[1] = 6

args->attributes[2] = "alias2"
args->attribute_lengths[2] = 6
```

- unsigned long *attribute_lengths

  The attribute_lengths array indicates the length of each argument name.

22.3.2.4 UDF Return Values and Error Handling

The initialization function should return 0 if no error occurred and 1 otherwise. If an error occurs, xxx_init() should store a null-terminated error message in the message parameter. The message is returned to the client. The message buffer is MYSQL_ERRMSG_SIZE characters long, but you should try to keep the message to less than 80 characters so that it fits the width of a standard terminal screen.

The return value of the main function xxx() is the function value, for long long and double functions. A string function should return a pointer to the result and set *length to the length (in bytes) of the return value. For example:

```
memcpy(result, "result string", 13);
*length = 13;
```

MySQL passes a buffer to the xxx() function using the result parameter. This buffer is sufficiently long to hold 255 characters, which can be multi-byte characters. The xxx() function can store the result in this buffer if it fits, in which case the return value should be a pointer to the buffer. If the function stores the result in a different buffer, it should return a pointer to that buffer.

If your string function does not use the supplied buffer (for example, if it needs to return a string longer than 255 characters), you must allocate the space for your own buffer with malloc() in your xxx_init() function or your xxx() function and free it in your xxx_deinit() function. You can store the allocated memory in the ptr slot in the UDF_INIT structure for reuse by future xxx() calls. See Section 22.3.2.1, “UDF Calling Sequences for Simple Functions”.

To indicate a return value of NULL in the main function, set *is_null to 1:

```
*is_null = 1;
```

To indicate an error return in the main function, set `*error` to `1`:

```
*error = 1;
```

If `xxx()` sets `*error` to `1` for any row, the function value is `NULL` for the current row and for any subsequent rows processed by the statement in which `XXX()` was invoked. (`xxx()` is not even called for subsequent rows.)

22.3.2.5 Compiling and Installing User-Defined Functions

Files implementing UDFs must be compiled and installed on the host where the server runs. This process is described below for the example UDF file `sql/udf_example.c` that is included in MySQL source distributions.

If a UDF will be referred to in statements that will be replicated to slave servers, you must ensure that every slave also has the function available. Otherwise, replication will fail on the slaves when they attempt to invoke the function.

The immediately following instructions are for Unix. Instructions for Windows are given later in this section.

The `udf_example.c` file contains the following functions:

- `metaphon()` returns a metaphon string of the string argument. This is something like a soundex string, but it is more tuned for English.
- `myfunc_double()` returns the sum of the ASCII values of the characters in its arguments, divided by the sum of the length of its arguments.
- `myfunc_int()` returns the sum of the length of its arguments.
- `sequence([const int])` returns a sequence starting from the given number or 1 if no number has been given.
- `lookup()` returns the IP address for a host name.
- `reverse_lookup()` returns the host name for an IP address. The function may be called either with a single string argument of the form `'xxx.xxx.xxx.xxx'` or with four numbers.
- `avgcost()` returns an average cost. This is an aggregate function.

A dynamically loadable file should be compiled as a sharable object file, using a command something like this:

```
shell> gcc -shared -o udf_example.so udf_example.c
```

If you are using `gcc` with `CMake` (which is how MySQL is configured), you should be able to create `udf_example.so` with a simpler command:

```
shell> make udf_example
```

After you compile a shared object containing UDFs, you must install it and tell MySQL about it. Compiling a shared object from `udf_example.c` using `gcc` directly produces a file named `udf_example.so`. Copy the shared object to the server's plugin directory and name it `udf_example.so`. This directory is given by the value of the `plugin_dir` system variable.

On some systems, the `ldconfig` program that configures the dynamic linker does not recognize a shared object unless its name begins with `lib`. In this case you should rename a file such as `udf_example.so` to `libudf_example.so`.

On Windows, you can compile user-defined functions by using the following procedure:

1. Obtain a MySQL source distribution. See Section 2.1.3, “How to Get MySQL”.
2. Obtain the `CMake` build utility, if necessary, from http://www.cmake.org. (Version 2.6 or later is required).
3. In the source tree, look in the `sql` directory. There are files named `udf_example.def` `udf_example.c` there. Copy both files from this directory to your working directory.
4. Create a `CMake` `makefile` (`CMakeLists.txt`) with these contents:

   ```
   PROJECT(udf_example)

   # Path for MySQL include directory
   INCLUDE_DIRECTORIES("c:/mysql/include")

   ADD_DEFINITIONS("-DHAVE_DLOPEN")
   ADD_LIBRARY(udf_example MODULE udf_example.c udf_example.def)
   TARGET_LINK_LIBRARIES(udf_example wsock32)
   ```

5. Create the VC project and solution files:

   ```
   cmake -G "<Generator>"
   ```

   Invoking `cmake --help` shows you a list of valid Generators.

6. Create `udf_example.dll`:

   ```
   devenv udf_example.sln /build Release
   ```

After the shared object file has been installed, notify `mysqld` about the new functions with the following statements. If object files have a suffix different from `.so` on your system, substitute the correct suffix throughout (for example, `.dll` on Windows).

```
mysql> CREATE FUNCTION metaphon RETURNS STRING SONAME 'udf_example.so';
mysql> CREATE FUNCTION myfunc_double RETURNS REAL SONAME 'udf_example.so';
mysql> CREATE FUNCTION myfunc_int RETURNS INTEGER SONAME 'udf_example.so';
mysql> CREATE FUNCTION sequence RETURNS INTEGER SONAME 'udf_example.so';
mysql> CREATE FUNCTION lookup RETURNS STRING SONAME 'udf_example.so';
mysql> CREATE FUNCTION reverse_lookup
    ->        RETURNS STRING SONAME 'udf_example.so';
mysql> CREATE AGGREGATE FUNCTION avgcost
    ->        RETURNS REAL SONAME 'udf_example.so';
```

To delete functions, use `DROP FUNCTION`:

```
mysql> DROP FUNCTION metaphon;
mysql> DROP FUNCTION myfunc_double;
mysql> DROP FUNCTION myfunc_int;
mysql> DROP FUNCTION sequence;
mysql> DROP FUNCTION lookup;
mysql> DROP FUNCTION reverse_lookup;
mysql> DROP FUNCTION avgcost;
```

The CREATE FUNCTION and DROP FUNCTION statements update the func system table in the mysql database. The function's name, type and shared library name are saved in the table. You must have the INSERT or DELETE privilege for the mysql database to create or drop functions, respectively.

You should not use CREATE FUNCTION to add a function that has previously been created. If you need to reinstall a function, you should remove it with DROP FUNCTION and then reinstall it with CREATE FUNCTION. You would need to do this, for example, if you recompile a new version of your function, so that mysqld gets the new version. Otherwise, the server continues to use the old version.

An active function is one that has been loaded with CREATE FUNCTION and not removed with DROP FUNCTION. All active functions are reloaded each time the server starts, unless you start mysqld with the --skip-grant-tables option. In this case, UDF initialization is skipped and UDFs are unavailable.

22.3.2.6 User-Defined Function Security Precautions

MySQL takes several measures to prevent misuse of user-defined functions.

UDF object files cannot be placed in arbitrary directories. They must be located in the server's plugin directory. This directory is given by the value of the plugin_dir system variable.

To use CREATE FUNCTION or DROP FUNCTION, you must have the INSERT or DELETE privilege, respectively, for the mysql database. This is necessary because those statements add and delete rows from the mysql.func table.

UDFs should have at least one symbol defined in addition to the xxx symbol that corresponds to the main xxx() function. These auxiliary symbols correspond to the xxx_init(), xxx_deinit(), xxx_reset(), xxx_clear(), and xxx_add() functions. mysqld also supports an --allow-suspicious-udfs option that controls whether UDFs that have only an xxx symbol can be loaded. By default, the option is off, to prevent attempts at loading functions from shared object files other than those containing legitimate UDFs. If you have older UDFs that contain only the xxx symbol and that cannot be recompiled to include an auxiliary symbol, it may be necessary to specify the --allow-suspicious-udfs option. Otherwise, you should avoid enabling this capability.

22.3.3 Adding a New Native Function

To add a new native MySQL function, use the procedure described here, which requires that you use a source distribution. You cannot add native functions to a binary distribution because it is necessary to modify MySQL source code and compile MySQL from the modified source. If you migrate to another version of MySQL (for example, when a new version is released), you must repeat the procedure with the new version.

If the new native function will be referred to in statements that will be replicated to slave servers, you must ensure that every slave server also has the function available. Otherwise, replication will fail on the slaves when they attempt to invoke the function.

To add a new native function, follow these steps to modify source files in the sql directory:

1. Create a subclass for the function in item_create.cc:

   - If the function takes a fixed number of arguments, create a subclass of Create_func_arg0, Create_func_arg1, Create_func_arg2, or Create_func_arg3, respectively, depending on whether the function takes zero, one, two, or three arguments. For examples, see the Create_func_uuid, Create_func_abs, Create_func_pow, and Create_func_lpad classes.

   - If the function takes a variable number of arguments, create a subclass of Create_native_func. For an example, see Create_func_concat.

2. To provide a name by which the function can be referred to in SQL statements, register the name in `item_create.cc` by adding a line to this array:

   ```
   static Native_func_registry func_array[]
   ```

   You can register several names for the same function. For example, see the lines for `"LCASE"` and `"LOWER"`, which are aliases for `Create_func_lcase`.

3. In `item_func.h`, declare a class inheriting from `Item_num_func` or `Item_str_func`, depending on whether your function returns a number or a string.

4. In `item_func.cc`, add one of the following declarations, depending on whether you are defining a numeric or string function:

   ```
   double   Item_func_newname::val()
   longlong Item_func_newname::val_int()
   String  *Item_func_newname::Str(String *str)
   ```

   If you inherit your object from any of the standard items (like `Item_num_func`), you probably only have to define one of these functions and let the parent object take care of the other functions. For example, the `Item_str_func` class defines a `val()` function that executes `atof()` on the value returned by `::str()`.

5. If the function is nondeterministic, include the following statement in the item constructor to indicate that function results should not be cached:

   ```
   current_thd->lex->safe_to_cache_query=0;
   ```

   A function is nondeterministic if, given fixed values for its arguments, it can return different results for different invocations.

6. You should probably also define the following object function:

   ```
   void Item_func_newname::fix_length_and_dec()
   ```

   This function should at least calculate `max_length` based on the given arguments. `max_length` is the maximum number of characters the function may return. This function should also set `maybe_null = 0` if the main function can't return a `NULL` value. The function can check whether any of the function arguments can return `NULL` by checking the arguments' `maybe_null` variable. Look at `Item_func_mod::fix_length_and_dec` for a typical example of how to do this.

All functions must be thread-safe. In other words, do not use any global or static variables in the functions without protecting them with mutexes.

If you want to return `NULL` from `::val()`, `::val_int()`, or `::str()`, you should set `null_value` to 1 and return 0.

For `::str()` object functions, there are additional considerations to be aware of:

- The `String *str` argument provides a string buffer that may be used to hold the result. (For more information about the `String` type, take a look at the `sql_string.h` file.)

- The `::str()` function should return the string that holds the result, or `(char*) 0` if the result is `NULL`.

- All current string functions try to avoid allocating any memory unless absolutely necessary!

22.4 Debugging and Porting MySQL

This section helps you port MySQL to other operating systems. Do check the list of currently supported operating systems first. See http://www.mysql.com/support/supportedplatforms/database.html. If you have created a new port of MySQL, please let us know so that we can list it here and on our Web site (http://www.mysql.com/), recommending it to other users.

Note

If you create a new port of MySQL, you are free to copy and distribute it under the GPL license, but it does not make you a copyright holder of MySQL.

A working POSIX thread library is needed for the server.

To build MySQL from source, your system must satisfy the tool requirements listed at Section 2.8, “Installing MySQL from Source”.

If you run into problems with a new port, you may have to do some debugging of MySQL! See Section 22.4.1, “Debugging a MySQL Server”.

Note

Before you start debugging `mysqld`, first get the test program `mysys/thr_lock` to work. This ensures that your thread installation has even a remote chance to work!

22.4.1 Debugging a MySQL Server

If you are using some functionality that is very new in MySQL, you can try to run `mysqld` with the `--skip-new` (which disables all new, potentially unsafe functionality). See Section C.5.4.2, “What to Do If MySQL Keeps Crashing”.

If `mysqld` doesn't want to start, you should verify that you don't have any `my.cnf` files that interfere with your setup! You can check your `my.cnf` arguments with `mysqld --print-defaults` and avoid using them by starting with `mysqld --no-defaults ...`.

If `mysqld` starts to eat up CPU or memory or if it “hangs,” you can use `mysqladmin processlist status` to find out if someone is executing a query that takes a long time. It may be a good idea to run `mysqladmin -i10 processlist status` in some window if you are experiencing performance problems or problems when new clients can't connect.

The command `mysqladmin debug` dumps some information about locks in use, used memory and query usage to the MySQL log file. This may help solve some problems. This command also provides some useful information even if you haven't compiled MySQL for debugging!

If the problem is that some tables are getting slower and slower you should try to optimize the table with `OPTIMIZE TABLE` or `myisamchk`. See Chapter 5, *MySQL Server Administration*. You should also check the slow queries with `EXPLAIN`.

You should also read the OS-specific section in this manual for problems that may be unique to your environment. See Section 2.1, “General Installation Guidance”.

22.4.1.1 Compiling MySQL for Debugging

If you have some very specific problem, you can always try to debug MySQL. To do this you must configure MySQL with the `-DWITH_DEBUG=1` option. You can check whether MySQL was compiled with debugging by doing: mysqld --help. If the --debug flag is listed with the options then you have debugging enabled. mysqladmin ver also lists the mysqld version as mysql ... --debug in this case.

If mysqld stops crashing when you configure it with the -DWITH_DEBUG=1 CMake option, you probably have found a compiler bug or a timing bug within MySQL. In this case, you can try to add -g using the CMAKE_C_FLAGS and CMAKE_CXX_FLAGS CMake options and not use -DWITH_DEBUG=1. If mysqld dies, you can at least attach to it with gdb or use gdb on the core file to find out what happened.

When you configure MySQL for debugging you automatically enable a lot of extra safety check functions that monitor the health of mysqld. If they find something “unexpected,” an entry is written to stderr, which mysqld_safe directs to the error log! This also means that if you are having some unexpected problems with MySQL and are using a source distribution, the first thing you should do is to configure MySQL for debugging! (The second thing is to send mail to a MySQL mailing list and ask for help. See Section 1.6.1, “MySQL Mailing Lists”. If you believe that you have found a bug, please use the instructions at Section 1.7, “How to Report Bugs or Problems”.

In the Windows MySQL distribution, mysqld.exe is by default compiled with support for trace files.

22.4.1.2 Creating Trace Files

If the mysqld server doesn't start or if you can cause it to crash quickly, you can try to create a trace file to find the problem.

To do this, you must have a mysqld that has been compiled with debugging support. You can check this by executing mysqld -V. If the version number ends with -debug, it is compiled with support for trace files. (On Windows, the debugging server is named mysqld-debug rather than mysqld as of MySQL 4.1.)

Start the mysqld server with a trace log in /tmp/mysqld.trace on Unix or \mysqld.trace on Windows:

```
shell> mysqld --debug
```

On Windows, you should also use the --standalone flag to not start mysqld as a service. In a console window, use this command:

```
C:\> mysqld-debug --debug --standalone
```

After this, you can use the mysql.exe command-line tool in a second console window to reproduce the problem. You can stop the mysqld server with mysqladmin shutdown.

The trace file can become **very large**! To generate a smaller trace file, you can use debugging options something like this:

mysqld --debug=d,info,error,query,general,where:O,/tmp/mysqld.trace

This only prints information with the most interesting tags to the trace file.

If you make a bug report about this, please only send the lines from the trace file to the appropriate mailing list where something seems to go wrong! If you can't locate the wrong place, you can open a bug report and upload the trace file to the report, so that a MySQL developer can take a look at it. For instructions, see Section 1.7, “How to Report Bugs or Problems”.

The trace file is made with the **DBUG** package by Fred Fish. See Section 22.4.3, “The DBUG Package”.

22.4.1.3 Using pdb to create a Windows crashdump

Program Database files (extension pdb) are included in the Noinstall distribution of MySQL. These files provide information for debugging your MySQL installation in the event of a problem.

The PDB file contains more detailed information about mysqld and other tools that enables more detailed trace and dump files to be created. You can use these with Dr Watson, WinDbg and Visual Studio to debug mysqld.

For more information on PDB files, see Microsoft Knowledge Base Article 121366. For more information on the debugging options available, see Debugging Tools for Windows.

Dr Watson is installed with all Windows distributions, but if you have installed Windows development tools, Dr Watson may have been replaced with WinDbg, the debugger included with Visual Studio, or the debugging tools provided with Borland or Delphi.

To generate a crash file using Dr Watson, follow these steps:

1. Start Dr Watson by running drwtsn32.exe interactively using the -i option:

   ```
   C:\> drwtsn32 -i
   ```

2. Set the **Log File Path** to the directory where you want to store trace files.

3. Make sure **Dump All Thread Contexts** and **Append To Existing Log File**.

4. Uncheck **Dump Symbol Table**, **Visual Notification**, **Sound Notification** and **Create Crash Dump File**.

5. Set the **Number of Instructions** to a suitable value to capture enough calls in the stacktrace. A value of at 25 should be enough.

Note that the file generated can become very large.

22.4.1.4 Debugging mysqld under gdb

On most systems you can also start mysqld from gdb to get more information if mysqld crashes.

With some older gdb versions on Linux you must use run --one-thread if you want to be able to debug mysqld threads. In this case, you can only have one thread active at a time. It is best to upgrade to gdb 5.1 because thread debugging works much better with this version!

NPTL threads (the new thread library on Linux) may cause problems while running mysqld under gdb. Some symptoms are:

- mysqld hangs during startup (before it writes ready for connections).
- mysqld crashes during a pthread_mutex_lock() or pthread_mutex_unlock() call.

In this case, you should set the following environment variable in the shell before starting gdb:

```
LD_ASSUME_KERNEL=2.4.1
export LD_ASSUME_KERNEL
```

When running mysqld under gdb, you should disable the stack trace with --skip-stack-trace to be able to catch segfaults within gdb.

In MySQL 4.0.14 and above you should use the `--gdb` option to `mysqld`. This installs an interrupt handler for `SIGINT` (needed to stop `mysqld` with `^C` to set breakpoints) and disable stack tracing and core file handling.

It is very hard to debug MySQL under `gdb` if you do a lot of new connections the whole time as `gdb` doesn't free the memory for old threads. You can avoid this problem by starting `mysqld` with `thread_cache_size` set to a value equal to `max_connections` + 1. In most cases just using `--thread_cache_size=5'` helps a lot!

If you want to get a core dump on Linux if `mysqld` dies with a SIGSEGV signal, you can start `mysqld` with the `--core-file` option. This core file can be used to make a backtrace that may help you find out why `mysqld` died:

```
shell> gdb mysqld core
gdb>   backtrace full
gdb>   quit
```

See Section C.5.4.2, “What to Do If MySQL Keeps Crashing”.

If you are using `gdb` 4.17.x or above on Linux, you should install a `.gdb` file, with the following information, in your current directory:

```
set print sevenbit off
handle SIGUSR1 nostop noprint
handle SIGUSR2 nostop noprint
handle SIGWAITING nostop noprint
handle SIGLWP nostop noprint
handle SIGPIPE nostop
handle SIGALRM nostop
handle SIGHUP nostop
handle SIGTERM nostop noprint
```

If you have problems debugging threads with `gdb`, you should download gdb 5.x and try this instead. The new `gdb` version has very improved thread handling!

Here is an example how to debug `mysqld`:

```
shell> gdb /usr/local/libexec/mysqld
gdb> run
...
backtrace full # Do this when mysqld crashes
```

Include the preceding output in a bug report, which you can file using the instructions in Section 1.7, “How to Report Bugs or Problems”.

If `mysqld` hangs, you can try to use some system tools like `strace` or `/usr/proc/bin/pstack` to examine where `mysqld` has hung.

```
strace /tmp/log libexec/mysqld
```

If you are using the Perl `DBI` interface, you can turn on debugging information by using the `trace` method or by setting the `DBI_TRACE` environment variable.

22.4.1.5 Using a Stack Trace

On some operating systems, the error log contains a stack trace if `mysqld` dies unexpectedly. You can use this to find out where (and maybe why) `mysqld` died. See Section 5.2.2, “The Error Log”. To get a stack trace, you must not compile mysqld with the -fomit-frame-pointer option to gcc. See Section 22.4.1.1, “Compiling MySQL for Debugging”.

A stack trace in the error log looks something like this:

```
mysqld got signal 11;
Attempting backtrace. You can use the following information
to find out where mysqld died. If you see no messages after
this, something went terribly wrong...

stack_bottom = 0x41fd0110 thread_stack 0x40000
mysqld(my_print_stacktrace+0x32)[0x9da402]
mysqld(handle_segfault+0x28a)[0x6648e9]
/lib/libpthread.so.0[0x7f1a5af000f0]
/lib/libc.so.6(strcmp+0x2)[0x7f1a5a10f0f2]
mysqld(_Z21check_change_passwordP3THDPKcS2_Pcj+0x7c)[0x7412cb]
mysqld(_ZN16set_var_password5checkEP3THD+0xd0)[0x688354]
mysqld(_Z17sql_set_variablesP3THDP4ListI12set_var_baseE+0x68)[0x688494]
mysqld(_Z21mysql_execute_commandP3THD+0x41a0)[0x67a170]
mysqld(_Z11mysql_parseP3THDPKcjPS2_+0x282)[0x67f0ad]
mysqld(_Z16dispatch_command19enum_server_commandP3THDPcj+0xbb7[0x67fdf8]
mysqld(_Z10do_commandP3THD+0x24d)[0x6811b6]
mysqld(handle_one_connection+0x11c)[0x66e05e]
```

If resolution of function names for the trace fails, the trace contains less information:

```
mysqld got signal 11;
Attempting backtrace. You can use the following information
to find out where mysqld died. If you see no messages after
this, something went terribly wrong...

stack_bottom = 0x41fd0110 thread_stack 0x40000
[0x9da402]
[0x6648e9]
[0x7f1a5af000f0]
[0x7f1a5a10f0f2]
[0x7412cb]
[0x688354]
[0x688494]
[0x67a170]
[0x67f0ad]
[0x67fdf8]
[0x6811b6]
[0x66e05e]
```

In the latter case, you can use the resolve_stack_dump utility to determine where mysqld died by using the following procedure:

1. Copy the numbers from the stack trace to a file, for example mysqld.stack. The numbers should not include the surrounding square brackets:

   ```
   0x9da402
   0x6648e9
   0x7f1a5af000f0
   0x7f1a5a10f0f2
   0x7412cb
   0x688354
   0x688494
   0x67a170
   0x67f0ad
   0x67fdf8
   0x6811b6
   ```

```
0x66e05e
```

2. Make a symbol file for the mysqld server:

   ```
   shell> nm -n libexec/mysqld > /tmp/mysqld.sym
   ```

   If mysqld is not linked statically, use the following command instead:

   ```
   shell> nm -D -n libexec/mysqld > /tmp/mysqld.sym
   ```

   If you want to decode C++ symbols, use the --demangle, if available, to nm. If your version of nm does not have this option, you will need to use the c++filt command after the stack dump has been produced to demangle the C++ names.

3. Execute the following command:

   ```
   shell> resolve_stack_dump -s /tmp/mysqld.sym -n mysqld.stack
   ```

   If you were not able to include demangled C++ names in your symbol file, process the resolve_stack_dump output using c++filt:

   ```
   shell> resolve_stack_dump -s /tmp/mysqld.sym -n mysqld.stack | c++filt
   ```

   This prints out where mysqld died. If that does not help you find out why mysqld died, you should create a bug report and include the output from the preceding command with the bug report.

   However, in most cases it does not help us to have just a stack trace to find the reason for the problem. To be able to locate the bug or provide a workaround, in most cases we need to know the statement that killed mysqld and preferably a test case so that we can repeat the problem! See Section 1.7, “How to Report Bugs or Problems”.

Newer versions of glibc stack trace functions also print the address as relative to the object. On glibc-based systems (Linux), the trace for a crash within a plugin looks something like:

```
plugin/auth/auth_test_plugin.so(+0x9a6)[0x7ff4d11c29a6]
```

To translate the relative address (+0x9a6) into a file name and line number, use this command:

```
shell> addr2line -fie auth_test_plugin.so 0x9a6
auth_test_plugin
mysql-trunk/plugin/auth/test_plugin.c:65
```

The addr2line utility is part of the binutils package on Linux.

On Solaris, the procedure is similar. The Solaris printstack() already prints relative addresses:

```
plugin/auth/auth_test_plugin.so:0x1510
```

To translate, use this command:

```
shell> gaddr2line -fie auth_test_plugin.so 0x1510
mysql-trunk/plugin/auth/test_plugin.c:88
```

Windows already prints the address, function name and line:

```
000007FEF07E10A4 auth_test_plugin.dll!auth_test_plugin()[test_plugin.c:72]
```

22.4.1.6 Using Server Logs to Find Causes of Errors in mysqld

Note that before starting mysqld with the general query log enabled, you should check all your tables with myisamchk. See Chapter 5, *MySQL Server Administration*.

If mysqld dies or hangs, you should start mysqld with the general query log enabled. See Section 5.2.3, “The General Query Log”. When mysqld dies again, you can examine the end of the log file for the query that killed mysqld.

If you use the default general query log file, the log is stored in the database directory as *host_name*.log In most cases it is the last query in the log file that killed mysqld, but if possible you should verify this by restarting mysqld and executing the found query from the mysql command-line tools. If this works, you should also test all complicated queries that didn't complete.

You can also try the command EXPLAIN on all SELECT statements that takes a long time to ensure that mysqld is using indexes properly. See Section 13.8.2, “EXPLAIN Syntax”.

You can find the queries that take a long time to execute by starting mysqld with the slow query log enabled. See Section 5.2.5, “The Slow Query Log”.

If you find the text mysqld restarted in the error log file (normally named hostname.err) you probably have found a query that causes mysqld to fail. If this happens, you should check all your tables with myisamchk (see Chapter 5, *MySQL Server Administration*), and test the queries in the MySQL log files to see whether one fails. If you find such a query, try first upgrading to the newest MySQL version. If this doesn't help and you can't find anything in the mysql mail archive, you should report the bug to a MySQL mailing list. The mailing lists are described at http://lists.mysql.com/, which also has links to online list archives.

If you have started mysqld with --myisam-recover-options, MySQL automatically checks and tries to repair MyISAM tables if they are marked as 'not closed properly' or 'crashed'. If this happens, MySQL writes an entry in the hostname.err file 'Warning: Checking table ...' which is followed by Warning: Repairing table if the table needs to be repaired. If you get a lot of these errors, without mysqld having died unexpectedly just before, then something is wrong and needs to be investigated further. See Section 5.1.3, “Server Command Options”.

In MySQL 5.7, when the server detects MyISAM table corruption, it writes additional information to the error log, such as the name and line number of the source file, and the list of threads accessing the table. Example: Got an error from thread_id=1, mi_dynrec.c:368. This is useful information to include in bug reports.

It is not a good sign if mysqld did die unexpectedly, but in this case, you should not investigate the Checking table... messages, but instead try to find out why mysqld died.

22.4.1.7 Making a Test Case If You Experience Table Corruption

If you get corrupted tables or if mysqld always fails after some update commands, you can test whether this bug is reproducible by doing the following:

- Take down the MySQL daemon (with mysqladmin shutdown).
- Make a backup of the tables (to guard against the very unlikely case that the repair does something bad).

- Check all tables with `myisamchk -s database/*.MYI`. Repair any wrong tables with `myisamchk -r database/`*`table`*`.MYI`.
- Make a second backup of the tables.
- Remove (or move away) any old log files from the MySQL data directory if you need more space.
- Start `mysqld` with the binary log enabled. If you want to find a query that crashes `mysqld`, you should start the server with both the general query log enabled as well. See Section 5.2.3, “The General Query Log”, and Section 5.2.4, “The Binary Log”.
- When you have gotten a crashed table, stop the `mysqld server`.
- Restore the backup.
- Restart the `mysqld` server **without** the binary log enabled.
- Re-execute the commands with `mysqlbinlog binary-log-file | mysql`. The binary log is saved in the MySQL database directory with the name `hostname-bin.`*`NNNNNN`*.
- If the tables are corrupted again or you can get `mysqld` to die with the above command, you have found reproducible bug that should be easy to fix! FTP the tables and the binary log to our bugs database using the instructions given in Section 1.7, “How to Report Bugs or Problems”. If you are a support customer, you can use the MySQL Customer Support Center https://support.mysql.com/ to alert the MySQL team about the problem and have it fixed as soon as possible.

22.4.2 Debugging a MySQL Client

To be able to debug a MySQL client with the integrated debug package, you should configure MySQL with `-DWITH_DEBUG=1`. See Section 2.8.4, “MySQL Source-Configuration Options”.

Before running a client, you should set the `MYSQL_DEBUG` environment variable:

```
shell> MYSQL_DEBUG=d:t:O,/tmp/client.trace
shell> export MYSQL_DEBUG
```

This causes clients to generate a trace file in `/tmp/client.trace`.

If you have problems with your own client code, you should attempt to connect to the server and run your query using a client that is known to work. Do this by running `mysql` in debugging mode (assuming that you have compiled MySQL with debugging on):

```
shell> mysql --debug=d:t:O,/tmp/client.trace
```

This provides useful information in case you mail a bug report. See Section 1.7, “How to Report Bugs or Problems”.

If your client crashes at some 'legal' looking code, you should check that your `mysql.h` include file matches your MySQL library file. A very common mistake is to use an old `mysql.h` file from an old MySQL installation with new MySQL library.

22.4.3 The DBUG Package

The MySQL server and most MySQL clients are compiled with the DBUG package originally created by Fred Fish. When you have configured MySQL for debugging, this package makes it possible to get a trace file of what the program is doing. See Section 22.4.1.2, “Creating Trace Files”.

This section summarizes the argument values that you can specify in debug options on the command line for MySQL programs that have been built with debugging support. For more information about programming with the DBUG package, see the DBUG manual in the `dbug` directory of MySQL source distributions. It's best to use a recent distribution to get the most updated DBUG manual.

The DBUG package can be used by invoking a program with the `--debug[=`*`debug_options`*`]` or `-# [`*`debug_options`*`]` option. If you specify the `--debug` or `-#` option without a *`debug_options`* value, most MySQL programs use a default value. The server default is `d:t:i:o,/tmp/mysqld.trace` on Unix and `d:t:i:O,\mysqld.trace` on Windows. The effect of this default is:

- `d`: Enable output for all debug macros
- `t`: Trace function calls and exits
- `i`: Add PID to output lines
- `o,/tmp/mysqld.trace`, `O,\mysqld.trace`: Set the debug output file.

Most client programs use a default *`debug_options`* value of `d:t:o,/tmp/`*`program_name`*`.trace`, regardless of platform.

Here are some example debug control strings as they might be specified on a shell command line:

```
--debug=d:t
--debug=d:f,main,subr1:F:L:t,20
--debug=d,input,output,files:n
--debug=d:t:i:O,\\mysqld.trace
```

For `mysqld`, it is also possible to change DBUG settings at runtime by setting the `debug` system variable. This variable has global and session values:

```
mysql> SET GLOBAL debug = 'debug_options';
mysql> SET SESSION debug = 'debug_options';
```

Changes at runtime require the `SUPER` privilege, even for the session value.

The *`debug_options`* value is a sequence of colon-separated fields:

```
field_1:field_2:...:field_N
```

Each field within the value consists of a mandatory flag character, optionally preceded by a `+` or `-` character, and optionally followed by a comma-delimited list of modifiers:

```
[+|-]flag[,modifier,modifier,...,modifier]
```

The following table describes the permitted flag characters. Unrecognized flag characters are silently ignored.

Flag	Description
`d`	Enable output from DBUG_*XXX* macros for the current state. May be followed by a list of keywords, which enables output only for the DBUG macros with that keyword. An empty list of keywords enables output for all macros. In MySQL, common debug macro keywords to enable are `enter`, `exit`, `error`, `warning`, `info`, and `loop`.

D	Delay after each debugger output line. The argument is the delay, in tenths of seconds, subject to machine capabilities. For example, D,20 specifies a delay of two seconds.
f	Limit debugging, tracing, and profiling to the list of named functions. An empty list enables all functions. The appropriate d or t flags must still be given; this flag only limits their actions if they are enabled.
F	Identify the source file name for each line of debug or trace output.
i	Identify the process with the PID or thread ID for each line of debug or trace output.
L	Identify the source file line number for each line of debug or trace output.
n	Print the current function nesting depth for each line of debug or trace output.
N	Number each line of debug output.
o	Redirect the debugger output stream to the specified file. The default output is stderr.
O	Like o, but the file is really flushed between each write. When needed, the file is closed and reopened between each write.
p	Limit debugger actions to specified processes. A process must be identified with the DBUG_PROCESS macro and match one in the list for debugger actions to occur.
P	Print the current process name for each line of debug or trace output.
r	When pushing a new state, do not inherit the previous state's function nesting level. Useful when the output is to start at the left margin.
S	Do function _sanity(_file_,_line_) at each debugged function until _sanity() returns something that differs from 0.
t	Enable function call/exit trace lines. May be followed by a list (containing only one modifier) giving a numeric maximum trace level, beyond which no output occurs for either debugging or tracing macros. The default is a compile time option.

The leading + or - character and trailing list of modifiers are used for flag characters such as d or f that can enable a debug operation for all applicable modifiers or just some of them:

- With no leading + or -, the flag value is set to exactly the modifier list as given.
- With a leading + or -, the modifiers in the list are added to or subtracted from the current modifier list.

The following examples show how this works for the d flag. An empty d list enabled output for all debug macros. A nonempty list enables output only for the macro keywords in the list.

These statements set the d value to the modifier list as given:

```
mysql> SET debug = 'd';
mysql> SELECT @@debug;
+---------+
| @@debug |
+---------+
| d       |
+---------+
mysql> SET debug = 'd,error,warning';
mysql> SELECT @@debug;
+-----------------+
| @@debug         |
+-----------------+
| d,error,warning |
+-----------------+
```

A leading + or - adds to or subtracts from the current d value:

```
mysql> SET debug = '+d,loop';
mysql> SELECT @@debug;
+----------------------+
| @@debug              |
+----------------------+
| d,error,warning,loop |
+----------------------+
mysql> SET debug = '-d,error,loop';
mysql> SELECT @@debug;
+-----------+
| @@debug   |
+-----------+
| d,warning |
+-----------+
```

Adding to “all macros enabled” results in no change:

```
mysql> SET debug = 'd';
mysql> SELECT @@debug;
+---------+
| @@debug |
+---------+
| d       |
+---------+
mysql> SET debug = '+d,loop';
mysql> SELECT @@debug;
+---------+
| @@debug |
+---------+
| d       |
+---------+
```

Disabling all enabled macros disables the `d` flag entirely:

```
mysql> SET debug = 'd,error,loop';
mysql> SELECT @@debug;
+--------------+
| @@debug      |
+--------------+
| d,error,loop |
+--------------+
mysql> SET debug = '-d,error,loop';
mysql> SELECT @@debug;
+---------+
| @@debug |
+---------+
|         |
+---------+
```

Note

Prior to MySQL 5.7.2, the `+` and `-` modifiers were not always handled correctly and could leave a flag value in an incorrect state. Verify your `debug`-setting sequence in advance or set it without using `+` or `-`.

Chapter 23 MySQL Enterprise Edition

Table of Contents

MySQL Enterprise Edition is a commercial product. Like MySQL Community Edition, MySQL Enterprise Edition includes MySQL Server, a fully integrated transaction-safe, ACID-compliant database with full commit, rollback, crash-recovery, and row-level locking capabilities. In addition, MySQL Enterprise Edition includes the following components designed to provide monitoring and online backup, as well as improved security and scalability:

The following sections briefly discuss each of these components and indicate where to find more detailed information. To learn more about commercial products, see http://www.mysql.com/products/.

- MySQL Enterprise Monitor
- MySQL Enterprise Backup
- MySQL Enterprise Security
- MySQL Enterprise Audit
- MySQL Enterprise Thread Pool

23.1 MySQL Enterprise Monitor

MySQL Enterprise Monitor is an enterprise monitoring system for MySQL that keeps an eye on your MySQL servers, notifies you of potential issues and problems, and advises you how to fix the issues. MySQL Enterprise Monitor can monitor all kinds of configurations, from a single MySQL server that is important to your business, all the way up to a huge farm of MySQL servers powering a busy web site.

The following discussion briefly summarizes the basic components that make up the MySQL Enterprise Monitor product. For more information, see the MySQL Enterprise Monitor manual, available at http://dev.mysql.com/doc/mysql-monitor/en/.

MySQL Enterprise Monitor components can be installed in various configurations depending on your database and network topology, to give you the best combination of reliable and responsive monitoring data, with minimal overhead on the database server machines. A typical MySQL Enterprise Monitor installation consists of:

- One or more MySQL servers to monitor. MySQL Enterprise Monitor can monitor both Community and Enterprise MySQL server releases.
- A MySQL Enterprise Monitor Agent for each monitored MySQL server.
- A single MySQL Enterprise Service Manager, which collates information from the agents and provides the user interface to the collected data.

MySQL Enterprise Monitor is designed to monitor one or more MySQL servers. The monitoring information is collected by using an agent, *MySQL Enterprise Monitor Agent*. The agent communicates with the

MySQL server that it monitors, collecting variables, status and health information, and sending this information to the MySQL Enterprise Service Manager. If you have multiple MySQL servers, then you have multiple MySQL Enterprise Monitor Agent processes monitoring each MySQL server.

The information collected by the agent about each MySQL server you are monitoring is sent to the *MySQL Enterprise Service Manager*. This server collates all of the information from the agents. As it collates the information sent by the agents, the MySQL Enterprise Service Manager continually tests the collected data, comparing the status of the server to reasonable values. When thresholds are reached, the server can trigger an event (including an alarm and notification) to highlight a potential issue, such as low memory, high CPU usage, or more complex conditions such insufficient buffer sizes and status information. We call each test, with its associated threshold value, a *rule*.

These rules, and the alarms and notifications, are each known as a *MySQL Enterprise Advisor*. Advisors form a critical part of the MySQL Enterprise Service Manager, as they provide warning information and troubleshooting advice about potential problems.

The MySQL Enterprise Service Manager includes a web server, and you interact with it through any web browser. This interface, the MySQL Enterprise Monitor User Interface, displays all of the information collected by the agents, and lets you view all of your servers and their current status as a group or individually. You control and configure all aspects of the service using the MySQL Enterprise Monitor User Interface.

The information supplied by the MySQL Enterprise Monitor Agent processes also includes statistical and query information, which you can view in the form of graphs. For example, you can view aspects such as server load, query numbers, or index usage information as a graph over time. The graph lets you pinpoint problems or potential issues on your server, and can help diagnose the impact from database or external problems (such as external system or network failure) by examining the data from a specific time interval.

The MySQL Enterprise Monitor Agent can also be configured to collect detailed information about the queries executed on your server, including the row counts and performance times for executing each query. You can correlate the detailed query data with the graphical information to identify which queries were executing when you experienced a particularly high load, index or other issue. The query data is supported by a system called Query Analyzer, and the data can be presented in different ways depending on your needs.

23.2 MySQL Enterprise Backup

MySQL Enterprise Backup performs hot backup operations for MySQL databases. The product is architected for efficient and reliable backups of tables created by the InnoDB storage engine. For completeness, it can also back up tables from MyISAM and other storage engines.

The following discussion briefly summarizes MySQL Enterprise Backup. For more information, see the MySQL Enterprise Backup manual, available at http://dev.mysql.com/doc/mysql-enterprise-backup/en/.

Hot backups are performed while the database is running and applications are reading and writing to it. This type of backup does not block normal database operations, and it captures even changes that occur while the backup is happening. For these reasons, hot backups are desirable when your database “grows up” -- when the data is large enough that the backup takes significant time, and when your data is important enough to your business that you must capture every last change, without taking your application, web site, or web service offline.

MySQL Enterprise Backup does a hot backup of all tables that use the InnoDB storage engine. For tables using MyISAM or other non-InnoDB storage engines, it does a “warm” backup, where the database continues to run, but those tables cannot be modified while being backed up. For efficient backup operations, you can designate InnoDB as the default storage engine for new tables, or convert existing tables to use the InnoDB storage engine.

23.3 MySQL Enterprise Security

MySQL Enterprise Edition provides plugins that implement authentication using external services:

- MySQL Enterprise Edition includes an authentication plugin that enables MySQL Server to use PAM (Pluggable Authentication Modules) to authenticate MySQL users. PAM enables a system to use a standard interface to access various kinds of authentication methods, such as Unix passwords or an LDAP directory.

- MySQL Enterprise Edition includes an authentication plugin that performs external authentication on Windows, enabling MySQL Server to use native Windows services to authenticate client connections. Users who have logged in to Windows can connect from MySQL client programs to the server based on the information in their environment without specifying an additional password.

For more information, see The PAM Authentication Plugin, and The Windows Native Authentication Plugin.

23.4 MySQL Enterprise Audit

MySQL Enterprise Edition includes MySQL Enterprise Audit, implemented using a server plugin. MySQL Enterprise Audit uses the open MySQL Audit API to enable standard, policy-based monitoring and logging of connection and query activity executed on specific MySQL servers. Designed to meet the Oracle audit specification, MySQL Enterprise Audit provides an out of box, easy to use auditing and compliance solution for applications that are governed by both internal and external regulatory guidelines.

When installed, the audit plugin enables MySQL Server to produce a log file containing an audit record of server activity. The log contents include when clients connect and disconnect, and what actions they perform while connected, such as which databases and tables they access.

For more information, see Section 6.3.13, “MySQL Enterprise Audit Log Plugin”.

23.5 MySQL Enterprise Thread Pool

MySQL Enterprise Edition includes the MySQL Thread Pool, implemented using a server plugin. The default thread-handling model in MySQL Server executes statements using one thread per client connection. As more clients connect to the server and execute statements, overall performance degrades. In MySQL Enterprise Edition, a thread pool plugin provides an alternative thread-handling model designed to reduce overhead and improve performance. The plugin implements a thread pool that increases server performance by efficiently managing statement execution threads for large numbers of client connections.

For more information, see The Thread Pool Plugin.

Chapter 24 MySQL Workbench

MySQL Workbench provides a graphical tool for working with MySQL Servers and databases. MySQL Workbench fully supports MySQL Server versions 5.1 and above. It is also compatible with MySQL Server 5.0, but not every feature of 5.0 may be supported. It does not support MySQL Server versions 4.x.

The following discussion briefly describes MySQL Workbench capabilities. For more information, see the MySQL Workbench manual, available at http://dev.mysql.com/doc/workbench/en/.

MySQL Workbench provides five main areas of functionality:

- **SQL Development**: Enables you to create and manage connections to database servers. As well as enabling you to configure connection parameters, MySQL Workbench provides the capability to execute SQL queries on the database connections using the built-in SQL Editor. This functionality replaces that previously provided by the Query Browser standalone application.

- **Data Modeling**: Enables you to create models of your database schema graphically, reverse and forward engineer between a schema and a live database, and edit all aspects of your database using the comprehensive Table Editor. The Table Editor provides easy-to-use facilities for editing Tables, Columns, Indexes, Triggers, Partitioning, Options, Inserts and Privileges, Routines and Views.

- **Server Administration**: Enables you to create and administer server instances.

- **Data Migration**: Allows you to migrate from Microsoft SQL Server, Sybase ASE, SQLite, SQL Anywhere, PostreSQL, and other RDBMS tables, objects and data to MySQL. Migration also supports migrating from earlier versions of MySQL to the latest releases.

- **MySQL Enterprise Support**: Support for Enterprise products such as MySQL Enterprise Backup and MySQL Audit.

MySQL Workbench is available in two editions, the Community Edition and the Commercial Edition. The Community Edition is available free of charge. The Commercial Edition provides additional Enterprise features, such as database documentation generation, at low cost.

Chapter 25 Introduction

Table of Contents

MySQL for Excel enables you to work with a MySQL database from within Microsoft Excel. MySQL data can be imported into Excel, Excel data can be exported into MySQL as a new table or appended to a current table, and MySQL for Excel enables you to edit the MySQL data directly from within Excel.

Visit the MySQL for Excel forum for additional MySQL for Excel help and support.

For release notes detailing the changes in each release of MySQL for Excel, see MySQL for Excel Release Notes.

25.1 Installing and Configuring

MySQL for Excel is a product for Microsoft Windows, and it is installed with MySQL Installer. And typically you will not be required to install or configure additional tools to use MySQL for Excel.

Note

To install, download and execute the MySQL Installer. Select the MySQL for Excel product and then proceed with the installation. See the MySQL Installer manual for additional details.

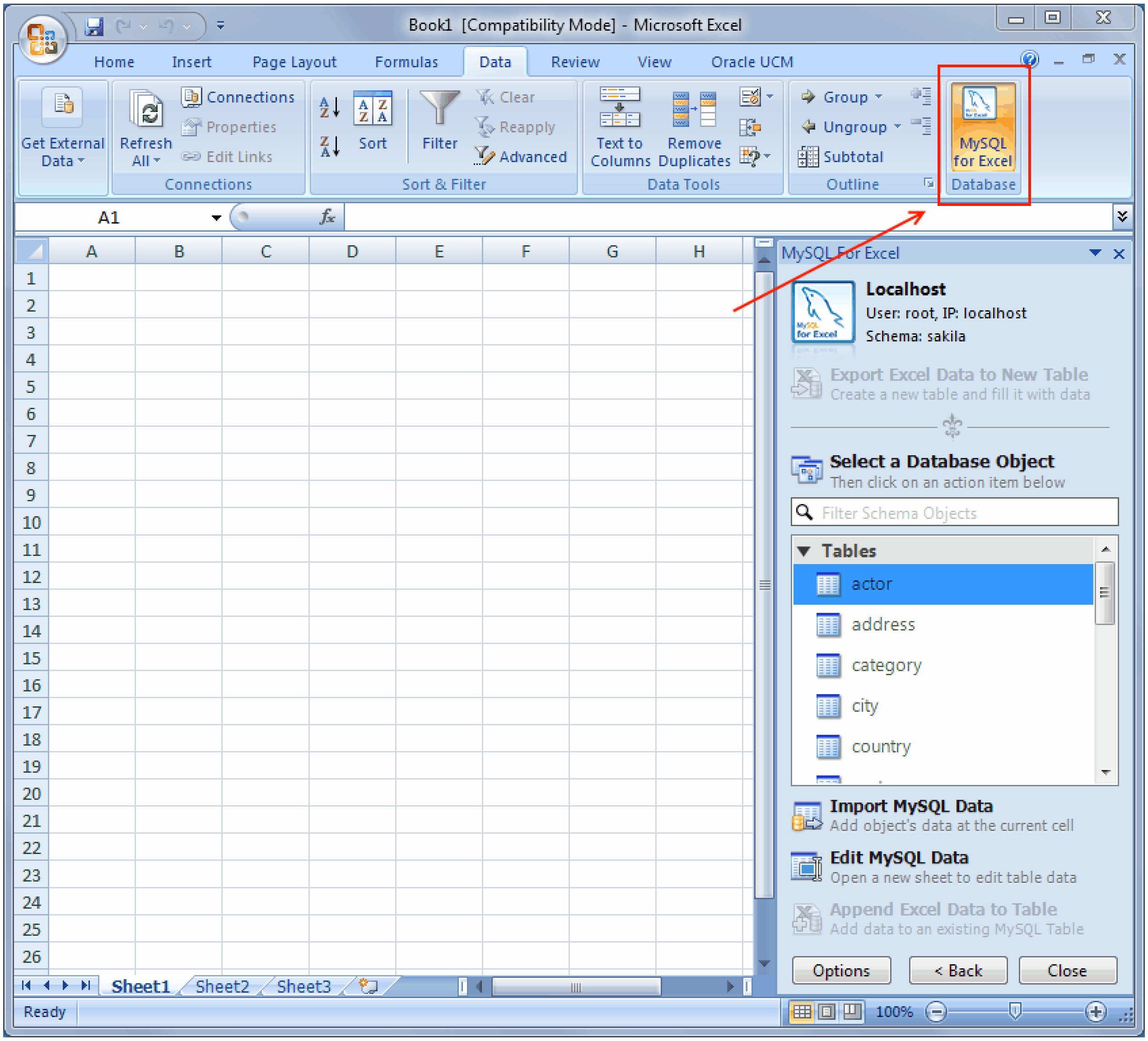
MySQL for Excel Requirements

The MySQL Installer installation process will check if these requirements are met, or notify you if further action is required before proceeding with the installation.

- .NET Framework 4.0 (Client or Full Profile).
- Microsoft Office Excel 2007 or greater, for Microsoft Windows.
- Visual Studio Tools for Office 4.0, and MySQL Installer may install this for you.
- An available MySQL Server connection.

MySQL for Excel is loaded and executed by selecting the Data menu tab in Excel, and then choosing the "MySQL for Excel" Database icon. This opens a new Excel sidebar with the available MySQL for Excel options. The navigation bar with the MySQL for Excel icon is shown in the following screenshot:

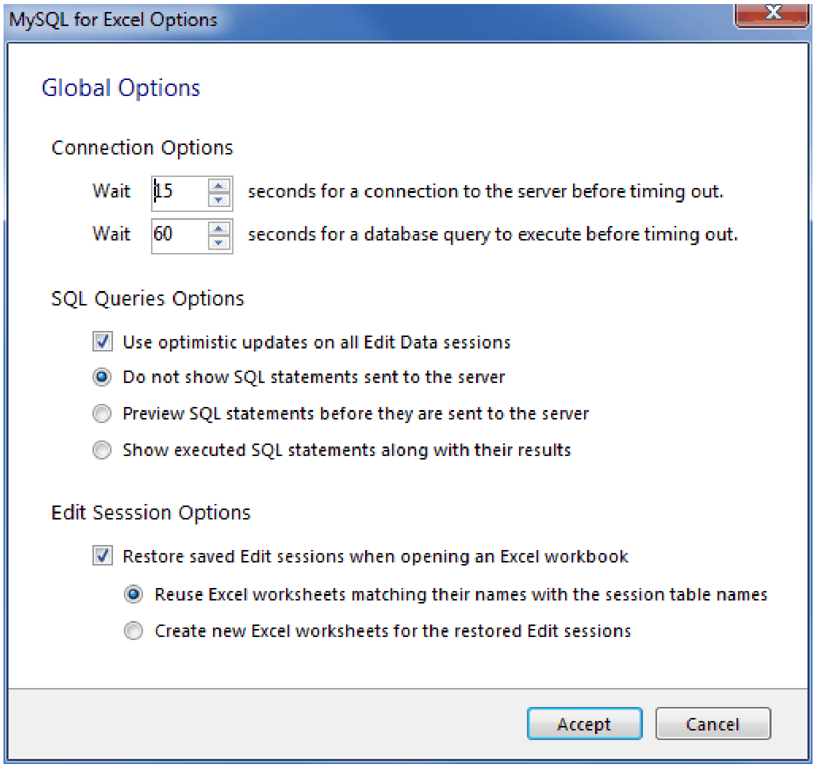
Figure 25.1 The MySQL for Excel navigation bar



Configuration

While each action, such as **Import MySQL Data**, has its own set of options, this section describes the global options that affect the entire plugin.

Figure 25.2 The MySQL for Excel configuration: Global Options



MySQL for Excel Options

Global Options

Connection Options

Wait 15 seconds for a connection to the server before timing out.

Wait 60 seconds for a database query to execute before timing out.

SQL Queries Options

☑ Use optimistic updates on all Edit Data sessions

● Do not show SQL statements sent to the server

○ Preview SQL statements before they are sent to the server

○ Show executed SQL statements along with their results

Edit Sesssion Options

☑ Restore saved Edit sessions when opening an Excel workbook

● Reuse Excel worksheets matching their names with the session table names

○ Create new Excel worksheets for the restored Edit sessions

- Connection Options:
  - Wait [ ] seconds for a connection to the server before timing out. Defaults to 15.
  - Wait [ ] seconds for a database query to execute before timing out. Defaults to 60.
- SQL Queries Options:
  - [ ] Use optimistic updates on all Edit Data sessions. Enabled by default.
  - ( ) Do not show SQL statements sent to the server. Enabled by default.
  - ( ) Preview SQL statements before they are sent to the server. Disabled by default.
  - ( ) Show executed SQL statements along with their results. Disabled by default.
- Edit Session Options:
  - [ ] Restore saved Edit sessions when opening an Excel workbook. Enabled by default.
  - ( ) Reuse Excel worksheets matching their names with the session table names. Enabled by default.
  - ( ) Create new Excel worksheets for the restored Edit sessions. Disabled by default.

25.2 Edit MySQL Data in Excel

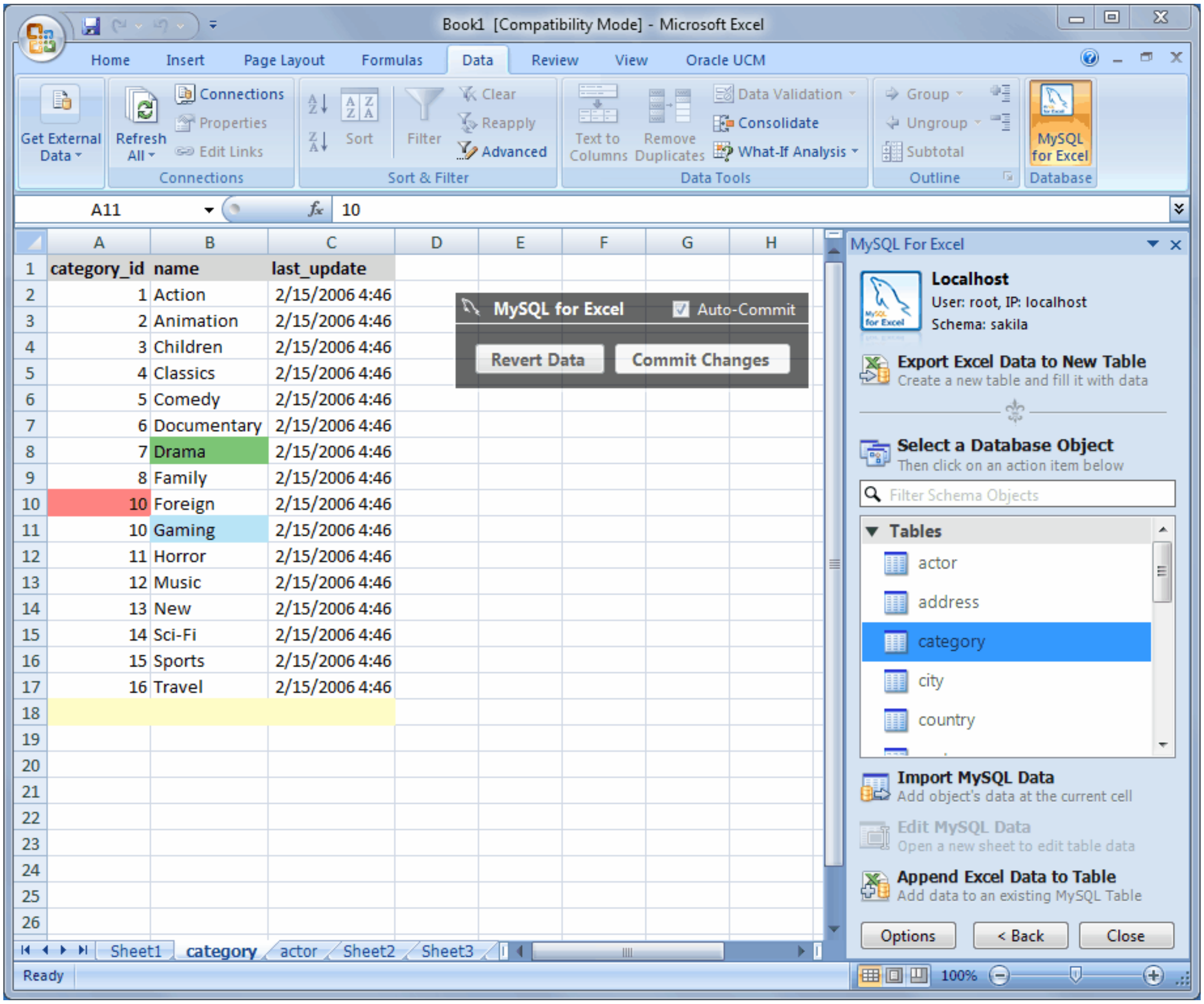
MySQL for Excel enables you to load and edit MySQL data directly from Microsoft Excel. Changes are immediately committed if the **Auto-Commit** option is enabled, or done manually by pressing Commit Changes.

The example below uses the `category` table of the example `sakila` database, but the screen will look the same for any table. Within MySQL for Excel, **Open a MySQL Connection**, click the `sakila` schema, Next, select the `category` table, click **Edit MySQL Data**, then choose Import to import the data into a new Microsoft Excel worksheet for editing.

Note

For additional information about the importing procedure, see Section 25.3, "Import MySQL Data into Excel".

Figure 25.3 Editing table data with MySQL for Excel



A11 | *fx* 10

	A	B	C
1	category_id	name	last_update
2	1	Action	2/15/2006 4:46
3	2	Animation	2/15/2006 4:46
4	3	Children	2/15/2006 4:46
5	4	Classics	2/15/2006 4:46
6	5	Comedy	2/15/2006 4:46
7	6	Documentary	2/15/2006 4:46
8	7	Drama	2/15/2006 4:46
9	8	Family	2/15/2006 4:46
10	10	Foreign	2/15/2006 4:46
11	10	Gaming	2/15/2006 4:46
12	11	Horror	2/15/2006 4:46
13	12	Music	2/15/2006 4:46
14	13	New	2/15/2006 4:46
15	14	Sci-Fi	2/15/2006 4:46
16	15	Sports	2/15/2006 4:46
17	16	Travel	2/15/2006 4:46
18			

The background color represents the status of each cell, and there are four distinct colors that are used while editing table data:

Note

The Green and Blue colors were switched in MySQL for Excel 1.2.0.

Table 25.1 Background cell colors

Color	Description
White	Default color for all cells. This is either the original data, or the data after Refresh from DB is clicked.
Green	Cells that were committed with success.
Blue	Cells that were modified but have not yet been committed.
Red	Cells that generated an error when a commit was attempted. An error dialog is also displayed while the commit is attempted.
Orange	Cells that had a commit attempted, but the commit failed due to detected changes from external sources. For example, a different user made a change to a field after it was imported into Excel. This is a feature of Optimistic Updates.
Yellow	Cells that accept new data. Data entered here is inserted into the MySQL table.

In our example, the green "Drama" field was changed and then committed first, then the blue "Gaming" field was changed but not committed, and then **Auto-Commit** was enabled before changing the "9" to a "10" in column 10, which generated an error because this commit would have added a duplicate value as primary key.

25.3 Import MySQL Data into Excel

Data can be imported from MySQL into a Microsoft Excel spreadsheet by using the **Import MySQL Data** option after selecting either a table, view, or procedure to import.

Choosing columns to import

By default, all columns are selected and will be imported. Specific columns may be selected (or unselected) using the standard Microsoft Windows method of either **Control** + `Mouse click` to toggle the selection of individual columns, or **Shift** + `Mouse click` to select a range of columns.

The background color of a column shows the status of each column. The color white means that the column has been selected, and therefore it will be imported. Conversely, a gray background means that the column will not be imported.

Right-clicking anywhere in the preview grid opens a context-menu with either a `Select None` or `Select All` option, depending on the current status.

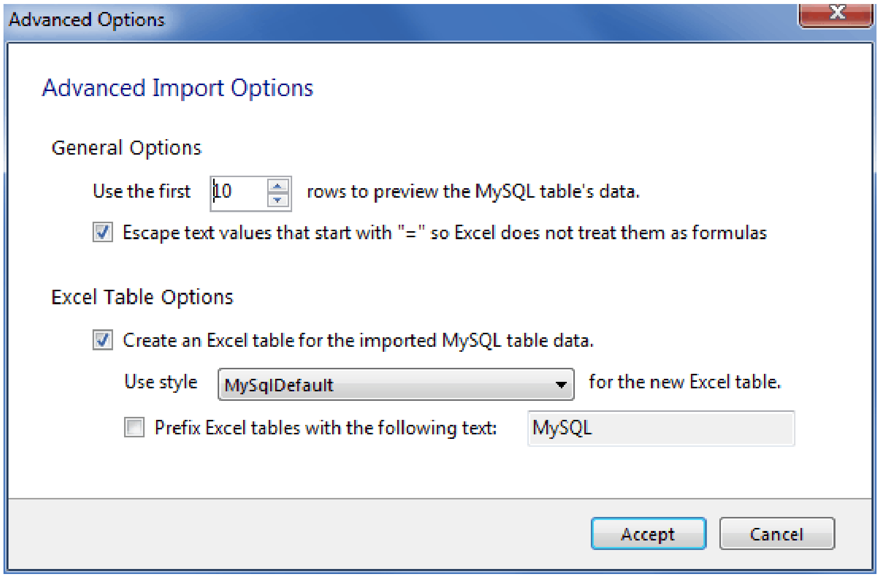
Importing a table

The dialog while importing a table includes the following options:

- **Include Column Names as Headers**: Enabled by default, this inserts the column names at the top of the Microsoft Excel spreadsheet as a "headers" row.
- **Limit to ___ Rows and Start with Row ___**: Disabled by default, this limits the range of imported data. The `Limit to` option defaults to `1`, and defines the number of rows to import. The `Start with Row` option defaults to `1` (the first row), and defines where the import begins. Each option has a maximum value of COUNT(rows) in the table.

The Advanced Options include:

Figure 25.4 Importing table data with MySQL for Excel: Advanced options



Advanced Options

Advanced Import Options

General Options

Use the first 10 rows to preview the MySQL table's data.

☑ Escape text values that start with "=" so Excel does not treat them as formulas

Excel Table Options

☑ Create an Excel table for the imported MySQL table data.

Use style MySqlDefault for the new Excel table.

□ Prefix Excel tables with the following text: MySQL

General Options:

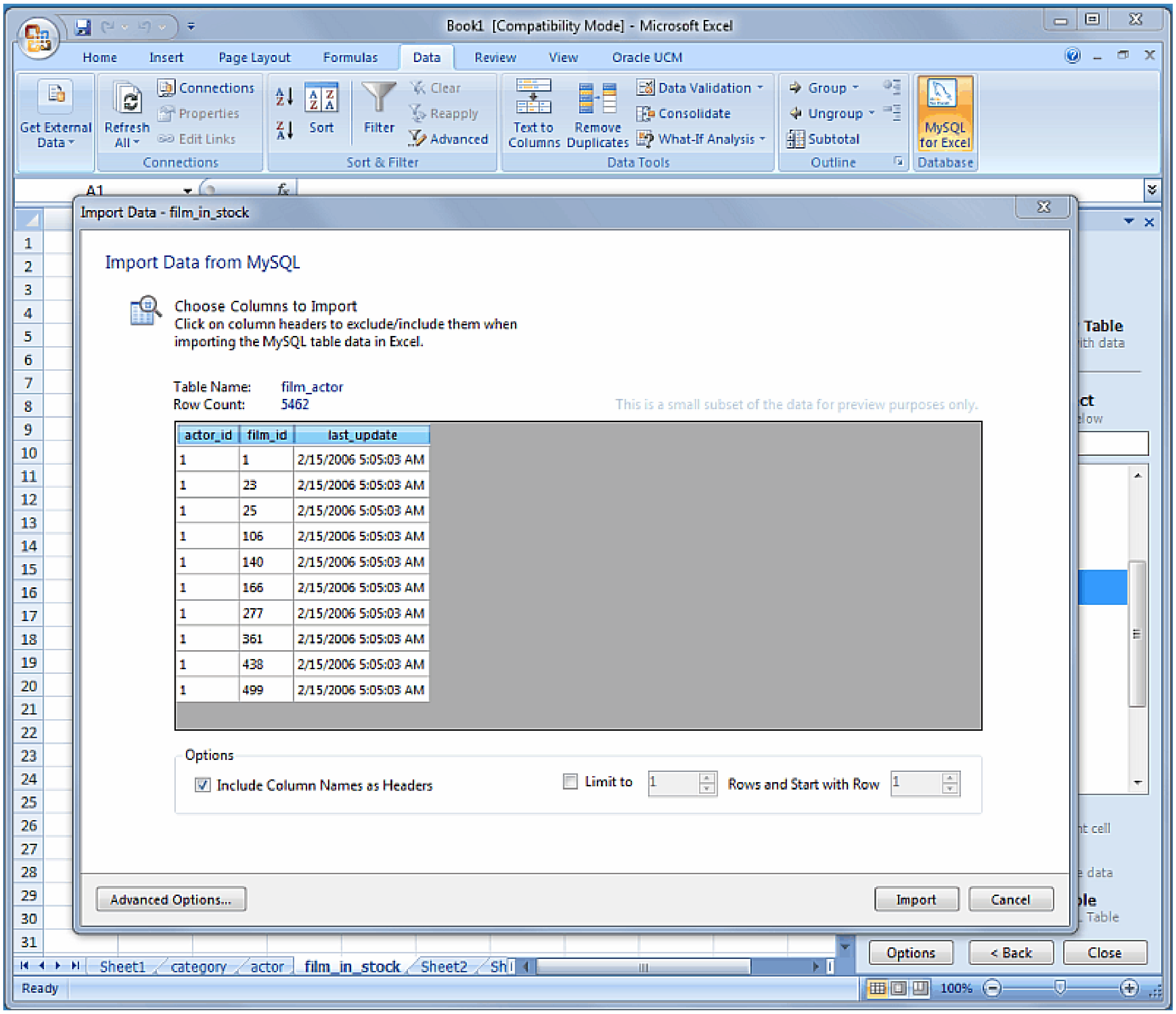
- Use the first [ ] rows to preview the MySQL tables data. Defaults to 10.
- [] Escape text values that start with "=" so Excel does not treat them as formulas. Enabled by default.

Excel Table Options:

- [] Create an Excel table for the imported MySQL table data. Enabled by default.
- Use style [ ] for the new Excel table. Defaults to `MySqlDefault`.
- [] Prefix Excel tables with the following text: _______. Disabled by default.

Importing a table displays a dialog similar to the following:

Figure 25.5 Importing table data with MySQL for Excel



Import Data - film_in_stock

Import Data from MySQL

Choose Columns to Import
Click on column headers to exclude/include them when importing the MySQL table data in Excel.

Table Name: film_actor
Row Count: 5462

This is a small subset of the data for preview purposes only.

actor_id	film_id	last_update
1	1	2/15/2006 5:05:03 AM
1	23	2/15/2006 5:05:03 AM
1	25	2/15/2006 5:05:03 AM
1	106	2/15/2006 5:05:03 AM
1	140	2/15/2006 5:05:03 AM
1	166	2/15/2006 5:05:03 AM
1	277	2/15/2006 5:05:03 AM
1	361	2/15/2006 5:05:03 AM
1	438	2/15/2006 5:05:03 AM
1	499	2/15/2006 5:05:03 AM

Options

☑ Include Column Names as Headers

□ Limit to 1 Rows and Start with Row 1

Advanced Options... Import Cancel

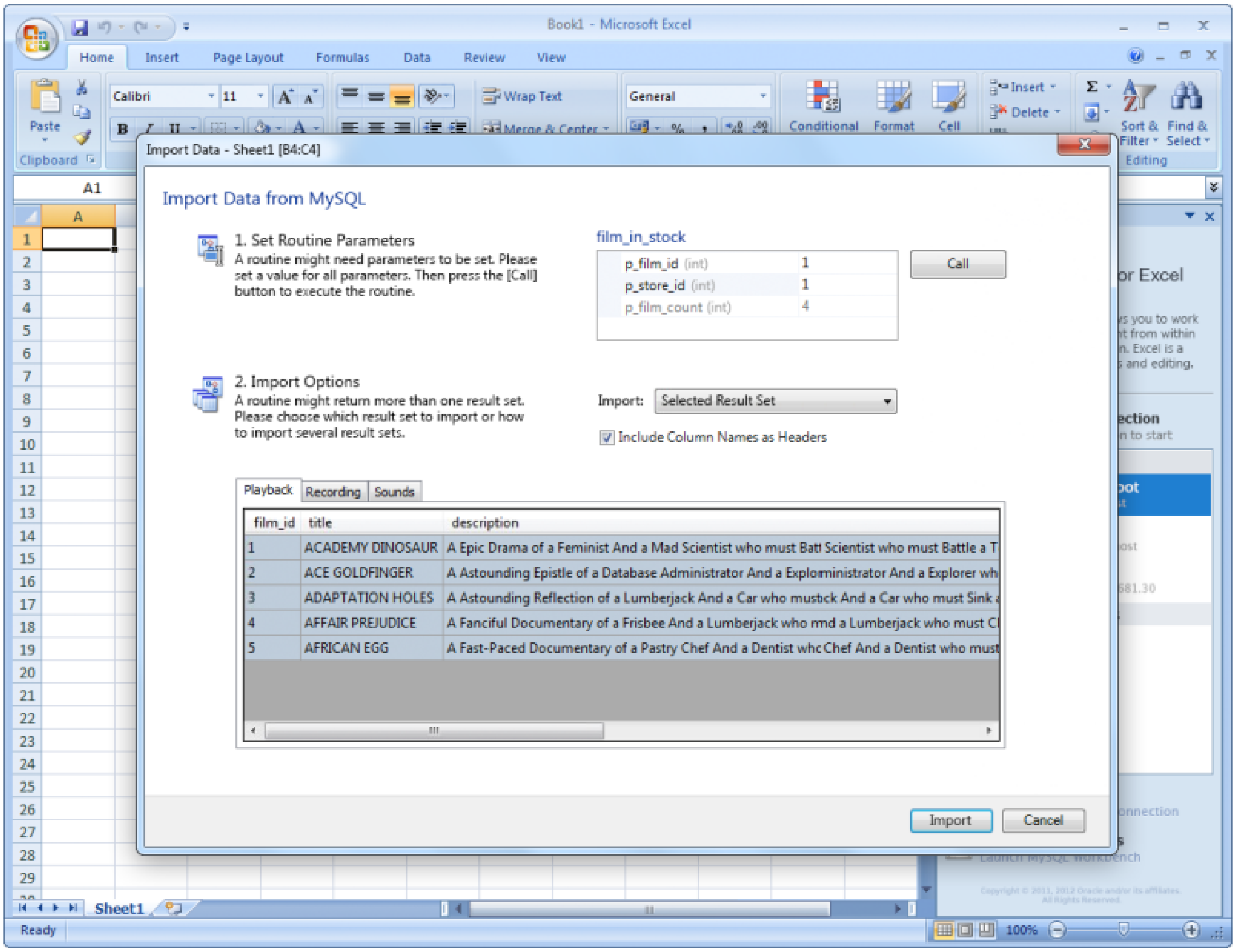
Importing a view or procedure

Importing a view or procedure displays a similar dialogue, but with the following options:

- **Include Column Names as Headers**: Enabled by default, this will insert the column names at the top of the Excel spreadsheet as a "headers" row.
- **Import**: Because a procedure might return multiple result sets, the import options include:
  - Selected Result Set: Imports the selected tab sheet. This is the default behavior.
  - All Result Sets - Arranged Horizontally: Imports all result sets into the Excel Worksheet horizontally, and inserts one empty column between each result set.
  - All Result Sets - Arranged Vertically: Imports all result sets into the Excel Worksheet vertically, and inserts one empty row between each result set.

For example, a dialogue like the following is displayed after importing a procedure and pressing the Call button to invoke the stored procedure:

Figure 25.6 Importing called stored procedure data with MySQL for Excel



Import Data - Sheet1 [B4:C4]

Import Data from MySQL

1. Set Routine Parameters

A routine might need parameters to be set. Please set a value for all parameters. Then press the [Call] button to execute the routine.

film_in_stock

p_film_id (int)	1
p_store_id (int)	1
p_film_count (int)	4

Call

2. Import Options

A routine might return more than one result set. Please choose which result set to import or how to import several result sets.

Import: Selected Result Set

☑ Include Column Names as Headers

Playback | Recording | Sounds

film_id	title	description
1	ACADEMY DINOSAUR	A Epic Drama of a Feminist And a Mad Scientist who must Batt Scientist who must Battle a T
2	ACE GOLDFINGER	A Astounding Epistle of a Database Administrator And a Explorninistrator And a Explorer wh
3	ADAPTATION HOLES	A Astounding Reflection of a Lumberjack And a Car who mustick And a Car who must Sink
4	AFFAIR PREJUDICE	A Fanciful Documentary of a Frisbee And a Lumberjack who rnd a Lumberjack who must Cl
5	AFRICAN EGG	A Fast-Paced Documentary of a Pastry Chef And a Dentist whc Chef And a Dentist who must

Import Cancel

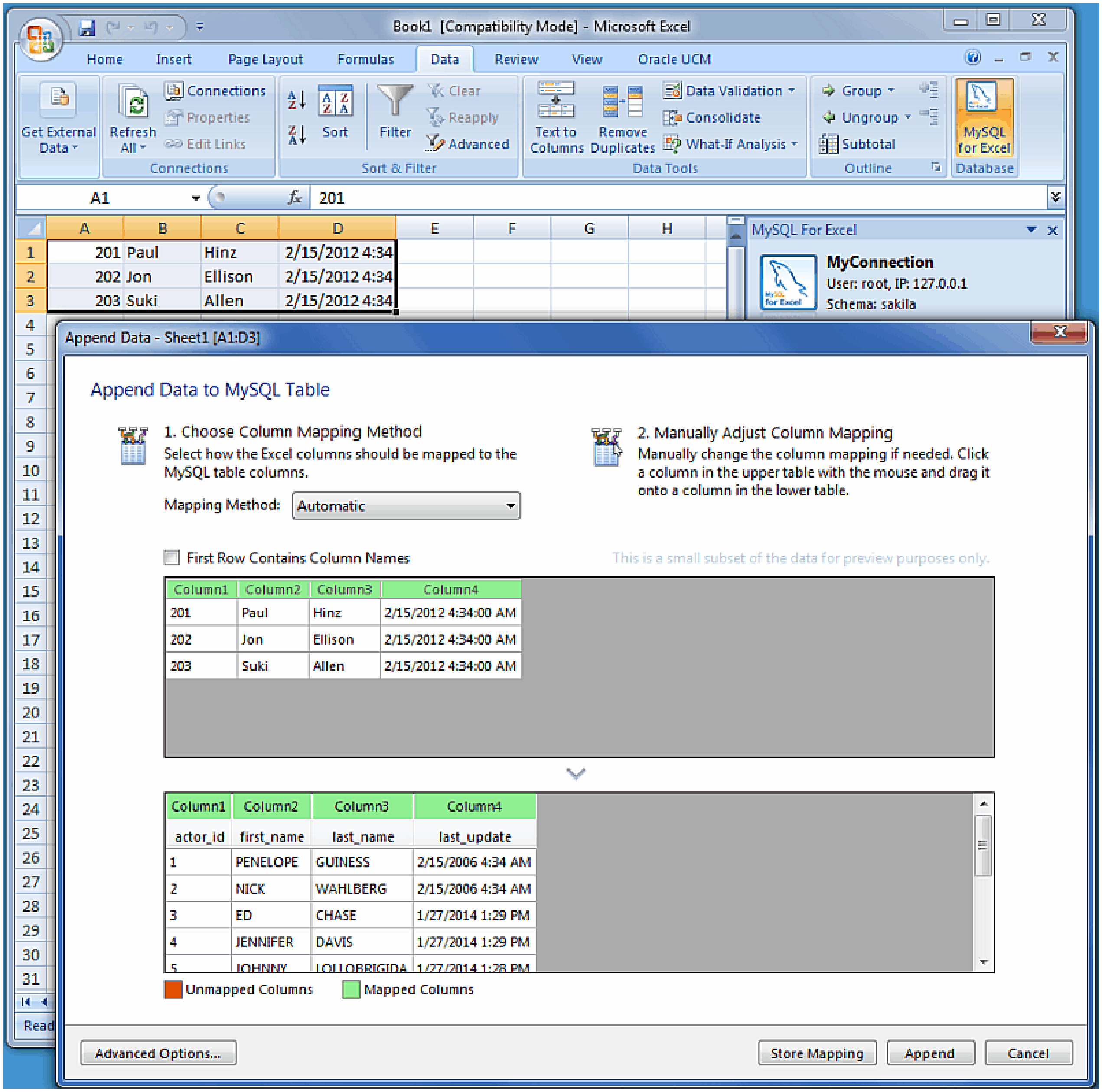
25.4 Append Excel Data into MySQL

Data from a Microsoft Excel spreadsheet can be appended to a MySQL database table by using the **Append Excel MySQL Data to Table** option.

Column mappings

Mapping the Excel columns to the MySQL columns can be executed automatically (default), manually, or by using a stored mapping routine. An automatic mapping routine is the default, and can be can be tweaked if every column cannot be matched automatically. The following screenshot shows two columns of Excel data, and the preview dialog after choosing **Append Excel Data to Table**:

Figure 25.7 Appending Excel data to MySQL (Automatic mapping)



	A	B	C	D
1	201	Paul	Hinz	2/15/2012 4:34
2	202	Jon	Ellison	2/15/2012 4:34
3	203	Suki	Allen	2/15/2012 4:34

Append Data - Sheet1 [A1:D3]

Append Data to MySQL Table

1. Choose Column Mapping Method

Select how the Excel columns should be mapped to the MySQL table columns.

Mapping Method: Automatic

2. Manually Adjust Column Mapping

Manually change the column mapping if needed. Click a column in the upper table with the mouse and drag it onto a column in the lower table.

□ First Row Contains Column Names

This is a small subset of the data for preview purposes only.

Column1	Column2	Column3	Column4
201	Paul	Hinz	2/15/2012 4:34:00 AM
202	Jon	Ellison	2/15/2012 4:34:00 AM
203	Suki	Allen	2/15/2012 4:34:00 AM

Column1	Column2	Column3	Column4
actor_id	first_name	last_name	last_update
1	PENELOPE	GUINESS	2/15/2006 4:34 AM
2	NICK	WAHLBERG	2/15/2006 4:34 AM
3	ED	CHASE	1/27/2014 1:29 PM
4	JENNIFER	DAVIS	1/27/2014 1:29 PM
5	JOHNNY	LOLLOBRIGIDA	1/27/2014 1:28 PM

🟥 Unmapped Columns 🟩 Mapped Columns

General mapping information

It is common to tweak the column mappings. A few notes about the manual mapping process:

- Manual mapping is performed by dragging a column from the upper source grid (Excel spreadsheet) and dropping it into the lower target column MySQL table grid. Click anywhere within the column to initiate this dragging routine.
- The color of the header field for each column defines the current mapping status of the column. The colors include:
  - Green: A source column is mapped to a target column.

  - Red: A target column is not mapped.
  - Gray: A source column is not mapped.
- A source column may be mapped to multiple target columns, although this action generates a warning dialog.
- Right-clicking a target column shows a context menu with options to either Remove Column Mapping for a single column, or to Clear All Mappings for all columns. Dragging a target column outside of the grid removes the mapping.

Mapping methods

The three mapping methods are described below:

- **Automatic**: The automatic mapping method attempts to match the Excel source column names with the MySQL target table column names. It is then possible to manually tweak the mapping afterwards.

  If the automatic process finds zero columns to match, then a simple 1 to 1 matching routine is attempted. Meaning, SourceColumn #1 to TargetColumn #1, SourceColumn #2 to TargetColumn #2, and so on.

- **Manual**: The source column names are manually dragged (matched) with the target column names. Manual dragging can also be performed after the **Automatic** method is selected.

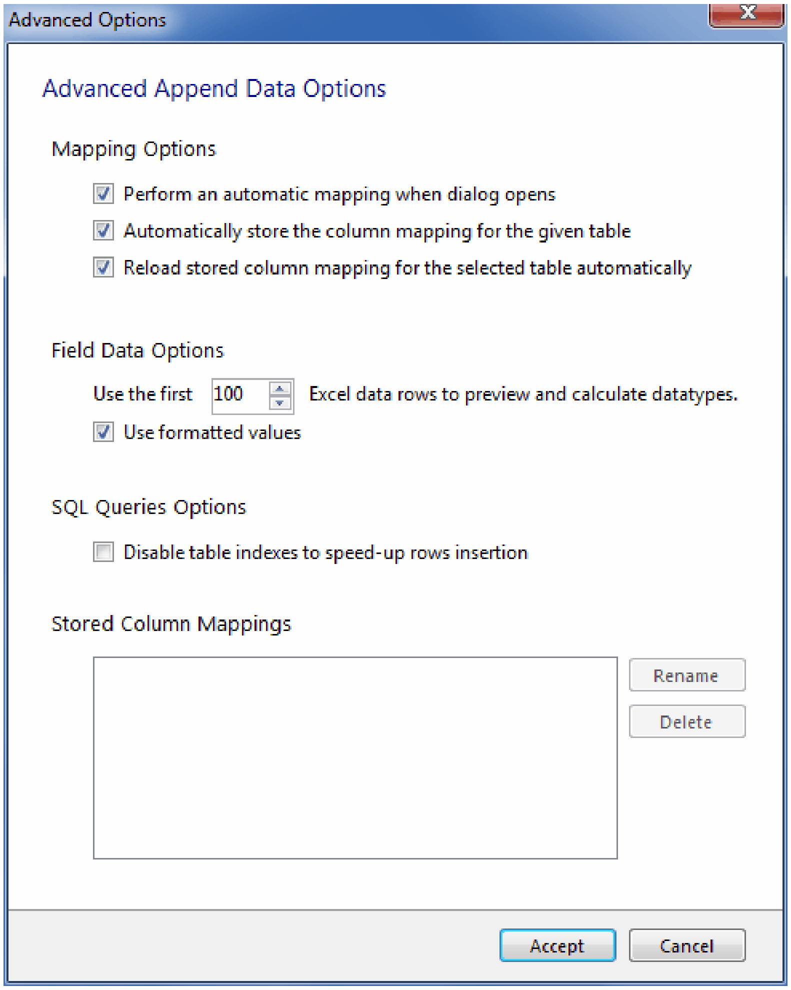
- **Stored**: Manual mapping styles may be saved using the Store Mapping button, which will also prompt for a name and then save it using a "*`name`* (dbname.tablename)" naming scheme. The saved mapping style will then be available alongside the **Automatic** and **Manual** options.

  Stored mappings may be deleted or renamed within the Advanced Options dialog.

Advanced Options

There are several advanced options that are configured and stored between sessions for each Excel user. The dialog looks similar to:

Figure 25.8 Appending Excel data to MySQL (Advanced Options)



Advanced Options

Advanced Append Data Options

Mapping Options

☑ Perform an automatic mapping when dialog opens

☑ Automatically store the column mapping for the given table

☑ Reload stored column mapping for the selected table automatically

Field Data Options

Use the first 100 Excel data rows to preview and calculate datatypes.

☑ Use formatted values

SQL Queries Options

☐ Disable table indexes to speed-up rows insertion

Stored Column Mappings

The advanced **Mapping Options**:

- `Perform an automatic mapping when dialog opens`: Automatically attempt to map the target and source when the **Append Data** dialog is opened. This feature is enabled by default.
- `Automatically store the column mapping for the given table`: Stores each mapping routine after executing the Append operation. The mapping routine is saved using the "tablenameMapping (dbname.tablename)" format. This may also be performed manually using the Store Mapping button. It is enabled by default, and this feature was added in MySQL for Excel 1.1.0.
- `Reload stored column mapping for the selected table automatically`: If a stored mapping routine exists that matches all column names in the source grid with the target grid, then it is automatically be loaded. This is enabled by default, and this feature was added in MySQL for Excel 1.1.0.

The advanced **Field Data Options**:

- **Use the first** *`100`* (default) Excel data rows to preview and calculate data types. This determines the number of rows that the preview displays, and the values that affect the automatic mapping feature.

- **Use formatted values**: The data from Excel is treated as `Text`, `Double`, or `Date`. This is enabled by default. When disabled, data is never treated as a `Date` type, so for example, this means that a date would be represented as a number.

The advanced **SQL Queries Options**:

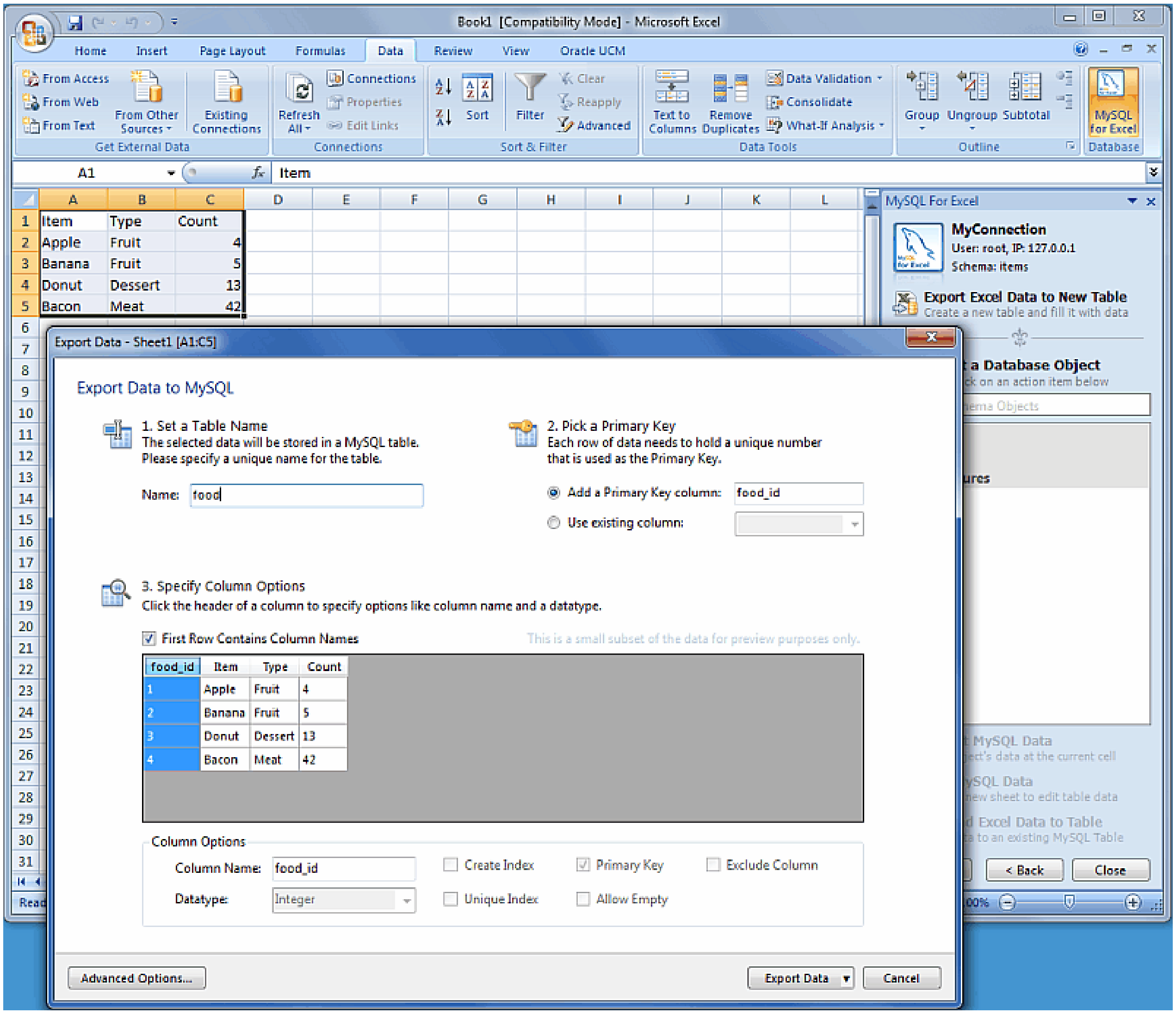
- `Disable table indexes to speed-up rows insertion`: This option is disabled by default, since you must make sure that if unique indexes are present, that the data mapped to that column does not contain duplicate data. This option was added in MySQL for Excel 1.2.1.

The **Stored Column Mappings** is a list of saved column mappings that were saved with the "Automatically store the column mapping for the given table" feature, or manually with the Store Mapping option.

25.5 Export Excel Data into MySQL

Data from a Microsoft Excel spreadsheet can be exported to a new MySQL database table by using the **Export Excel Data to New Table** option. Exporting data looks like so:

Figure 25.9 Exporting Excel data to MySQL



A1 *fx* Item

	A	B	C
1	Item	Type	Count
2	Apple	Fruit	4
3	Banana	Fruit	5
4	Donut	Dessert	13
5	Bacon	Meat	42

Export Data - Sheet1 [A1:C5]

Export Data to MySQL

1. Set a Table Name
The selected data will be stored in a MySQL table.
Please specify a unique name for the table.

Name: food

2. Pick a Primary Key
Each row of data needs to hold a unique number that is used as the Primary Key.

● Add a Primary Key column: food_id

○ Use existing column:

3. Specify Column Options
Click the header of a column to specify options like column name and a datatype.

☑ First Row Contains Column Names

This is a small subset of the data for preview purposes only.

food_id	Item	Type	Count
1	Apple	Fruit	4
2	Banana	Fruit	5
3	Donut	Dessert	13
4	Bacon	Meat	42

Column Options

Column Name: food_id

Datatype: Integer

☐ Create Index ☑ Primary Key ☐ Exclude Column

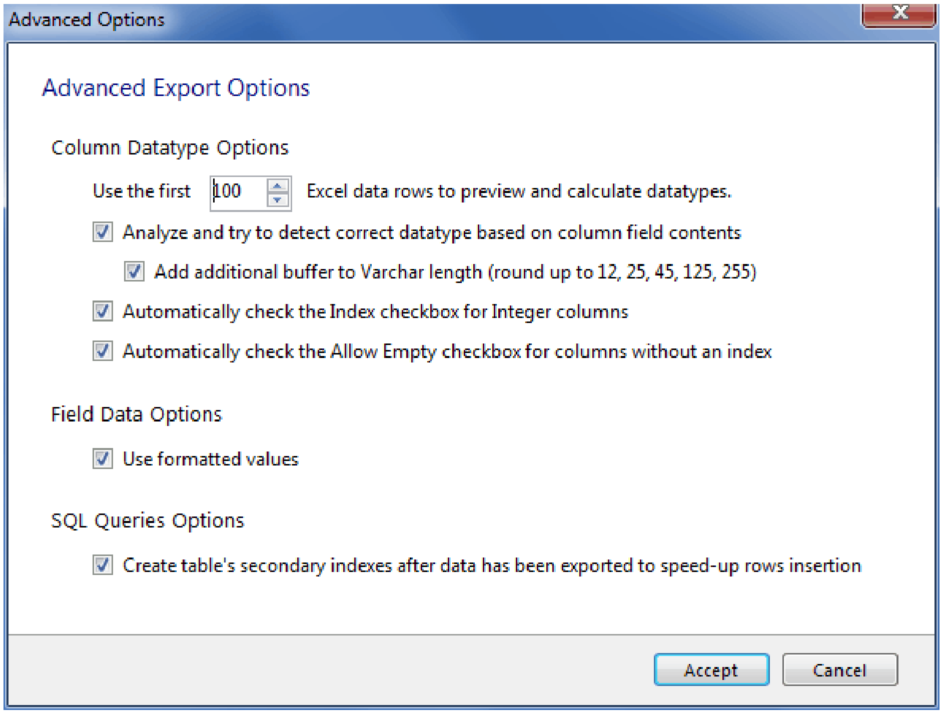
☐ Unique Index ☐ Allow Empty

Advanced Options... Export Data Cancel

Advanced Export options

Several advanced options enables you to tweak the exported data. The advanced options dialog looks like so:

Figure 25.10 Exporting Excel data to MySQL (Advanced options)



Advanced Export Options

Column Datatype Options

Use the first 100 Excel data rows to preview and calculate datatypes.

☑ Analyze and try to detect correct datatype based on column field contents

☑ Add additional buffer to Varchar length (round up to 12, 25, 45, 125, 255)

☑ Automatically check the Index checkbox for Integer columns

☑ Automatically check the Allow Empty checkbox for columns without an index

Field Data Options

☑ Use formatted values

SQL Queries Options

☑ Create table's secondary indexes after data has been exported to speed-up rows insertion

- **Column Datatype Options**:
  - Use the first `100` (default) Excel data rows to preview and calculate data types: This determines the number of rows that the preview displays, and the values that affect the automatic mapping feature.
  - Analyze and try to detect correct datatype based on column field contents: Attempts to analyze the data and determine the data type for the column. The column type is defined as `VARCHAR` if it contains multiple types.
  - Add additional buffer to `VARCHAR` length (round up to 12, 25, 45, 125, 255): When the data type is automatically detected and is set to `VARCHAR`, then it calculates the maximum length for all rows within the column, and rounds up the maximum length to one of the defined lengths above.

    If disabled, then the `VARCHAR` length is set to the length of the longest entry in the Excel spreadsheet.
  - Automatically check the Index checkbox for Integer columns: If enabled (default), columns with an Integer data type will have the **Create Index** option enabled by default.
  - Automatically check the Allow Empty checkbox for columns without an index: If enabled (default), columns without the **Create Index** checkbox checked will automatically enable the **Allow Empty** configuration option.
- **Field Data options**:

  - Use formatted values: When enabled (default), the data from Excel is treated as `Text`, `Double`, or `Date`. When disabled, data is never treated as a `Date` type, so for example this means that a date would be represented as a number.

- **Other options**:

  - Create table's secondary indexes after data has been exported to speed-up rows insertion: This saves disk I/O for bulk inserts (thousands of rows) since reindexing will not happen every time a row is inserted, but only once at the end of the data insertion. This option is enabled by default, and was added in MySQL for Excel 1.2.1.

  - Note: This option was `Removed` in MySQL for Excel 1.2.1. Now, the default behavior is to always remove empty columns from the calculations.

    Remove columns that contain no data, otherwise flag them as "Excluded": If enabled, columns without data in Excel are removed and not shown in the preview panel. If disabled (default), these columns will exist but have the **Exclude Column** option checked. This option was added in MySQL for Excel 1.1.0.

25.6 What Is New In MySQL for Excel

Version 1.2.0

- `Edit Connections`: MySQL connections can now be edited from within the MySQL for Excel plugin by right-clicking and choosing **Edit Connection**. Before, these connections could only be edited with MySQL Workbench.

- `Optimistic Updates`: Previously, only "Pessimistic Updates" were used, which means that pressing **Commit Changes** would overwrite changes performed outside of MySQL for Excel for the edited cells.

  Both options remain available today, and optimistic updates are enabled by default. This update type can be set either as a preference, or toggled per session.

- The **Append Data** dialog will now notify you of incompatible types (with visual warnings) when mapping source Excel columns to target MySQL columns.

  If a mismatch is discovered, then the column in the source grid that contains the mapped Excel data turns red, and selecting this column displays a warning with text explaining that the source data is not suitable for the mapped target column's data type.

- New preview preferences allow you to enable one of the following three options:

  - **Preview SQL statements before they are sent to the Server**: View (and optionally) edit the MySQL UPDATE/INSERT statements before they are committed.

  - **Show executed SQL statements along with the results**: View the statements after they are committed, which is the current behavior.

  - **Do not show the MySQL statements**: Only show summary information, such as number of affected rows, and not MySQL statements. This is enabled by default.

- Create Table: The **Data Export** feature now has the option to only create the table without inserting the data.

  To execute, toggle the Export Data button to Create Table, and then click.

- The selected schema name is now displayed on top of the MySQL for Excel Database Object Selection panel.
- The **Advanced Options** dialogs opened from the Import, Export and Append Data windows now immediately apply the option changes, when before the **Advanced Options** dialog had to be reopened before the changes could be previewed.
- **Edit Data** sessions can now be saved: Using the new **Edit Session** preferences, these sessions were automatically closed after closing an Excel workbook. This data, such as the Workbench connection ID, MySQL schema, and MySQL table name, can now be preserved if the Excel workbook is saved to disk, and available when the Excel workbook is reopened.
- Excel tables are automatically created for any data imported from MySQL to an Excel worksheet, with a name like "Schema.DB-Object-name". The DB object name can be a MySQL table, view, or stored procedure. Options for this feature are listed under **Import Data**, **Advanced Options**. The newly created Excel tables can be referenced for data analysis in Pivot Tables or reports.

25.7 MySQL for Excel FAQ

Frequently Asked Questions, with Answers.

Questions

- 26.7.1: [2843] I installed the MySQL for Excel plugin, but can't find it in Microsoft Excel. How do I start it?
- 26.7.2: [2843] I click on **Edit Data** and after importing the table data into Excel, I can't sort or move columns around. How can I do that?
- 26.7.3: [2843] When editing a MySQL table's data, the Excel worksheet where the data is dumped is protected. How can unprotect it?
- 26.7.4: [2844] The MySQL Workbench connections that use SSH tunneling appear grayed out (disabled) in MySQL for Excel. How can I use a SSH connection?

Questions and Answers

26.7.1: I installed the MySQL for Excel plugin, but can't find it in Microsoft Excel. How do I start it?

The MySQL for Excel plugin is automatically added to Microsoft Excel's data menu when it is installed. Look for the MySQL for Excel icon, by default it will be listed on the right side of the main menu.

If it's not there, then you might have to reinstall the plugin. But before doing so, first check if it's listed under "Add/Remove Programs" in Microsoft Windows. If not, then it has not been installed. Next, check the Excel Add-Ins list. For Office 2007 this is found by clicking the Office logo in Excel (top left corner), click Excel Options, then select **Add-Ins**. Is MySQL for Excel listed as a COM Add-in? If so, then consider filing a bug report (bugs.mysql.com), or attempt to reinstall the plugin.

26.7.2: I click on Edit Data and after importing the table data into Excel, I can't sort or move columns around. How can I do that?

In order to maintain the mapping of rows and columns in the Excel Worksheet against the rows and columns in the MySQL table, no alteration is permitted on the worksheet (i.e. sorting, deleting rows, deleting columns). If you need to alter the data there you can do that by right-clicking the **Edit Data** window and selecting **Exit Edit Mode**.

26.7.3: When editing a MySQL table's data, the Excel worksheet where the data is dumped is protected. How can unprotect it?

The Excel worksheet is protected to not allow alterations to the order of rows and columns. The password used for the protection is a GUID auto-generated at runtime so that the protection is not violated in any way. If you wish to unprotect the worksheet to manipulate your data, you can do that by right-clicking the **Edit Data** window and selecting **Exit Edit Mode**.

26.7.4: The MySQL Workbench connections that use SSH tunneling appear grayed out (disabled) in MySQL for Excel. How can I use a SSH connection?

This is a known limitation of MySQL for Excel. MySQL for Excel uses MySQL Connector/NET to connect and communicate with MySQL databases. Connector/NET does not have SSH support, so the behavior will change if Connector/NET supports it in the future.

Appendix A Licenses for Third-Party Components

Table of Contents

The following is a list of the libraries we have included with the MySQL Server source and components used to test MySQL. We are thankful to all individuals that have created these. Some of the components require that their licensing terms be included in the documentation of products that include them. Cross references to these licensing terms are given with the applicable items in the list.

- GroupLens Research Project

  The MySQL Quality Assurance team would like to acknowledge the use of the MovieLens Data Sets (10 million ratings and 100,000 tags for 10681 movies by 71567 users) to help test MySQL products and to thank the GroupLens Research Project at the University of Minnesota for making the data sets available.

MySQL 5.7

- Section A.1, “Artistic License (Perl) 1.0”
- Section A.2, “Boost Library License”
- Section A.3, “`dtoa.c` License”
- Section A.4, “Editline Library (`libedit`) License”
- Section A.5, “`Expect.pm` License”
- Section A.6, “Facebook Fast Checksum Patch License”
- Section A.7, “Facebook Patches License”
- Section A.8, “`FindGTest.cmake` License”
- Section A.9, “Fred Fish's Dbug Library License”
- Section A.10, “`getarg` License”
- Section A.12, “GNU General Public License Version 2.0, June 1991”
- Section A.13, “GNU General Public License Version 3.0, 29 June 2007 and GCC Runtime Library Exception Version 3.1, 31 March 2009”
- Section A.14, “GNU Lesser General Public License Version 2.1, February 1999”
- Section A.15, “GNU Readline License”
- Section A.16, “GNU Standard C++ Library (libstdc++) License”
- Section A.17, “Google Controlling Master Thread I/O Rate Patch License”
- Section A.18, “Google Perftools (TCMalloc utility) License”
- Section A.19, “Google SMP Patch License”
- Section A.20, “`lib_sql.cc` License”
- Section A.21, “Libaio License”
- Section A.22, “`libevent` License”
- Section A.25, “Linux-PAM License”
- Section A.29, “md5 (Message-Digest Algorithm 5) License”
- Section A.30, “MeCab License”
- Section A.31, “`memcached` License”
- Section A.32, “`Memcached.pm` License”

- Section A.33, “`mkpasswd.pl` License”
- Section A.34, “nt_servc (Windows NT Service class library) License”
- Section A.35, “OpenPAM License”
- Section A.36, “OpenSSL v1.0 License”
- Section A.38, “Percona Multiple I/O Threads Patch License”
- Section A.39, “Red HAT RPM Spec File License”
- Section A.40, “RegEX-Spencer Library License”
- Section A.41, “Richard A. O'Keefe String Library License”
- Section A.42, “SHA-1 in C License”
- Section A.43, “Unicode Data Files”
- Section A.44, “`zlib` License”

MySQL Proxy

- Section A.11, “GLib License (for MySQL Proxy)”
- Section A.14, “GNU Lesser General Public License Version 2.1, February 1999”
- Section A.22, “`libevent` License”
- Section A.23, “Libiconv License”
- Section A.24, “`libintl` License”
- Section A.26, “`LPeg` Library License”
- Section A.27, “Lua (liblua) License”
- Section A.28, “`LuaFileSystem` Library License”
- Section A.37, “PCRE License”

A.1 Artistic License (Perl) 1.0

```
The "Artistic License"

Preamble

The intent of this document is to state the conditions under which a
Package may be copied, such that the Copyright Holder maintains some
semblance of artistic control over the development of the package,
while giving the users of the package the right to use and distribute
the Package in a more-or-less customary fashion, plus the right to make
reasonable modifications.

Definitions:

        "Package" refers to the collection of files distributed by the
        Copyright Holder, and derivatives of that collection of files
        created through textual modification.

        "Standard Version" refers to such a Package if it has not been
        modified, or has been modified in accordance with the wishes
        of the Copyright Holder as specified below.

        "Copyright Holder" is whoever is named in the copyright or
        copyrights for the package.

        "You" is you, if you're thinking about copying or distributing
        this Package.

        "Reasonable copying fee" is whatever you can justify on the
        basis of media cost, duplication charges, time of people involved,
        and so on.  (You will not be required to justify it to the
        Copyright Holder, but only to the computing community at large
        as a market that must bear the fee.)

        "Freely Available" means that no fee is charged for the item
        itself, though there may be fees involved in handling the item.
        It also means that recipients of the item may redistribute it
        under the same conditions they received it.

1. You may make and give away verbatim copies of the source form of the
Standard Version of this Package without restriction, provided that you
duplicate all of the original copyright notices and associated disclaimers.

2. You may apply bug fixes, portability fixes and other modifications
derived from the Public Domain or from the Copyright Holder.  A Package
modified in such a way shall still be considered the Standard Version.

3. You may otherwise modify your copy of this Package in any way, provided
that you insert a prominent notice in each changed file stating how and
when you changed that file, and provided that you do at least ONE of the
following:

    a) place your modifications in the Public Domain or otherwise make them
    Freely Available, such as by posting said modifications to Usenet or
    an equivalent medium, or placing the modifications on a major archive
    site such as uunet.uu.net, or by allowing the Copyright Holder to include
    your modifications in the Standard Version of the Package.

    b) use the modified Package only within your corporation or organization.

    c) rename any non-standard executables so the names do not conflict
    with standard executables, which must also be provided, and provide
    a separate manual page for each non-standard executable that clearly
    documents how it differs from the Standard Version.

    d) make other distribution arrangements with the Copyright Holder.

4. You may distribute the programs of this Package in object code or
executable form, provided that you do at least ONE of the following:

    a) distribute a Standard Version of the executables and library files,
    together with instructions (in the manual page or equivalent) on where
    to get the Standard Version.

    b) accompany the distribution with the machine-readable source of
    the Package with your modifications.

    c) give non-standard executables non-standard names, and clearly
    document the differences in manual pages (or equivalent), together
    with instructions on where to get the Standard Version.

    d) make other distribution arrangements with the Copyright Holder.

5. You may charge a reasonable copying fee for any distribution of this
Package.  You may charge any fee you choose for support of this
Package.  You may not charge a fee for this Package itself.  However,
you may distribute this Package in aggregate with other (possibly
commercial) programs as part of a larger (possibly commercial) software
distribution provided that you do not advertise this Package as a
product of your own.  You may embed this Package's interpreter within
an executable of yours (by linking); this shall be construed as a mere
form of aggregation, provided that the complete Standard Version of the
interpreter is so embedded.

6. The scripts and library files supplied as input to or produced as
output from the programs of this Package do not automatically fall
under the copyright of this Package, but belong to whoever generated
them, and may be sold commercially, and may be aggregated with this
Package.  If such scripts or library files are aggregated with this
Package via the so-called "undump" or "unexec" methods of producing a
binary executable image, then distribution of such an image shall
neither be construed as a distribution of this Package nor shall it
fall under the restrictions of Paragraphs 3 and 4, provided that you do
not represent such an executable image as a Standard Version of this
Package.

7. C subroutines (or comparably compiled subroutines in other
languages) supplied by you and linked into this Package in order to
emulate subroutines and variables of the language defined by this
Package shall not be considered part of this Package, but are the
equivalent of input as in Paragraph 6, provided these subroutines do
not change the language in any way that would cause it to fail the
regression tests for the language.

8. Aggregation of this Package with a commercial distribution is always
permitted provided that the use of this Package is embedded; that is,
when no overt attempt is made to make this Package's interfaces visible
to the end user of the commercial distribution.  Such use shall not be
construed as a distribution of this Package.

9. The name of the Copyright Holder may not be used to endorse or promote
products derived from this software without specific prior written
permission.

10. THIS PACKAGE IS PROVIDED "AS IS" AND WITHOUT ANY EXPRESS OR
IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED
WARRANTIES OF MERCHANTIBILITY AND FITNESS FOR A PARTICULAR PURPOSE.

                                The End
```

A.2 Boost Library License

The following software may be included in this product:

Boost C++ Libraries

Use of any of this software is governed by the terms of the license below:

```
Boost Software License - Version 1.0 - August 17th, 2003

Permission is hereby granted, free of charge, to any person or
organization obtaining a copy of the software and accompanying
documentation covered by this license (the "Software") to use,
reproduce, display, distribute, execute, and transmit the Software,
and to prepare derivative works of the Software, and to permit
third-parties to whom the Software is furnished to do so, all
subject to the following:

The copyright notices in the Software and this entire statement,
including the above license grant, this restriction and the
following disclaimer, must be included in all copies of the
Software, in whole or in part, and all derivative works of the
```

```
Software, unless such copies or derivative works are solely in the
form of machine-executable object code generated by a source
language processor.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND
NON-INFRINGEMENT. IN NO EVENT SHALL THE COPYRIGHT HOLDERS OR ANYONE
DISTRIBUTING THE SOFTWARE BE LIABLE FOR ANY DAMAGES OR OTHER
LIABILITY, WHETHER IN CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT
OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS
IN THE SOFTWARE.
```

A.3 dtoa.c License

The following software may be included in this product:

dtoa.c

```

The author of this software is David M. Gay.

Copyright (c) 1991, 2000, 2001 by Lucent Technologies.

Permission to use, copy, modify, and distribute this software for
any purpose without fee is hereby granted, provided that this entire
notice is included in all copies of any software which is or includes
a copy or modification of this software and in all copies of the
supporting documentation for such software.

THIS SOFTWARE IS BEING PROVIDED "AS IS", WITHOUT ANY EXPRESS OR
IMPLIED WARRANTY. IN PARTICULAR, NEITHER THE AUTHOR NOR LUCENT
MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND CONCERNING THE
MERCHANTABILITY OF THIS SOFTWARE OR ITS FITNESS FOR ANY PARTICULAR
PURPOSE.
```

A.4 Editline Library (libedit) License

The following software may be included in this product:

Editline Library (libedit)

Some files are:

```

Copyright (c) 1992, 1993
The Regents of the University of California. All rights reserved.

This code is derived from software contributed to
Berkeley by Christos Zoulas of Cornell University.

Redistribution and use in source and binary forms,
with or without modification, are permitted provided
that the following conditions are met:

1. Redistributions of source code must retain the
  above copyright notice, this list of conditions
  and the following disclaimer.
2. Redistributions in binary form must reproduce the
  above copyright notice, this list of conditions and
  the following disclaimer in the documentation and/or
  other materials provided with the distribution.
3. Neither the name of the University nor the names of
  its contributors may be used to endorse or promote
  products derived from this software without specific
  prior written permission.

THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY
AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO
EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE FOR ANY
DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED
AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED
OF THE POSSIBILITY OF SUCH DAMAGE.
```

Some files are:

```

Copyright (c) 2001 The NetBSD Foundation, Inc.
All rights reserved.

This code is derived from software contributed to The NetBSD Foundation

by Anthony Mallet.

Redistribution and use in source and binary forms,
with or without modification, are permitted provided
that the following conditions are met:

1. Redistributions of source code must retain the
   above copyright notice, this list of conditions
   and the following disclaimer.
2. Redistributions in binary form must reproduce the
    above copyright notice, this list of conditions and the
    following disclaimer in the documentation and/or
    other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE NETBSD FOUNDATION, INC.
AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED
WARRANTIES, INCLUDING, BUT NOT LIMITED  TO, THE IMPLIED
WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL
THE FOUNDATION OR CONTRIBUTORS BE LIABLE FOR ANY
DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
DAMAGE.
```

Some files are:

```

Copyright (c) 1997 The NetBSD Foundation, Inc.
All rights reserved.

This code is derived from software contributed to The NetBSD Foundation

by Jaromir Dolecek.

Redistribution and use in source and binary forms,
with or without modification, are permitted provided
that the following conditions are met:
```

```
1. Redistributions of source code must retain the
   above copyright notice, this list of conditions
   and the following disclaimer.
2. Redistributions in binary form must reproduce
    the above copyright notice, this list of conditions
    and the following disclaimer in the documentation
    and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE NETBSD FOUNDATION, INC.
AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED
WARRANTIES, INCLUDING, BUT NOT LIMITED  TO, THE IMPLIED
WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL
THE FOUNDATION OR CONTRIBUTORS BE LIABLE FOR ANY
DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
DAMAGE.
```

Some files are:

```
Copyright (c) 1998 Todd C. Miller <Todd.Miller@courtesan.com>

Permission to use, copy, modify, and distribute this
software for any purpose with or without fee is hereby
granted, provided that the above copyright notice and
this permission notice appear in all copies.

THE SOFTWARE IS PROVIDED "AS IS" AND TODD C. MILLER
DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE
INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY
AND FITNESS. IN NO EVENT SHALL TODD C. MILLER BE LIABLE
FOR ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL
DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM
LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION
OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION,
ARISING OUT OF OR IN CONNECTION WITH THE USE OR
PERFORMANCE OF THIS SOFTWARE.
```

Some files are:

```
Copyright (c) 1998 The NetBSD Foundation, Inc.
All rights reserved.

This code is derived from software contributed to The NetBSD
Foundation by Christos Zoulas.

Redistribution and use in source and binary forms, with or
without modification, are permitted provided that the following
conditions
are met:
1. Redistributions of source code must retain the above copyright
   notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above
   copyright notice, this list of conditions and the following
   disclaimer in the documentation and/or other materials provided
   with the distribution.

THIS SOFTWARE IS PROVIDED BY THE NETBSD FOUNDATION, INC. AND
```

```
CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES,
INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
DISCLAIMED. IN NO EVENT SHALL THE FOUNDATION OR CONTRIBUTORS
BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY,
OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT
OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS;
OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

Some files are:

```

Copyright (c) 2009 The NetBSD Foundation, Inc.
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:
1. Redistributions of source code must retain the above copyright
   notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright
   notice, this list of conditions and the following disclaimer in the
   documentation and/or other materials provided with the
   distribution.
3. All advertising materials mentioning features or use of this
   software must display the following acknowledgement:
       This product includes software developed by the NetBSD
       Foundation, Inc. and its contributors.
4. Neither the name of The NetBSD Foundation nor the names of its
   contributors may be used to endorse or promote products derived
   from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE NETBSD FOUNDATION, INC. AND
CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES,
INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
DISCLAIMED. IN NO EVENT SHALL THE FOUNDATION OR CONTRIBUTORS
BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY,
OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT
OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS;
OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF
THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

A.5 Expect.pm License

The following software may be included in this product:

Expect.pm Perl module

```

Expect.pm is licensed under the Perl license, which is essentially a dual
license.

Oracle may use, redistribute and/or modify this code under the terms of
either:

 a) the GNU General Public License as published by the Free Software
Foundation; either version 1, or (at your option) any later version, or

 b) the "Artistic License" which comes with the Expect/pr code.
```

```
Oracle elects to use the GPLv2 for version of MySQL that are licensed under
the GPL.

Oracle elects to use the Artistic license for all other (commercial) versions
of MySQL.

A copy of the GPLv2 and the Artistic License (Perl) 1.0 must be included with
any distribution:

The GNU General Public License (GPL-2.0)
Version 2, June 1991

Copyright (C) 1989, 1991 Free Software Foundation, Inc.
59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.

Preamble

The licenses for most software are designed to take away your freedom to
share and change it. By contrast, the GNU General Public License is intended
to guarantee your freedom to share and change free software--to make sure the
software is free for all its users. This General Public License applies to
most of the Free Software Foundation's software and to any other program
whose authors commit to using it. (Some other Free Software Foundation
software is covered by the GNU Library General Public License instead.) You
can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our
General Public Licenses are designed to make sure that you have the freedom
to distribute copies of free software (and charge for this service if you
wish), that you receive source code or can get it if you want it, that you
can change the software or use pieces of it in new free programs; and that
you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to
deny you these rights or to ask you to surrender the rights. These
restrictions translate to certain responsibilities for you if you distribute
copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or
for a fee, you must give the recipients all the rights that you have. You
must make sure that they, too, receive or can get the source code. And you
must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2)
offer you this license which gives you legal permission to copy, distribute
and/or modify the software.

Also, for each author's protection and ours, we want to make certain that
everyone understands that there is no warranty for this free software. If the
software is modified by someone else and passed on, we want its recipients to
know that what they have is not the original, so that any problems introduced
by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We
wish to avoid the danger that redistributors of a free program will
individually obtain patent licenses, in effect making the program
proprietary. To prevent this, we have made it clear that any patent must be
licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification
follow.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION
```

```
0. This License applies to any program or other work which contains a notice
placed by the copyright holder saying it may be distributed under the terms
of this General Public License. The "Program", below, refers to any such
program or work, and a "work based on the Program" means either the Program
or any derivative work under copyright law: that is to say, a work containing
the Program or a portion of it, either verbatim or with modifications and/or
translated into another language. (Hereinafter, translation is included
without limitation in the term "modification".) Each licensee is addressed as
"you".

Activities other than copying, distribution and modification are not covered
by this License; they are outside its scope. The act of running the Program
is not restricted, and the output from the Program is covered only if its
contents constitute a work based on the Program (independent of having been
made by running the Program). Whether that is true depends on what the
Program does.

1. You may copy and distribute verbatim copies of the Program's source code
as you receive it, in any medium, provided that you conspicuously and
appropriately publish on each copy an appropriate copyright notice and
disclaimer of warranty; keep intact all the notices that refer to this
License and to the absence of any warranty; and give any other recipients of
the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may
at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it,
thus forming a work based on the Program, and copy and distribute such
modifications or work under the terms of Section 1 above, provided that you
also meet all of these conditions:

    a) You must cause the modified files to carry prominent notices stating
that you changed the files and the date of any change.

    b) You must cause any work that you distribute or publish, that in whole
or in part contains or is derived from the Program or any part thereof, to be
licensed as a whole at no charge to all third parties under the terms of this
License.

    c) If the modified program normally reads commands interactively when
run, you must cause it, when started running for such interactive use in the
most ordinary way, to print or display an announcement including an
appropriate copyright notice and a notice that there is no warranty (or else,
saying that you provide a warranty) and that users may redistribute the
program under these conditions, and telling the user how to view a copy of
this License. (Exception: if the Program itself is interactive but does not
normally print such an announcement, your work based on the Program is not
required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable
sections of that work are not derived from the Program, and can be reasonably
considered independent and separate works in themselves, then this License,
and its terms, do not apply to those sections when you distribute them as
separate works. But when you distribute the same sections as part of a whole
which is a work based on the Program, the distribution of the whole must be
on the terms of this License, whose permissions for other licensees extend to
the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your
rights to work written entirely by you; rather, the intent is to exercise the
right to control the distribution of derivative or collective works based on
the Program.

In addition, mere aggregation of another work not based on the Program with
the Program (or with a work based on the Program) on a volume of a storage or
distribution medium does not bring the other work under the scope of this
```

```
License.

3. You may copy and distribute the Program (or a work based on it, under
Section 2) in object code or executable form under the terms of Sections 1
and 2 above provided that you also do one of the following:

    a) Accompany it with the complete corresponding machine-readable source
code, which must be distributed under the terms of Sections 1 and 2 above on
a medium customarily used for software interchange; or,

    b) Accompany it with a written offer, valid for at least three years, to
give any third party, for a charge no more than your cost of physically
performing source distribution, a complete machine-readable copy of the
corresponding source code, to be distributed under the terms of Sections 1
and 2 above on a medium customarily used for software interchange; or,

    c) Accompany it with the information you received as to the offer to
distribute corresponding source code. (This alternative is allowed only for
noncommercial distribution and only if you received the program in object
code or executable form with such an offer, in accord with Subsection b
above.)

The source code for a work means the preferred form of the work for making
modifications to it. For an executable work, complete source code means all
the source code for all modules it contains, plus any associated interface
definition files, plus the scripts used to control compilation and
installation of the executable. However, as a special exception, the source
code distributed need not include anything that is normally distributed (in
either source or binary form) with the major components (compiler, kernel,
and so on) of the operating system on which the executable runs, unless that
component itself accompanies the executable.

If distribution of executable or object code is made by offering access to
copy from a designated place, then offering equivalent access to copy the
source code from the same place counts as distribution of the source code,
even though third parties are not compelled to copy the source along with the
object code.

4. You may not copy, modify, sublicense, or distribute the Program except as
expressly provided under this License. Any attempt otherwise to copy, modify,
sublicense or distribute the Program is void, and will automatically
terminate your rights under this License. However, parties who have received
copies, or rights, from you under this License will not have their licenses
terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it.
However, nothing else grants you permission to modify or distribute the
Program or its derivative works. These actions are prohibited by law if you
do not accept this License. Therefore, by modifying or distributing the
Program (or any work based on the Program), you indicate your acceptance of
this License to do so, and all its terms and conditions for copying,
distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program),
the recipient automatically receives a license from the original licensor to
copy, distribute or modify the Program subject to these terms and conditions.
You may not impose any further restrictions on the recipients' exercise of
the rights granted herein. You are not responsible for enforcing compliance
by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent
infringement or for any other reason (not limited to patent issues),
conditions are imposed on you (whether by court order, agreement or
otherwise) that contradict the conditions of this License, they do not excuse
you from the conditions of this License. If you cannot distribute so as to
satisfy simultaneously your obligations under this License and any other
pertinent obligations, then as a consequence you may not distribute the
```

```
Program at all. For example, if a patent license would not permit
royalty-free redistribution of the Program by all those who receive copies
directly or indirectly through you, then the only way you could satisfy both
it and this License would be to refrain entirely from distribution of the
Program.

If any portion of this section is held invalid or unenforceable under any
particular circumstance, the balance of the section is intended to apply and
the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents
or other property right claims or to contest validity of any such claims;
this section has the sole purpose of protecting the integrity of the free
software distribution system, which is implemented by public license
practices. Many people have made generous contributions to the wide range of
software distributed through that system in reliance on consistent
application of that system; it is up to the author/donor to decide if he or
she is willing to distribute software through any other system and a licensee
cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a
consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain
countries either by patents or by copyrighted interfaces, the original
copyright holder who places the Program under this License may add an
explicit geographical distribution limitation excluding those countries, so
that distribution is permitted only in or among countries not thus excluded.
In such case, this License incorporates the limitation as if written in the
body of this License.

9. The Free Software Foundation may publish revised and/or new versions of
the General Public License from time to time. Such new versions will be
similar in spirit to the present version, but may differ in detail to address
new problems or concerns.

Each version is given a distinguishing version number. If the Program
specifies a version number of this License which applies to it and "any later
version", you have the option of following the terms and conditions either of
that version or of any later version published by the Free Software
Foundation. If the Program does not specify a version number of this License,
you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs
whose distribution conditions are different, write to the author to ask for
permission. For software which is copyrighted by the Free Software
Foundation, write to the Free Software Foundation; we sometimes make
exceptions for this. Our decision will be guided by the two goals of
preserving the free status of all derivatives of our free software and of
promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR
THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE
STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE
PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED,
INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND
PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE,
YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING
WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR
REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES,
INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING
OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO
LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR
THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER
PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE
POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible
use to the public, the best way to achieve this is to make it free software
which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach
them to the start of each source file to most effectively convey the
exclusion of warranty; and each file should have at least the "copyright"
line and a pointer to where the full notice is found.

    One line to give the program's name and a brief idea of what it does.
    Copyright (C) <year> <name of author>

    This program is free software; you can redistribute it and/or modify it
under the terms of the GNU General Public License as published by the Free
Software Foundation; either version 2 of the License, or (at your option) any
later version.

    This program is distributed in the hope that it will be useful, but
WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more
details.

    You should have received a copy of the GNU General Public License along
with this program; if not, write to the Free Software Foundation, Inc., 59
Temple Place, Suite 330, Boston, MA 02111-1307 USA

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when
it starts in an interactive mode:

    Gnomovision version 69, Copyright (C) year name of author Gnomovision
comes with ABSOLUTELY NO WARRANTY; for details type `show w'. This is free
software, and you are welcome to redistribute it under certain conditions;
type `show c' for details.

The hypothetical commands `show w' and `show c' should show the appropriate
parts of the General Public License. Of course, the commands you use may be
called something other than `show w' and `show c'; they could even be
mouse-clicks or menu items--whatever suits your program.

You should also get your employer (if you work as a programmer) or your
school, if any, to sign a "copyright disclaimer" for the program, if
necessary. Here is a sample; alter the names:

    Yoyodyne, Inc., hereby disclaims all copyright interest in the program
`Gnomovision' (which makes passes at compilers) written by James Hacker.

    signature of Ty Coon, 1 April 1989
    Ty Coon, President of Vice

This General Public License does not permit incorporating your program into
proprietary programs. If your program is a subroutine library, you may
consider it more useful to permit linking proprietary applications with the
library. If this is what you want to do, use the GNU Library General Public
License instead of this License.
```

```
The "Artistic License"

Preamble

The intent of this document is to state the conditions under which a
Package may be copied, such that the Copyright Holder maintains some
semblance of artistic control over the development of the package,
while giving the users of the package the right to use and distribute
the Package in a more-or-less customary fashion, plus the right to make
reasonable modifications.

Definitions:

        "Package" refers to the collection of files distributed by the
        Copyright Holder, and derivatives of that collection of files
        created through textual modification.

        "Standard Version" refers to such a Package if it has not been
        modified, or has been modified in accordance with the wishes
        of the Copyright Holder as specified below.

        "Copyright Holder" is whoever is named in the copyright or
        copyrights for the package.

        "You" is you, if you're thinking about copying or distributing
        this Package.

        "Reasonable copying fee" is whatever you can justify on the
        basis of media cost, duplication charges, time of people involved,
        and so on.  (You will not be required to justify it to the
        Copyright Holder, but only to the computing community at large
        as a market that must bear the fee.)

        "Freely Available" means that no fee is charged for the item
        itself, though there may be fees involved in handling the item.
        It also means that recipients of the item may redistribute it
        under the same conditions they received it.

1. You may make and give away verbatim copies of the source form of the
Standard Version of this Package without restriction, provided that you
duplicate all of the original copyright notices and associated disclaimers.

2. You may apply bug fixes, portability fixes and other modifications
derived from the Public Domain or from the Copyright Holder.  A Package
modified in such a way shall still be considered the Standard Version.

3. You may otherwise modify your copy of this Package in any way, provided
that you insert a prominent notice in each changed file stating how and
when you changed that file, and provided that you do at least ONE of the
following:

    a) place your modifications in the Public Domain or otherwise make them
    Freely Available, such as by posting said modifications to Usenet or
    an equivalent medium, or placing the modifications on a major archive
    site such as uunet.uu.net, or by allowing the Copyright Holder to include
    your modifications in the Standard Version of the Package.

    b) use the modified Package only within your corporation or organization.

    c) rename any non-standard executables so the names do not conflict
    with standard executables, which must also be provided, and provide
    a separate manual page for each non-standard executable that clearly
    documents how it differs from the Standard Version.

    d) make other distribution arrangements with the Copyright Holder.
```

```
4. You may distribute the programs of this Package in object code or
executable form, provided that you do at least ONE of the following:

    a) distribute a Standard Version of the executables and library files,
    together with instructions (in the manual page or equivalent) on where
    to get the Standard Version.

    b) accompany the distribution with the machine-readable source of
    the Package with your modifications.

    c) give non-standard executables non-standard names, and clearly
    document the differences in manual pages (or equivalent), together
    with instructions on where to get the Standard Version.

    d) make other distribution arrangements with the Copyright Holder.

5. You may charge a reasonable copying fee for any distribution of this
Package.  You may charge any fee you choose for support of this
Package.  You may not charge a fee for this Package itself.  However,
you may distribute this Package in aggregate with other (possibly
commercial) programs as part of a larger (possibly commercial) software
distribution provided that you do not advertise this Package as a
product of your own.  You may embed this Package's interpreter within
an executable of yours (by linking); this shall be construed as a mere
form of aggregation, provided that the complete Standard Version of the
interpreter is so embedded.

6. The scripts and library files supplied as input to or produced as
output from the programs of this Package do not automatically fall
under the copyright of this Package, but belong to whoever generated
them, and may be sold commercially, and may be aggregated with this
Package.  If such scripts or library files are aggregated with this
Package via the so-called "undump" or "unexec" methods of producing a
binary executable image, then distribution of such an image shall
neither be construed as a distribution of this Package nor shall it
fall under the restrictions of Paragraphs 3 and 4, provided that you do
not represent such an executable image as a Standard Version of this
Package.

7. C subroutines (or comparably compiled subroutines in other
languages) supplied by you and linked into this Package in order to
emulate subroutines and variables of the language defined by this
Package shall not be considered part of this Package, but are the
equivalent of input as in Paragraph 6, provided these subroutines do
not change the language in any way that would cause it to fail the
regression tests for the language.

8. Aggregation of this Package with a commercial distribution is always
permitted provided that the use of this Package is embedded; that is,
when no overt attempt is made to make this Package's interfaces visible
to the end user of the commercial distribution.  Such use shall not be
construed as a distribution of this Package.

9. The name of the Copyright Holder may not be used to endorse or promote
products derived from this software without specific prior written
permission.

10. THIS PACKAGE IS PROVIDED "AS IS" AND WITHOUT ANY EXPRESS OR
IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED
WARRANTIES OF MERCHANTIBILITY AND FITNESS FOR A PARTICULAR PURPOSE.

                                The End
```

A.6 Facebook Fast Checksum Patch License

The following software may be included in this product:

```
Facebook Fast Checksum Patch

Copyright (C) 2009-2010 Facebook, Inc.  All Rights Reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice,
   this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice,
   this list of conditions and the following disclaimer in the documentation
   and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY FACEBOOK, INC. “AS IS” AND ANY EXPRESS OR
IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED.  IN NO
EVENT SHALL FACEBOOK, INC. BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS;
OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR
OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Also included:

crc32.c -- compute the CRC-32 of a buf stream
Copyright (C) 1995-2005 Mark Adler

This software is provided 'as-is', without any express or implied
warranty.  In no event will the authors be held liable for any damages
arising from the use of this software.

Permission is granted to anyone to use this software for any purpose,
including commercial applications, and to alter it and redistribute it
freely, subject to the following restrictions:

1. The origin of this software must not be misrepresented; you must not
   claim that you wrote the original software. If you use this software
   in a product, an acknowledgment in the product documentation would be
   appreciated but is not required.
2. Altered source versions must be plainly marked as such, and must not be
   misrepresented as being the original software.
3. This notice may not be removed or altered from any source distribution.

Jean-loup Gailly jloup@gzip.org
Mark Adler madler@alumni.caltech.edu
```

A.7 Facebook Patches License

The following software may be included in this product:

```
Copyright (c) 2012, Facebook, Inc.
All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:
* Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
* Redistributions in binary form must reproduce the above copyright notice,
this list of conditions and the following disclaimer in the documentation
and/or other materials provided with the distribution.
THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
```

```
ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE
LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
POSSIBILITY OF SUCH DAMAGE.
```

A.8 FindGTest.cmake License

The following software may be included in this product:

```
FindGTest.cmake helper script (part of CMake)

Copyright 2009 Kitware, Inc.
Copyright 2009 Philip Lowman
Copyright 2009 Daniel Blezek

Distributed under the OSI-approved BSD License (the "License");
see accompanying file Copyright.txt for details.

This software is distributed WITHOUT ANY WARRANTY; without even the
implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
See the License for more information.
==========================================================================
(To distributed this file outside of CMake, substitute the full
 License text for the above reference.)

Thanks to Daniel Blezek  for the GTEST_ADD_TESTS code

Text of Copyright.txt mentioned above:

CMake - Cross Platform Makefile Generator
Copyright 2000-2009 Kitware, Inc., Insight Software Consortium
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:

* Redistributions of source code must retain the above copyright
  notice, this list of conditions and the following disclaimer.

* Redistributions in binary form must reproduce the above copyright
  notice, this list of conditions and the following disclaimer in the
  documentation and/or other materials provided with the distribution.

* Neither the names of Kitware, Inc., the Insight Software Consortium,
  nor the names of their contributors may be used to endorse or promote
  products derived from this software without specific prior written
  permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

A.9 Fred Fish's Dbug Library License

The following software may be included in this product:

Fred Fish's Dbug Library

```
                              N O T I C E

                 Copyright Abandoned, 1987, Fred Fish

     This previously copyrighted work has been placed into the  public
     domain  by  the  author  and  may be freely used for any purpose,
     private or commercial.

     Because of the number of inquiries I was receiving about the  use
     of this product in commercially developed works I have decided to
     simply make it public domain to further its unrestricted use.   I
     specifically  would  be  most happy to see this material become a
     part of the standard Unix distributions by AT&T and the  Berkeley
     Computer  Science  Research Group, and a standard part of the GNU
     system from the Free Software Foundation.

     I would appreciate it, as a courtesy, if this notice is  left  in
     all copies and derivative works.  Thank you.

     The author makes no warranty of any kind  with  respect  to  this
     product  and  explicitly disclaims any implied warranties of mer-
     chantability or fitness for any particular purpose.
The dbug_analyze.c file is subject to the following notice:

     Copyright June 1987, Binayak Banerjee
     All rights reserved.

     This program may be freely distributed under the same terms and
     conditions as Fred Fish's Dbug package.
```

A.10 `getarg` License

The following software may be included in this product:

`getarg` Function (`getarg.h`, `getarg.c` files)

```
Copyright (c) 1997 – 2000 Kungliga Tekniska Högskolan
(Royal Institute of Technology, Stockholm, Sweden).
All rights reserved.

Redistribution and use in source and binary forms, with
or without modification, are permitted provided that the
following conditions are met:

1. Redistributions of source code must retain the above
   copyright notice, this list of conditions and the
   following disclaimer.
2. Redistributions in binary form must reproduce the above
   copyright notice, this list of conditions and the following
   disclaimer in the documentation and/or other materials
   provided with the distribution.
3. Neither the name of the Institute nor the names of its
   contributors may be used to endorse or promote products
   derived from this software without specific prior written
   permission.

THIS SOFTWARE IS PROVIDED BY THE INSTITUTE AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY
AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO
EVENT SHALL THE INSTITUTE OR CONTRIBUTORS BE LIABLE FOR ANY
DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE
GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF
THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
DAMAGE.
```

A.11 GLib License (for MySQL Proxy)

The following software may be included in this product:

```
GLib

You are receiving a copy of the GLib library in both source
and object code in the following [proxy install dir]/lib/ and
[proxy install dir]/licenses/lgpl folders. The terms of the
Oracle license do NOT apply to the GLib library; it is licensed
under the following license, separately from the Oracle programs
you receive. If you do not wish to install this library, you may
create an "exclude" file and run tar with the X option, as in
the following example, but the Oracle program might not operate
properly or at all without the library:
  tar -xvfX <package-tar-file> <exclude-file>
where the exclude-file contains, e.g.:
  <package-name>/lib/libglib-2.0.so.0.1600.6
  <package-name>/lib/libglib-2.0.so.0
  ...

Example:
tar -xvfX mysql-proxy-0.8.1-solaris10-x86-64bit.tar.gz Exclude

Exclude File:
mysql-proxy-0.8.1-solaris10-x86-64bit/lib/libglib-2.0.so
mysql-proxy-0.8.1-solaris10-x86-64bit/lib/libglib-2.0.so.0
mysql-proxy-0.8.1-solaris10-x86-64bit/lib/libglib-2.0.so.0.1600.6
mysql-proxy-0.8.1-solaris10-x86-64bit/lib/libgmodule-2.0.so
```

```
mysql-proxy-0.8.1-solaris10-x86-64bit/lib/libgmodule-2.0.so.0
mysql-proxy-0.8.1-solaris10-x86-64bit/lib/libgmodule-2.0.so.0.1600.6
mysql-proxy-0.8.1-solaris10-x86-64bit/lib/libgthread-2.0.so
mysql-proxy-0.8.1-solaris10-x86-64bit/lib/libgthread-2.0.so.0
mysql-proxy-0.8.1-solaris10-x86-64bit/lib/libgthread-2.0.so.0.1600.6
mysql-proxy-0.8.1-solaris10-x86-64bit/licenses/lgpl/glib-2.16.6.tar.gz
```

This component is licensed under Section A.14, “GNU Lesser General Public License Version 2.1, February 1999”.

A.12 GNU General Public License Version 2.0, June 1991

```
The following applies to all products licensed under the GNU General
Public License, Version 2.0: You may not use the identified files
except in compliance with the GNU General Public License, Version
2.0 (the "License.") You may obtain a copy of the License at
http://www.gnu.org/licenses/gpl-2.0.txt. A copy of the license is
also reproduced below. Unless required by applicable law or agreed
to in writing, software distributed under the License is distributed
on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND,
either express or implied. See the License for the specific language
governing permissions and limitations under the License.

GNU GENERAL PUBLIC LICENSE
Version 2, June 1991

Copyright (C) 1989, 1991 Free Software Foundation, Inc.,
51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA
Everyone is permitted to copy and distribute verbatim
copies of this license document, but changing it is not
allowed.

                     Preamble

  The licenses for most software are designed to take away your
freedom to share and change it.  By contrast, the GNU General Public
License is intended to guarantee your freedom to share and change free
software--to make sure the software is free for all its users.  This
General Public License applies to most of the Free Software
Foundation's software and to any other program whose authors commit to
using it.  (Some other Free Software Foundation software is covered by
the GNU Lesser General Public License instead.)  You can apply it to
your programs, too.

  When we speak of free software, we are referring to freedom, not
price.  Our General Public Licenses are designed to make sure that you
have the freedom to distribute copies of free software (and charge for
this service if you wish), that you receive source code or can get it
if you want it, that you can change the software or use pieces of it
in new free programs; and that you know you can do these things.

  To protect your rights, we need to make restrictions that forbid
anyone to deny you these rights or to ask you to surrender the rights.
These restrictions translate to certain responsibilities for you if you
distribute copies of the software, or if you modify it.

  For example, if you distribute copies of such a program, whether
gratis or for a fee, you must give the recipients all the rights that
you have.  You must make sure that they, too, receive or can get the
source code.  And you must show them these terms so they know their
rights.

  We protect your rights with two steps: (1) copyright the software,
and (2) offer you this license which gives you legal permission to
```

```
copy, distribute and/or modify the software.

  Also, for each author's protection and ours, we want to make certain
that everyone understands that there is no warranty for this free
software.  If the software is modified by someone else and passed on,
we want its recipients to know that what they have is not the original,
so that any problems introduced by others will not reflect on the
original authors' reputations.

  Finally, any free program is threatened constantly by software
patents.  We wish to avoid the danger that redistributors of a free
program will individually obtain patent licenses, in effect making the
program proprietary.  To prevent this, we have made it clear that any
patent must be licensed for everyone's free use or not licensed at all.

  The precise terms and conditions for copying, distribution and
modification follow.

                    GNU GENERAL PUBLIC LICENSE
   TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

  0. This License applies to any program or other work which contains
a notice placed by the copyright holder saying it may be distributed
under the terms of this General Public License.  The "Program", below,
refers to any such program or work, and a "work based on the Program"
means either the Program or any derivative work under copyright law:
that is to say, a work containing the Program or a portion of it,
either verbatim or with modifications and/or translated into another
language.  (Hereinafter, translation is included without limitation in
the term "modification".)  Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not
covered by this License; they are outside its scope.  The act of
running the Program is not restricted, and the output from the Program
is covered only if its contents constitute a work based on the
Program (independent of having been made by running the Program).
Whether that is true depends on what the Program does.

  1. You may copy and distribute verbatim copies of the Program's
source code as you receive it, in any medium, provided that you
conspicuously and appropriately publish on each copy an appropriate
copyright notice and disclaimer of warranty; keep intact all the
notices that refer to this License and to the absence of any warranty;
and give any other recipients of the Program a copy of this License
along with the Program.

You may charge a fee for the physical act of transferring a copy, and
you may at your option offer warranty protection in exchange for a fee.

  2. You may modify your copy or copies of the Program or any portion
of it, thus forming a work based on the Program, and copy and
distribute such modifications or work under the terms of Section 1
above, provided that you also meet all of these conditions:

    a) You must cause the modified files to carry prominent notices
    stating that you changed the files and the date of any change.

    b) You must cause any work that you distribute or publish, that in
    whole or in part contains or is derived from the Program or any
    part thereof, to be licensed as a whole at no charge to all third
    parties under the terms of this License.

    c) If the modified program normally reads commands interactively
    when run, you must cause it, when started running for such
    interactive use in the most ordinary way, to print or display an
    announcement including an appropriate copyright notice and a
    notice that there is no warranty (or else, saying that you provide
    a warranty) and that users may redistribute the program under
    these conditions, and telling the user how to view a copy of this
    License.  (Exception: if the Program itself is interactive but
    does not normally print such an announcement, your work based on
    the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole.  If
identifiable sections of that work are not derived from the Program,
and can be reasonably considered independent and separate works in
themselves, then this License, and its terms, do not apply to those
sections when you distribute them as separate works.  But when you
distribute the same sections as part of a whole which is a work based
on the Program, the distribution of the whole must be on the terms of
this License, whose permissions for other licensees extend to the
entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest
your rights to work written entirely by you; rather, the intent is to
exercise the right to control the distribution of derivative or
collective works based on the Program.

In addition, mere aggregation of another work not based on the Program
with the Program (or with a work based on the Program) on a volume of
a storage or distribution medium does not bring the other work under
the scope of this License.

  3. You may copy and distribute the Program (or a work based on it,
under Section 2) in object code or executable form under the terms of
Sections 1 and 2 above provided that you also do one of the following:

    a) Accompany it with the complete corresponding machine-readable
    source code, which must be distributed under the terms of Sections
    1 and 2 above on a medium customarily used for software
    interchange; or,

    b) Accompany it with a written offer, valid for at least three
    years, to give any third party, for a charge no more than your
    cost of physically performing source distribution, a complete
    machine-readable copy of the corresponding source code, to be
    distributed under the terms of Sections 1 and 2 above on a medium
    customarily used for software interchange; or,

    c) Accompany it with the information you received as to the offer
    to distribute corresponding source code.  (This alternative is
    allowed only for noncommercial distribution and only if you
    received the program in object code or executable form with such
    an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for
making modifications to it.  For an executable work, complete source
code means all the source code for all modules it contains, plus any
associated interface definition files, plus the scripts used to
control compilation and installation of the executable.  However, as
a special exception, the source code distributed need not include
anything that is normally distributed (in either source or binary
form) with the major components (compiler, kernel, and so on) of the
operating system on which the executable runs, unless that component
itself accompanies the executable.

If distribution of executable or object code is made by offering
access to copy from a designated place, then offering equivalent
access to copy the source code from the same place counts as
distribution of the source code, even though third parties are not
compelled to copy the source along with the object code.

  4. You may not copy, modify, sublicense, or distribute the Program
except as expressly provided under this License.  Any attempt
```

```
otherwise to copy, modify, sublicense or distribute the Program is
void, and will automatically terminate your rights under this License.
However, parties who have received copies, or rights, from you under
this License will not have their licenses terminated so long as such
parties remain in full compliance.

  5. You are not required to accept this License, since you have not
signed it.  However, nothing else grants you permission to modify or
distribute the Program or its derivative works.  These actions are
prohibited by law if you do not accept this License.  Therefore, by
modifying or distributing the Program (or any work based on the
Program), you indicate your acceptance of this License to do so, and
all its terms and conditions for copying, distributing or modifying
the Program or works based on it.

  6. Each time you redistribute the Program (or any work based on the
Program), the recipient automatically receives a license from the
original licensor to copy, distribute or modify the Program subject to
these terms and conditions.  You may not impose any further
restrictions on the recipients' exercise of the rights granted herein.
You are not responsible for enforcing compliance by third parties to
this License.

  7. If, as a consequence of a court judgment or allegation of patent
infringement or for any other reason (not limited to patent issues),
conditions are imposed on you (whether by court order, agreement or
otherwise) that contradict the conditions of this License, they do not
excuse you from the conditions of this License.  If you cannot
distribute so as to satisfy simultaneously your obligations under this
License and any other pertinent obligations, then as a consequence you
may not distribute the Program at all.  For example, if a patent
license would not permit royalty-free redistribution of the Program by
all those who receive copies directly or indirectly through you, then
the only way you could satisfy both it and this License would be to
refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under
any particular circumstance, the balance of the section is intended to
apply and the section as a whole is intended to apply in other
circumstances.

It is not the purpose of this section to induce you to infringe any
patents or other property right claims or to contest validity of any
such claims; this section has the sole purpose of protecting the
integrity of the free software distribution system, which is
implemented by public license practices.  Many people have made
generous contributions to the wide range of software distributed
through that system in reliance on consistent application of that
system; it is up to the author/donor to decide if he or she is willing
to distribute software through any other system and a licensee cannot
impose that choice.

This section is intended to make thoroughly clear what is believed to
be a consequence of the rest of this License.

  8. If the distribution and/or use of the Program is restricted in
certain countries either by patents or by copyrighted interfaces, the
original copyright holder who places the Program under this License
may add an explicit geographical distribution limitation excluding
those countries, so that distribution is permitted only in or among
countries not thus excluded.  In such case, this License incorporates
the limitation as if written in the body of this License.

  9. The Free Software Foundation may publish revised and/or new
versions of the General Public License from time to time.  Such new
versions will be similar in spirit to the present version, but may
differ in detail to address new problems or concerns.
```

```
Each version is given a distinguishing version number.  If the Program
specifies a version number of this License which applies to it and
"any later version", you have the option of following the terms and
conditions either of that version or of any later version published by
the Free Software Foundation.  If the Program does not specify a
version number of this License, you may choose any version ever
published by the Free Software Foundation.

  10. If you wish to incorporate parts of the Program into other free
programs whose distribution conditions are different, write to the
author to ask for permission.  For software which is copyrighted by the
Free Software Foundation, write to the Free Software Foundation; we
sometimes make exceptions for this.  Our decision will be guided by the
two goals of preserving the free status of all derivatives of our free
software and of promoting the sharing and reuse of software generally.

                            NO WARRANTY

  11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO
WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW.
EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR
OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND,
EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.
THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS
WITH YOU.  SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF
ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

  12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN
WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY
AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU
FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR
CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE
PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING
RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A
FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF
SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH
DAMAGES.

                     END OF TERMS AND CONDITIONS

            How to Apply These Terms to Your New Programs

  If you develop a new program, and you want it to be of the greatest
possible use to the public, the best way to achieve this is to make it
free software which everyone can redistribute and change under these terms.

  To do so, attach the following notices to the program.  It is safest
to attach them to the start of each source file to most effectively
convey the exclusion of warranty; and each file should have at least
the "copyright" line and a pointer to where the full notice is found.

    <one line to give the program's name and a brief idea of what it
    does.>
    Copyright (C) <year>  <name of author>

    This program is free software; you can redistribute it and/or
    modify it under the terms of the GNU General Public License as
    published by     the Free Software Foundation; either version
    2 of the License, or (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.
    You should have received a copy of the GNU General Public License
    along with this program; if not, write to the Free Software
    Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA
    02110-1301 USA.

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this
when it starts in an interactive mode:

    Gnomovision version 69, Copyright (C) year name of author
    Gnomovision comes with ABSOLUTELY NO WARRANTY; for details
    type 'show w'. This is free software, and you are welcome
    to redistribute it under certain conditions; type 'show c'
    for details.

The hypothetical commands 'show w' and 'show c' should show the
appropriate parts of the General Public License.  Of course, the
commands you use may be called something other than 'show w' and
'show c'; they could even be mouse-clicks or menu items--whatever
suits your program.

You should also get your employer (if you work as a programmer) or your
school, if any, to sign a "copyright disclaimer" for the program, if
necessary.  Here is a sample; alter the names:

  Yoyodyne, Inc., hereby disclaims all copyright interest in the
  program 'Gnomovision' (which makes passes at compilers) written
  by James Hacker.

  <signature of Ty Coon>, 1 April 1989
  Ty Coon, President of Vice

This General Public License does not permit incorporating your program
into proprietary programs.  If your program is a subroutine library,
you may consider it more useful to permit linking proprietary
applications with the library.  If this is what you want to do, use
the GNU Lesser General Public License instead of this License.
```

A.13 GNU General Public License Version 3.0, 29 June 2007 and GCC Runtime Library Exception Version 3.1, 31 March 2009

```
                    GNU GENERAL PUBLIC LICENSE
                       Version 3, 29 June 2007

Copyright (C) 2007 Free Software Foundation, Inc. <http://fsf.org/>
Everyone is permitted to copy and distribute verbatim copies of
this license document, but changing it is not allowed.

                            Preamble

  The GNU General Public License is a free, copyleft license for
software and other kinds of works.

  The licenses for most software and other practical works are
designed to take away your freedom to share and change the works.
By contrast, the GNU General Public License is intended to guarantee
your freedom to share and change all versions of a program--to make
sure it remains free software for all its users.  We, the Free
Software Foundation, use the GNU General Public License for most
of our software; it applies also to any other work released this
way by its authors.  You can apply it to your programs, too.

  When we speak of free software, we are referring to freedom, not
```

price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS

0. Definitions.

"This License" refers to version 3 of the GNU General Public License.

"Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

"The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you". "Licensees" and "recipients" may be individuals or organizations.

To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

A "covered work" means either the unmodified Program or a work based on the Program.

To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.

The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work.

A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The "System Libraries" of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A "Major Component", in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users' Legal Rights From Anti-Circumvention Law.

No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

4. Conveying Verbatim Copies.

You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.

You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

```
    a) The work must carry prominent notices stating that you
    modified it, and giving a relevant date.

    b) The work must carry prominent notices stating that it is
    released under this License and any conditions added under
    section 7.  This requirement modifies the requirement in
    section 4 to "keep intact all notices".

    c) You must license the entire work, as a whole, under this
    License to anyone who comes into possession of a copy.  This
    License will therefore apply, along with any applicable section 7
    additional terms, to the whole of the work, and all its parts,
    regardless of how they are packaged.  This License gives no
    permission to license the work in any other way, but it does not
    invalidate such permission if you have separately received it.

    d) If the work has interactive user interfaces, each must display
    Appropriate Legal Notices; however, if the Program has
    interactive interfaces that do not display Appropriate Legal
    Notices, your work need not make them do so.

  A compilation of a covered work with other separate and independent
works, which are not by their nature extensions of the covered work,
and which are not combined with it such as to form a larger program,
in or on a volume of a storage or distribution medium, is called
an "aggregate" if the compilation and its resulting copyright are
not used to limit the access or legal rights of the compilation's
users beyond what the individual works permit.  Inclusion of a
covered work in an aggregate does not cause this License to apply
to the other parts of the aggregate.

  6. Conveying Non-Source Forms.

  You may convey a covered work in object code form under the terms
of sections 4 and 5, provided that you also convey the machine-readable
Corresponding Source under the terms of this License, in one of
these ways:

    a) Convey the object code in, or embodied in, a physical product
    (including a physical distribution medium), accompanied by the
    Corresponding Source fixed on a durable physical medium
    customarily used for software interchange.

    b) Convey the object code in, or embodied in, a physical product
    (including a physical distribution medium), accompanied by a
    written offer, valid for at least three years and valid for as
    long as you offer spare parts or customer support for that
    product model, to give anyone who possesses the object code
    either (1) a copy of the Corresponding Source for all the
    software in the product that is covered by this License, on a
    durable physical medium customarily used for software
    interchange, for a price no more than your reasonable cost
    of physically performing this conveying of source, or (2)
    access to copy the Corresponding Source from a network server
    at no charge.

    c) Convey individual copies of the object code with a copy of the
    written offer to provide the Corresponding Source.  This
    alternative is allowed only occasionally and noncommercially, and
    only if you received the object code with such an offer, in
    accord with subsection 6b.

    d) Convey the object code by offering access from a designated
    place (gratis or for a charge), and offer equivalent access to
    the Corresponding Source in the same way through the same place
    at no further charge.  You need not require recipients to copy
    the Corresponding Source along with the object code.  If the
    place to copy the object code is a network server, the
    Corresponding Source may be on a different server (operated
    by you or a third party) that supports equivalent copying
    facilities, provided you maintain clear directions next to the
    object code saying where to find the Corresponding Source.
    Regardless of what server hosts the Corresponding Source, you
    remain obligated to ensure that it is available for as long
    as needed to satisfy these requirements.

    e) Convey the object code using peer-to-peer transmission,
    provided you inform other peers where the object code and
    Corresponding Source of the work are being offered to the
    general public at no charge under subsection 6d.

  A separable portion of the object code, whose source code is excluded
from the Corresponding Source as a System Library, need not be
included in conveying the object code work.

  A "User Product" is either (1) a "consumer product", which means
any tangible personal property which is normally used for personal,
family, or household purposes, or (2) anything designed or sold for
incorporation into a dwelling.  In determining whether a product
is a consumer product, doubtful cases shall be resolved in favor
of coverage.  For a particular product received by a particular
user, "normally used" refers to a typical or common use of that
class of product, regardless of the status of the particular user
or of the way in which the particular user actually uses, or expects
or is expected to use, the product.  A product is a consumer product
regardless of whether the product has substantial commercial,
industrial or non-consumer uses, unless such uses represent the
only significant mode of use of the product.

  "Installation Information" for a User Product means any methods,
procedures, authorization keys, or other information required to
install and execute modified versions of a covered work in that
User Product from a modified version of its Corresponding Source.
The information must suffice to ensure that the continued functioning
of the modified object code is in no case prevented or interfered
with solely because modification has been made.

  If you convey an object code work under this section in, or with,
or specifically for use in, a User Product, and the conveying occurs
as part of a transaction in which the right of possession and use
of the User Product is transferred to the recipient in perpetuity
or for a fixed term (regardless of how the transaction is characterized),
the Corresponding Source conveyed under this section must be
accompanied by the Installation Information.  But this requirement
does not apply if neither you nor any third party retains the ability
to install modified object code on the User Product (for example,
the work has been installed in ROM).

  The requirement to provide Installation Information does not include
a requirement to continue to provide support service, warranty, or
updates for a work that has been modified or installed by the
recipient, or for the User Product in which it has been modified
or installed.  Access to a network may be denied when the modification
itself materially and adversely affects the operation of the network
or violates the rules and protocols for communication across the
network.

  Corresponding Source conveyed, and Installation Information provided,
in accord with this section must be in a format that is publicly
documented (and with an implementation available to the public in
source code form), and must require no special password or key for
unpacking, reading or copying.

  7. Additional Terms.
```

"Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or

b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or

c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or

d) Limiting the use for publicity purposes of names of licensors or authors of the material; or

e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or

f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies.

You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An "entity transaction" is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

11. Patents.

A "contributor" is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's "contributor version".

A contributor's "essential patent claims" are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, "control" includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a "patent license" is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To "grant" such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. "Knowingly relying" means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is "discriminatory" if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others' Freedom.

```
  If conditions are imposed on you (whether by court order, agreement
or otherwise) that contradict the conditions of this License, they
do not excuse you from the conditions of this License.  If you
cannot convey a covered work so as to satisfy simultaneously your
obligations under this License and any other pertinent obligations,
then as a consequence you may not convey it at all.  For example,
if you agree to terms that obligate you to collect a royalty for
further conveying from those to whom you convey the Program, the
only way you could satisfy both those terms and this License would
be to refrain entirely from conveying the Program.

  13. Use with the GNU Affero General Public License.

  Notwithstanding any other provision of this License, you have
permission to link or combine any covered work with a work licensed
under version 3 of the GNU Affero General Public License into a
single combined work, and to convey the resulting work.  The terms
of this License will continue to apply to the part which is the
covered work, but the special requirements of the GNU Affero General
Public License, section 13, concerning interaction through a network
will apply to the combination as such.

  14. Revised Versions of this License.

  The Free Software Foundation may publish revised and/or new versions
of the GNU General Public License from time to time.  Such new
versions will be similar in spirit to the present version, but may
differ in detail to address new problems or concerns.

  Each version is given a distinguishing version number.  If the
Program specifies that a certain numbered version of the GNU General
Public License "or any later version" applies to it, you have the
option of following the terms and conditions either of that numbered
version or of any later version published by the Free Software
Foundation.  If the Program does not specify a version number of
the GNU General Public License, you may choose any version ever
published by the Free Software Foundation.

  If the Program specifies that a proxy can decide which future
versions of the GNU General Public License can be used, that proxy's
public statement of acceptance of a version permanently authorizes
you to choose that version for the Program.

  Later license versions may give you additional or different
permissions.  However, no additional obligations are imposed on any
author or copyright holder as a result of your choosing to follow
a later version.

  15. Disclaimer of Warranty.

  THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY
APPLICABLE LAW.  EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT
HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT
WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT
NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
FOR A PARTICULAR PURPOSE.  THE ENTIRE RISK AS TO THE QUALITY AND
PERFORMANCE OF THE PROGRAM IS WITH YOU.  SHOULD THE PROGRAM PROVE
DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR
OR CORRECTION.

  16. Limitation of Liability.

  IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING
WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR
CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES,
INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES
```

```
ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING
BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE
OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE
PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER
OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

  17. Interpretation of Sections 15 and 16.

  If the disclaimer of warranty and limitation of liability provided
above cannot be given local legal effect according to their terms,
reviewing courts shall apply local law that most closely approximates
an absolute waiver of all civil liability in connection with the
Program, unless a warranty or assumption of liability accompanies a
copy of the Program in return for a fee.

                     END OF TERMS AND CONDITIONS

            How to Apply These Terms to Your New Programs

  If you develop a new program, and you want it to be of the greatest
possible use to the public, the best way to achieve this is to make
it free software which everyone can redistribute and change under
these terms.

To do so, attach the following notices to the program.  It is safest
to attach them to the start of each source file to most effectively
state the exclusion of warranty; and each file should have at least
the "copyright" line and a pointer to where the full notice is
found.

    <one line to give the program's name and a brief idea of what it does.>
    Copyright (C) <year>  <name of author>

    This program is free software: you can redistribute it and/or
    modify it under the terms of the GNU General Public License
    as published by the Free Software Foundation, either version 3
    of the License, or (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License
    along with this program.  If not, see
    <http://www.gnu.org/licenses/>.

Also add information on how to contact you by electronic and paper mail.

  If the program does terminal interaction, make it output a short
notice like this when it starts in an interactive mode:

    <program>  Copyright (C) <year>  <name of author>
    This program comes with ABSOLUTELY NO WARRANTY; for details type 'show w'.
    This is free software, and you are welcome to redistribute it
    under certain conditions; type 'show c' for details.

The hypothetical commands 'show w' and 'show c' should show the
appropriate parts of the General Public License.  Of course, your
program's commands might be different; for a GUI interface, you
would use an "about box".

  You should also get your employer (if you work as a programmer) or
school, if any, to sign a "copyright disclaimer" for the program,
if necessary. For more information on this, and how to apply and
follow the GNU GPL, see <http://www.gnu.org/licenses/>.
```

The GNU General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License. But first, please read <http://www.gnu.org/philosophy/why-not-lgpl.html>.

==

==

GCC RUNTIME LIBRARY EXCEPTION

Version 3.1, 31 March 2009

This GCC Runtime Library Exception ("Exception") is an additional permission under section 7 of the GNU General Public License, version 3 ("GPLv3"). It applies to a given file (the "Runtime Library") that bears a notice placed by the copyright holder of the file stating that the file is governed by GPLv3 along with this Exception.

When you use GCC to compile a program, GCC may combine portions of certain GCC header files and runtime libraries with the compiled program. The purpose of this Exception is to allow compilation of non-GPL (including proprietary) programs to use, in this way, the header files and runtime libraries covered by this Exception.

0. Definitions.

A file is an "Independent Module" if it either requires the Runtime Library for execution after a Compilation Process, or makes use of an interface provided by the Runtime Library, but is not otherwise based on the Runtime Library.

"GCC" means a version of the GNU Compiler Collection, with or without modifications, governed by version 3 (or a specified later version) of the GNU General Public License (GPL) with the option of using any subsequent versions published by the FSF.

"GPL-compatible Software" is software whose conditions of propagation, modification and use would permit combination with GCC in accord with the license of GCC.

"Target Code" refers to output from any compiler for a real or virtual target processor architecture, in executable form or suitable for input to an assembler, loader, linker and/or execution phase. Notwithstanding that, Target Code does not include data in any format that is used as a compiler intermediate representation, or used for producing a compiler intermediate representation.

The "Compilation Process" transforms code entirely represented in non-intermediate languages designed for human-written code, and/or in Java Virtual Machine byte code, into Target Code. Thus, for example, use of source code generators and preprocessors need not be considered part of the Compilation Process, since the Compilation Process can be understood as starting with the output of the generators or preprocessors.

A Compilation Process is "Eligible" if it is done using GCC, alone or with other GPL-compatible software, or if it is done without using any work based on GCC. For example, using non-GPL-compatible Software to optimize any GCC intermediate representations would not qualify as an Eligible Compilation Process.

```
1. Grant of Additional Permission.

You have permission to propagate a work of Target Code formed by
combining the Runtime Library with Independent Modules, even if
such propagation would otherwise violate the terms of GPLv3, provided
that all Target Code was generated by Eligible Compilation Processes.
You may then convey such a combination under terms of your choice,
consistent with the licensing of the Independent Modules.

2. No Weakening of GCC Copyleft.

The availability of this Exception does not imply any general
presumption that third-party software is unaffected by the copyleft
requirements of the license of GCC.
==
```

A.14 GNU Lesser General Public License Version 2.1, February 1999

```
The following applies to all products licensed under the
GNU Lesser General Public License, Version 2.1: You may
not use the identified files except in compliance with
the GNU Lesser General Public License, Version 2.1 (the
"License"). You may obtain a copy of the License at
http://www.gnu.org/licenses/lgpl-2.1.html. A copy of the
license is also reproduced below. Unless required by
applicable law or agreed to in writing, software distributed
under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express
or implied. See the License for the specific language governing
permissions and limitations under the License.

                  GNU LESSER GENERAL PUBLIC LICENSE
                       Version 2.1, February 1999

 Copyright (C) 1991, 1999 Free Software Foundation, Inc.
 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301  USA
 Everyone is permitted to copy and distribute verbatim copies
 of this license document, but changing it is not allowed.

[This is the first released version of the Lesser GPL.  It also counts
 as the successor of the GNU Library Public License, version 2, hence
 the version number 2.1.]

                            Preamble

  The licenses for most software are designed to take away your
freedom to share and change it.  By contrast, the GNU General Public
Licenses are intended to guarantee your freedom to share and change
free software--to make sure the software is free for all its users.

  This license, the Lesser General Public License, applies to some
specially designated software packages--typically libraries--of the
Free Software Foundation and other authors who decide to use it.  You
can use it too, but we suggest you first think carefully about whether
this license or the ordinary General Public License is the better
strategy to use in any particular case, based on the explanations below.

  When we speak of free software, we are referring to freedom of use,
not price.  Our General Public Licenses are designed to make sure that
you have the freedom to distribute copies of free software (and charge
for this service if you wish); that you receive source code or can get
it if you want it; that you can change the software and use pieces of
it in new free programs; and that you are informed that you can do
these things.
```

```
  To protect your rights, we need to make restrictions that forbid
distributors to deny you these rights or to ask you to surrender these
rights.  These restrictions translate to certain responsibilities for
you if you distribute copies of the library or if you modify it.

  For example, if you distribute copies of the library, whether gratis
or for a fee, you must give the recipients all the rights that we gave
you.  You must make sure that they, too, receive or can get the source
code.  If you link other code with the library, you must provide
complete object files to the recipients, so that they can relink them
with the library after making changes to the library and recompiling
it.  And you must show them these terms so they know their rights.

  We protect your rights with a two-step method: (1) we copyright the
library, and (2) we offer you this license, which gives you legal
permission to copy, distribute and/or modify the library.

  To protect each distributor, we want to make it very clear that
there is no warranty for the free library.  Also, if the library is
modified by someone else and passed on, the recipients should know
that what they have is not the original version, so that the original
author's reputation will not be affected by problems that might be
introduced by others.

  Finally, software patents pose a constant threat to the existence of
any free program.  We wish to make sure that a company cannot
effectively restrict the users of a free program by obtaining a
restrictive license from a patent holder.  Therefore, we insist that
any patent license obtained for a version of the library must be
consistent with the full freedom of use specified in this license.

  Most GNU software, including some libraries, is covered by the
ordinary GNU General Public License.  This license, the GNU Lesser
General Public License, applies to certain designated libraries, and
is quite different from the ordinary General Public License.  We use
this license for certain libraries in order to permit linking those
libraries into non-free programs.

  When a program is linked with a library, whether statically or using
a shared library, the combination of the two is legally speaking a
combined work, a derivative of the original library.  The ordinary
General Public License therefore permits such linking only if the
entire combination fits its criteria of freedom.  The Lesser General
Public License permits more lax criteria for linking other code with
the library.

  We call this license the "Lesser" General Public License because it
does Less to protect the user's freedom than the ordinary General
Public License.  It also provides other free software developers Less
of an advantage over competing non-free programs.  These disadvantages
are the reason we use the ordinary General Public License for many
libraries.  However, the Lesser license provides advantages in certain
special circumstances.

  For example, on rare occasions, there may be a special need to
encourage the widest possible use of a certain library, so that it
becomes a de-facto standard.  To achieve this, non-free programs
must be allowed to use the library.  A more frequent case is that
a free library does the same job as widely used non-free libraries.
In this case, there is little to gain by limiting the free library
to free software only, so we use the Lesser General Public License.

  In other cases, permission to use a particular library in non-free
programs enables a greater number of people to use a large body of
free software.  For example, permission to use the GNU C Library in
non-free programs enables many more people to use the whole GNU
operating system, as well as its variant, the GNU/Linux operating
```

```
system.

  Although the Lesser General Public License is Less protective of the
users' freedom, it does ensure that the user of a program that is
linked with the Library has the freedom and the wherewithal to run
that program using a modified version of the Library.

  The precise terms and conditions for copying, distribution and
modification follow.  Pay close attention to the difference between a
"work based on the library" and a "work that uses the library".  The
former contains code derived from the library, whereas the latter must
be combined with the library in order to run.

                  GNU LESSER GENERAL PUBLIC LICENSE
   TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

  0. This License Agreement applies to any software library or other
program which contains a notice placed by the copyright holder or
other authorized party saying it may be distributed under the terms of
this Lesser General Public License (also called "this License").
Each licensee is addressed as "you".

  A "library" means a collection of software functions and/or data
prepared so as to be conveniently linked with application programs
(which use some of those functions and data) to form executables.

  The "Library", below, refers to any such software library or work
which has been distributed under these terms.  A "work based on the
Library" means either the Library or any derivative work under
copyright law: that is to say, a work containing the Library or a
portion of it, either verbatim or with modifications and/or translated
straightforwardly into another language.  (Hereinafter, translation is
included without limitation in the term "modification".)

  "Source code" for a work means the preferred form of the work for
making modifications to it.  For a library, complete source code means
all the source code for all modules it contains, plus any associated
interface definition files, plus the scripts used to control
compilation and installation of the library.

  Activities other than copying, distribution and modification are not
covered by this License; they are outside its scope.  The act of
running a program using the Library is not restricted, and output from
such a program is covered only if its contents constitute a work based
on the Library (independent of the use of the Library in a tool for
writing it).  Whether that is true depends on what the Library does
and what the program that uses the Library does.

  1. You may copy and distribute verbatim copies of the Library's
complete source code as you receive it, in any medium, provided that
you conspicuously and appropriately publish on each copy an
appropriate copyright notice and disclaimer of warranty; keep intact
all the notices that refer to this License and to the absence of any
warranty; and distribute a copy of this License along with the
Library.

  You may charge a fee for the physical act of transferring a copy,
and you may at your option offer warranty protection in exchange for a
fee.

  2. You may modify your copy or copies of the Library or any portion
of it, thus forming a work based on the Library, and copy and
distribute such modifications or work under the terms of Section 1
above, provided that you also meet all of these conditions:

    a) The modified work must itself be a software library.
    b) You must cause the files modified to carry prominent notices
    stating that you changed the files and the date of any change.

    c) You must cause the whole of the work to be licensed at no
    charge to all third parties under the terms of this License.

    d) If a facility in the modified Library refers to a function or a
    table of data to be supplied by an application program that uses
    the facility, other than as an argument passed when the facility
    is invoked, then you must make a good faith effort to ensure that,
    in the event an application does not supply such function or
    table, the facility still operates, and performs whatever part of
    its purpose remains meaningful.

    (For example, a function in a library to compute square roots has
    a purpose that is entirely well-defined independent of the
    application.  Therefore, Subsection 2d requires that any
    application-supplied function or table used by this function must
    be optional: if the application does not supply it, the square
    root function must still compute square roots.)

These requirements apply to the modified work as a whole.  If
identifiable sections of that work are not derived from the Library,
and can be reasonably considered independent and separate works in
themselves, then this License, and its terms, do not apply to those
sections when you distribute them as separate works.  But when you
distribute the same sections as part of a whole which is a work based
on the Library, the distribution of the whole must be on the terms of
this License, whose permissions for other licensees extend to the
entire whole, and thus to each and every part regardless of who wrote
it.

Thus, it is not the intent of this section to claim rights or contest
your rights to work written entirely by you; rather, the intent is to
exercise the right to control the distribution of derivative or
collective works based on the Library.

In addition, mere aggregation of another work not based on the Library
with the Library (or with a work based on the Library) on a volume of
a storage or distribution medium does not bring the other work under
the scope of this License.

  3. You may opt to apply the terms of the ordinary GNU General Public
License instead of this License to a given copy of the Library.  To do
this, you must alter all the notices that refer to this License, so
that they refer to the ordinary GNU General Public License, version 2,
instead of to this License.  (If a newer version than version 2 of the
ordinary GNU General Public License has appeared, then you can specify
that version instead if you wish.)  Do not make any other change in
these notices.

  Once this change is made in a given copy, it is irreversible for
that copy, so the ordinary GNU General Public License applies to all
subsequent copies and derivative works made from that copy.

  This option is useful when you wish to copy part of the code of
the Library into a program that is not a library.

  4. You may copy and distribute the Library (or a portion or
derivative of it, under Section 2) in object code or executable form
under the terms of Sections 1 and 2 above provided that you accompany
it with the complete corresponding machine-readable source code, which
must be distributed under the terms of Sections 1 and 2 above on a
medium customarily used for software interchange.

  If distribution of object code is made by offering access to copy
from a designated place, then offering equivalent access to copy the
```

```
source code from the same place satisfies the requirement to
distribute the source code, even though third parties are not
compelled to copy the source along with the object code.

  5. A program that contains no derivative of any portion of the
Library, but is designed to work with the Library by being compiled or
linked with it, is called a "work that uses the Library".  Such a
work, in isolation, is not a derivative work of the Library, and
therefore falls outside the scope of this License.

  However, linking a "work that uses the Library" with the Library
creates an executable that is a derivative of the Library (because it
contains portions of the Library), rather than a "work that uses the
library".  The executable is therefore covered by this License.
Section 6 states terms for distribution of such executables.

  When a "work that uses the Library" uses material from a header file
that is part of the Library, the object code for the work may be a
derivative work of the Library even though the source code is not.
Whether this is true is especially significant if the work can be
linked without the Library, or if the work is itself a library.  The
threshold for this to be true is not precisely defined by law.

  If such an object file uses only numerical parameters, data
structure layouts and accessors, and small macros and small inline
functions (ten lines or less in length), then the use of the object
file is unrestricted, regardless of whether it is legally a derivative
work.  (Executables containing this object code plus portions of the
Library will still fall under Section 6.)

  Otherwise, if the work is a derivative of the Library, you may
distribute the object code for the work under the terms of Section 6.
Any executables containing that work also fall under Section 6,
whether or not they are linked directly with the Library itself.

  6. As an exception to the Sections above, you may also combine or
link a "work that uses the Library" with the Library to produce a
work containing portions of the Library, and distribute that work
under terms of your choice, provided that the terms permit
modification of the work for the customer's own use and reverse
engineering for debugging such modifications.

  You must give prominent notice with each copy of the work that the
Library is used in it and that the Library and its use are covered by
this License.  You must supply a copy of this License.  If the work
during execution displays copyright notices, you must include the
copyright notice for the Library among them, as well as a reference
directing the user to the copy of this License.  Also, you must do one
of these things:

    a) Accompany the work with the complete corresponding
    machine-readable source code for the Library including whatever
    changes were used in the work (which must be distributed under
    Sections 1 and 2 above); and, if the work is an executable linked
    with the Library, with the complete machine-readable "work that
    uses the Library", as object code and/or source code, so that the
    user can modify the Library and then relink to produce a modified
    executable containing the modified Library.  (It is understood
    that the user who changes the contents of definitions files in the
    Library will not necessarily be able to recompile the application
    to use the modified definitions.)

    b) Use a suitable shared library mechanism for linking with the
    Library.  A suitable mechanism is one that (1) uses at run time a
    copy of the library already present on the user's computer system,
    rather than copying library functions into the executable, and (2)
    will operate properly with a modified version of the library, if
    the user installs one, as long as the modified version is
    interface-compatible with the version that the work was made with.

    c) Accompany the work with a written offer, valid for at
    least three years, to give the same user the materials
    specified in Subsection 6a, above, for a charge no more
    than the cost of performing this distribution.

    d) If distribution of the work is made by offering access to copy
    from a designated place, offer equivalent access to copy the above
    specified materials from the same place.

    e) Verify that the user has already received a copy of these
    materials or that you have already sent this user a copy.

  For an executable, the required form of the "work that uses the
Library" must include any data and utility programs needed for
reproducing the executable from it.  However, as a special exception,
the materials to be distributed need not include anything that is
normally distributed (in either source or binary form) with the major
components (compiler, kernel, and so on) of the operating system on
which the executable runs, unless that component itself accompanies
the executable.

  It may happen that this requirement contradicts the license
restrictions of other proprietary libraries that do not normally
accompany the operating system.  Such a contradiction means you cannot
use both them and the Library together in an executable that you
distribute.

  7. You may place library facilities that are a work based on the
Library side-by-side in a single library together with other library
facilities not covered by this License, and distribute such a combined
library, provided that the separate distribution of the work based on
the Library and of the other library facilities is otherwise
permitted, and provided that you do these two things:

    a) Accompany the combined library with a copy of the same work
    based on the Library, uncombined with any other library
    facilities.  This must be distributed under the terms of the
    Sections above.

    b) Give prominent notice with the combined library of the fact
    that part of it is a work based on the Library, and explaining
    where to find the accompanying uncombined form of the same work.

  8. You may not copy, modify, sublicense, link with, or distribute
the Library except as expressly provided under this License.  Any
attempt otherwise to copy, modify, sublicense, link with, or
distribute the Library is void, and will automatically terminate your
rights under this License.  However, parties who have received copies,
or rights, from you under this License will not have their licenses
terminated so long as such parties remain in full compliance.

  9. You are not required to accept this License, since you have not
signed it.  However, nothing else grants you permission to modify or
distribute the Library or its derivative works.  These actions are
prohibited by law if you do not accept this License.  Therefore, by
modifying or distributing the Library (or any work based on the
Library), you indicate your acceptance of this License to do so, and
all its terms and conditions for copying, distributing or modifying
the Library or works based on it.

  10. Each time you redistribute the Library (or any work based on the
Library), the recipient automatically receives a license from the
original licensor to copy, distribute, link with or modify the Library
subject to these terms and conditions.  You may not impose any further
```

```
restrictions on the recipients' exercise of the rights granted herein.
You are not responsible for enforcing compliance by third parties with
this License.

  11. If, as a consequence of a court judgment or allegation of patent
infringement or for any other reason (not limited to patent issues),
conditions are imposed on you (whether by court order, agreement or
otherwise) that contradict the conditions of this License, they do not
excuse you from the conditions of this License.  If you cannot
distribute so as to satisfy simultaneously your obligations under this
License and any other pertinent obligations, then as a consequence you
may not distribute the Library at all.  For example, if a patent
license would not permit royalty-free redistribution of the Library by
all those who receive copies directly or indirectly through you, then
the only way you could satisfy both it and this License would be to
refrain entirely from distribution of the Library.

If any portion of this section is held invalid or unenforceable under
any particular circumstance, the balance of the section is intended
to apply, and the section as a whole is intended to apply in other
circumstances.

It is not the purpose of this section to induce you to infringe any
patents or other property right claims or to contest validity of any
such claims; this section has the sole purpose of protecting the
integrity of the free software distribution system which is
implemented by public license practices.  Many people have made
generous contributions to the wide range of software distributed
through that system in reliance on consistent application of that
system; it is up to the author/donor to decide if he or she is willing
to distribute software through any other system and a licensee cannot
impose that choice.

This section is intended to make thoroughly clear what is believed to
be a consequence of the rest of this License.

  12. If the distribution and/or use of the Library is restricted in
certain countries either by patents or by copyrighted interfaces, the
original copyright holder who places the Library under this License
may add an explicit geographical distribution limitation excluding
those countries, so that distribution is permitted only in or among
countries not thus excluded.  In such case, this License incorporates
the limitation as if written in the body of this License.

  13. The Free Software Foundation may publish revised and/or new
versions of the Lesser General Public License from time to time.
Such new versions will be similar in spirit to the present version,
but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number.  If the Library
specifies a version number of this License which applies to it and
"any later version", you have the option of following the terms and
conditions either of that version or of any later version published by
the Free Software Foundation.  If the Library does not specify a
license version number, you may choose any version ever published by
the Free Software Foundation.

  14. If you wish to incorporate parts of the Library into other free
programs whose distribution conditions are incompatible with these,
write to the author to ask for permission.  For software which is
copyrighted by the Free Software Foundation, write to the Free
Software Foundation; we sometimes make exceptions for this.  Our
decision will be guided by the two goals of preserving the free status
of all derivatives of our free software and of promoting the sharing
and reuse of software generally.

                            NO WARRANTY
  15. BECAUSE THE LIBRARY IS LICENSED FREE OF CHARGE, THERE IS NO
WARRANTY FOR THE LIBRARY, TO THE EXTENT PERMITTED BY APPLICABLE LAW.
EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR
OTHER PARTIES PROVIDE THE LIBRARY "AS IS" WITHOUT WARRANTY OF ANY
KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
PURPOSE.  THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE
LIBRARY IS WITH YOU.  SHOULD THE LIBRARY PROVE DEFECTIVE, YOU ASSUME
THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

  16. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN
WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY
AND/OR REDISTRIBUTE THE LIBRARY AS PERMITTED ABOVE, BE LIABLE TO YOU
FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR
CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE
LIBRARY (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING
RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A
FAILURE OF THE LIBRARY TO OPERATE WITH ANY OTHER SOFTWARE), EVEN IF
SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH
DAMAGES.

                     END OF TERMS AND CONDITIONS

           How to Apply These Terms to Your New Libraries

  If you develop a new library, and you want it to be of the greatest
possible use to the public, we recommend making it free software that
everyone can redistribute and change.  You can do so by permitting
redistribution under these terms (or, alternatively, under the terms
of the ordinary General Public License).

  To apply these terms, attach the following notices to the library.
It is safest to attach them to the start of each source file to most
effectively convey the exclusion of warranty; and each file should
have at least the "copyright" line and a pointer to where the full
notice is found.

    <one line to give the library's name and a brief idea of what it does.>
    Copyright (C) <year>  <name of author>

    This library is free software; you can redistribute it and/or
    modify it under the terms of the GNU Lesser General Public
    License as published by the Free Software Foundation; either
    version 2.1 of the License, or (at your option) any later version.

    This library is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
    Lesser General Public License for more details.

    You should have received a copy of the GNU Lesser General Public
    License along with this library; if not, write to the Free Software
    Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA
    02110-1301  USA

Also add information on how to contact you by electronic and paper mail.

You should also get your employer (if you work as a programmer) or your
school, if any, to sign a "copyright disclaimer" for the library, if
necessary.  Here is a sample; alter the names:

  Yoyodyne, Inc., hereby disclaims all copyright interest in the
  library `Frob' (a library for tweaking knobs) written by James
  Random Hacker.

  <signature of Ty Coon>, 1 April 1990
  Ty Coon, President of Vice

That's all there is to it!
```

A.15 GNU Readline License

The following software may be included in this product:

GNU Readline Library

```
GNU Readline Library
With respect to MySQL Server/Cluster software licensed
under GNU General Public License, you are receiving a
copy of the GNU Readline Library in source code. The
terms of any Oracle license that might accompany the
Oracle programs do NOT apply to the GNU Readline Library;
it is licensed under the following license, separately
from the Oracle programs you receive. Oracle elects to
use GNU General Public License version 2 (GPL) for any
software where a choice of GPL license versions are
made available with the language indicating that GPLv2
or any later version may be used, or where a choice of
which version of the GPL is applied is unspecified.
```

This component is licensed under Section A.12, “GNU General Public License Version 2.0, June 1991”

A.16 GNU Standard C++ Library (libstdc++) License

The following software may be included in this product: GNU Standard C++ Library (libstdc++)

This component is licensed under Section A.13, “GNU General Public License Version 3.0, 29 June 2007 and GCC Runtime Library Exception Version 3.1, 31 March 2009”.

Additional notices:

```
==
 Copyright (c) 1994
 Hewlett-Packard Company

 Permission to use, copy, modify, distribute and sell this software
 and its documentation for any purpose is hereby granted without fee,
 provided that the above copyright notice appear in all copies and
 that both that copyright notice and this permission notice appear
 in supporting documentation.  Hewlett-Packard Company makes no
 representations about the suitability of this software for any
 purpose.  It is provided "as is" without express or implied
 warranty.
==

==
 Copyright (c) 1996,1997
 Silicon Graphics Computer Systems, Inc.

 Permission to use, copy, modify, distribute and sell this software
 and its documentation for any purpose is hereby granted without fee,
 provided that the above copyright notice appear in all copies and
 that both that copyright notice and this permission notice appear
 in supporting documentation.  Silicon Graphics makes no
 representations about the suitability of this software for any
 purpose.  It is provided "as is" without express or implied
 warranty.
==

==
 shared_count.hpp
@  Copyright (c) 2001, 2002, 2003 Peter Dimov and Multi Media Ltd.

 shared_ptr.hpp
 Copyright (C) 1998, 1999 Greg Colvin and Beman Dawes.
 Copyright (C) 2001, 2002, 2003 Peter Dimov

 weak_ptr.hpp
 Copyright (C) 2001, 2002, 2003 Peter Dimov

 enable_shared_from_this.hpp
 Copyright (C) 2002 Peter Dimov

Distributed under the Boost Software License, Version 1.0.

Boost Software License - Version 1.0 - August 17th, 2003

Permission is hereby granted, free of charge, to any person or
organization obtaining a copy of the software and accompanying
documentation covered by this license (the "Software") to use,
reproduce, display, distribute, execute, and transmit the Software,
and to prepare derivative works of the Software, and to permit
third-parties to whom the Software is furnished to do so, all subject
to the following:

The copyright notices in the Software and this entire statement,
including the above license grant, this restriction and the following
disclaimer, must be included in all copies of the Software, in whole
or in part, and all derivative works of the Software, unless such
copies or derivative works are solely in the form of machine-executable
object code generated by a source language processor.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND
NON-INFRINGEMENT. IN NO EVENT SHALL THE COPYRIGHT HOLDERS OR ANYONE
DISTRIBUTING THE SOFTWARE BE LIABLE FOR ANY DAMAGES OR OTHER
LIABILITY, WHETHER IN CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
DEALINGS IN THE SOFTWARE.
==

==
Copyright (C) 2004 Ami Tavory and Vladimir Dreizin, IBM-HRL.

Permission to use, copy, modify, sell, and distribute this software
is hereby granted without fee, provided that the above copyright
notice appears in all copies, and that both that copyright notice
and this permission notice appear in supporting documentation. None
of the above authors, nor IBM Haifa Research Laboratories, make any
representation about the suitability of this software for any
purpose. It is provided "as is" without express or implied warranty.
==
```

A.17 Google Controlling Master Thread I/O Rate Patch License

The following software may be included in this product:

Google Controlling master thread I/O rate patch

```

Copyright (c) 2009, Google Inc.
```

```
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:

 * Redistributions of source code must retain the above copyright
   notice, this list of conditions and the following disclaimer.
 * Redistributions in binary form must reproduce the above copyright
   notice, this list of conditions and the following disclaimer in the
   documentation and/or other materials provided with the distribution.
 * Neither the name of the Google Inc. nor the names of its contributors
   may be used to endorse or promote products derived from this software
   without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN
ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
POSSIBILITY OF SUCH DAMAGE.
```

A.18 Google Perftools (TCMalloc utility) License

The following software may be included in this product:

```
Google Perftools (TCMalloc utility)

Copyright (c) 1998-2006, Google Inc.
All rights reserved.

Redistribution and use in source and binary forms, with or
without modification, are permitted provided that the following
conditions are met:

    * Redistributions of source code must retain the above
      copyright notice, this list of conditions and the following
      disclaimer.
    * Redistributions in binary form must reproduce the above
      copyright notice, this list of conditions and the following
      disclaimer in the documentation and/or other materials provided
      with the distribution.
    * Neither the name of Google Inc. nor the names of its
      contributors may be used to endorse or promote products
      derived from this software without specific prior written
      permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

A.19 Google SMP Patch License

The following software may be included in this product:

Google SMP Patch

```
Google SMP patch

Copyright (c) 2008, Google Inc.
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:

 * Redistributions of source code must retain the above copyright
   notice, this list of conditions and the following disclaimer.
 * Redistributions in binary form must reproduce the above copyright
   notice, this list of conditions and the following disclaimer in the
   documentation and/or other materials provided with the distribution.
 * Neither the name of the Google Inc. nor the names of its contributors
   may be used to endorse or promote products derived from this software
   without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN
ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
POSSIBILITY OF SUCH DAMAGE.
```

A.20 `lib_sql.cc` License

The following software may be included in this product:

`lib_sql.cc`

```
Copyright (c) 2000
SWsoft company

This material is provided "as is", with absolutely no warranty
expressed or implied. Any use is at your own risk.

Permission to use or copy this software for any purpose is hereby
granted without fee, provided the above notices are retained on
all copies. Permission to modify the code and to distribute modified
code is granted, provided the above notices are retained, and a
notice that the code was modified is included with the above copyright
notice.

This code was modified by the MySQL team.
```

A.21 Libaio License

The following software may be included in this product:

```
libaio
```

This component is licensed under Section A.14, “GNU Lesser General Public License Version 2.1, February 1999”.

A.22 libevent License

The following software may be included in this product:

```
libevent

Copyright (c) 2000-2007 Niels Provos <provos@citi.umich.edu>
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:
1. Redistributions of source code must retain the above copyright
   notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright
   notice, this list of conditions and the following disclaimer in the
   documentation and/or other materials provided with the distribution.
3. The name of the author may not be used to endorse or promote products
   derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE AUTHOR ``AS IS'' AND ANY EXPRESS OR
IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES
OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED.
IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF
THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE

==
Parts developed by Adam Langley
==

==
log.c
Based on err.c, which was adapted from OpenBSD libc *err*warncode.

Copyright (c) 2005 Nick Mathewson
Copyright (c) 2000 Dug Song
Copyright (c) 1993 The Regents of the University of California.
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:
1. Redistributions of source code must retain the above copyright
   notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright
   notice, this list of conditions and the following disclaimer in
   the documentation and/or other materials provided with the
   distribution.
3. Neither the name of the University nor the names of its
   contributors may be used to endorse or promote products derived
   from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS "AS IS"
AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED
```

```
TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE REGENTS
OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED
AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
THE POSSIBILITY OF SUCH DAMAGE.
==

==
min_heap.h

Copyright (c) 2006 Maxim Yegorushkin
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:
1. Redistributions of source code must retain the above copyright
   notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright
   notice, this list of conditions and the following disclaimer in
   the documentation and/or other materials provided with the
   distribution.
3. The name of the author may not be used to endorse or promote
   products derived from this software without specific prior
   written permission.

THIS SOFTWARE IS PROVIDED BY THE AUTHOR "AS IS" AND ANY EXPRESS
OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY
DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE
GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER
IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR
OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN
IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
==

==
win32.c

Copyright 2000-2002 Niels Provos
Copyright 2003 Michael A. Davis
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:
1. Redistributions of source code must retain the above copyright
   notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright
   notice, this list of conditions and the following disclaimer in
   the documentation and/or other materials provided with the
   distribution.
3.  The name of the author may not be used to endorse or promote
    products derived from this software without specific prior
    written permission.

THIS SOFTWARE IS PROVIDED BY THE AUTHOR "AS IS" AND ANY EXPRESS
OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
```

```
ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY
DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE
GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER
IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR
OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN
IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
==
```

A.23 Libiconv License

The following software may be included in this product:

```
Libiconv

You are receiving a copy of the GNU LIBICONV Library. The terms of the Oracle
license do NOT apply to the GNU LIBICONV Library; it is licensed under the
following license, separately from the Oracle programs you receive. If you do
not wish to install this program, you may delete [agent install
dir]/lib/libiconv.* and [agent install dir]/licenses/lgpl/iconv files.
```

This component is licensed under Section A.14, “GNU Lesser General Public License Version 2.1, February 1999”.

A.24 `libintl` License

The following software may be included in this product:

libintl

```
Copyright (C) 1994 X Consortium

Permission is hereby granted, free of charge, to any person obtaining a copy of
this software and associated documentation files (the "Software"), to deal in
the Software without restriction, including without limitation the rights to
use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of
the Software, and to permit persons to whom the Software is furnished to do so,
subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS
FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE X CONSORTIUM
BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF
CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Except as contained in this notice, the name of the X Consortium shall not be
used in advertising or otherwise to promote the sale, use or other dealings in
this Software without prior written authorization from the X Consortium.

FSF changes to this file are in the public domain.
 .
Copyright 1996-2007 Free Software Foundation, Inc. Taken from GNU libtool, 2001

Originally by Gordon Matzigkeit <gord@gnu.ai.mit.edu>, 1996

This file is free software; the Free Software Foundation gives unlimited
permission to copy and/or distribute it, with or without modifications, as long
as this notice is preserved.
```

```
.
You are receiving a copy of the libintl library. The terms of the Oracle license
do NOT apply to the libintl library; it is licensed under the following license,
separately from the Oracle programs you receive. If you do not wish to install
this program, you may create an "exclude" file and run tar with the X option.

This component is licensed under Section A.14, “GNU Lesser General Public License Version 2.1, February 19
```

A.25 Linux-PAM License

The following software may be included in this product:

```

Linux-PAM (pam-devel, Pluggable authentication modules for Linux)

Copyright Theodore Ts'o, 1996. All rights reserved.

(For the avoidance of doubt, Oracle uses and distributes this
component under the terms below and elects not to do so under
the GPL even though the GPL is referenced as an option below.)

Redistribution and use in source and binary forms, with or
without modification, are permitted provided that the following
conditions are met:

1. Redistributions of source code must retain the above copyright
   notice, and the entire permission notice in its entirety,
   including the disclaimer of warranties.
2. Redistributions in binary form must reproduce the above copyright
   notice, this list of conditions and the following disclaimer in
   the documentation and/or other materials provided with the
   distribution.
3. The name of the author may not be used to endorse or promote
   products derived from this software without specific prior
   written permission.

ALTERNATIVELY, this product may be distributed under the terms
of the GNU Public License, in which case the provisions of the
GPL are required INSTEAD OF the above restrictions. (This clause
is necessary due to a potential bad interaction between the GPL
and the restrictions contained in a BSD-style copyright.)

THIS SOFTWARE IS PROVIDED ``AS IS'' AND ANY EXPRESS OR IMPLIED
WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES
OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
DISCLAIMED. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT,
INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
(INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED
OF THE POSSIBILITY OF SUCH DAMAGE.
```

A.26 LPeg Library License

The following software may be included in this product:

```

LPeg

Use of any of this software is governed by the terms of the license below:

Copyright © 2008 Lua.org, PUC-Rio.
```

```
Permission is hereby granted, free of charge, to any person obtaining a copy of
this software and associated documentation files (the "Software"), to deal in
the Software without restriction, including without limitation the rights to
use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of
the Software, and to permit persons to whom the Software is furnished to do so,
subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS
FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER
IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN
CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
```

A.27 Lua (liblua) License

The following software may be included in this product:

```
Lua (liblua)

Copyright © 1994–2008 Lua.org, PUC-Rio.

Permission is hereby granted, free of charge, to any person obtaining
a copy of this software and associated documentation files (the
"Software"), to deal in the Software without restriction, including
without limitation the rights to use, copy, modify, merge, publish,
distribute, sublicense, and/or sell copies of the Software, and to
permit persons to whom the Software is furnished to do so, subject
to the following conditions:

The above copyright notice and this permission notice shall be
included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES
OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,
WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
OTHER DEALINGS IN THE SOFTWARE.
```

A.28 `LuaFileSystem` Library License

The following software may be included in this product:

```
LuaFileSystem

Copyright © 2003 Kepler Project.

Permission is hereby granted, free of charge, to any person obtaining
a copy of this software and associated documentation files (the
"Software"), to deal in the Software without restriction, including
without limitation the rights to use, copy, modify, merge, publish,
distribute, sublicense, and/or sell copies of the Software, and to
permit persons to whom the Software is furnished to do so, subject
to the following conditions:

The above copyright notice and this permission notice shall be
included in all copies or substantial portions of the Software.
```

```
THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.
IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR
ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF
CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION
WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
```

A.29 md5 (Message-Digest Algorithm 5) License

The following software may be included in this product:

```
md5 (Message-Digest Algorithm 5)

 This code implements the MD5 message-digest algorithm.
 The algorithm is due to Ron Rivest.  This code was
 written by Colin Plumb in 1993, no copyright is claimed.
 This code is in the public domain; do with it what you wish.

 Equivalent code is available from RSA Data Security, Inc.
 This code has been tested against that, and is equivalent,
 except that you don't need to include two pages of legalese
 with every copy.

 The code has been modified by Mikael Ronstroem to handle
 calculating a hash value of a key that is always a multiple
 of 4 bytes long. Word 0 of the calculated 4-word hash value
 is returned as the hash value.
```

A.30 MeCab License

The following software may be included in this product:

```
Copyright (c) 2001-2008, Taku Kudo
Copyright (c) 2004-2008, Nippon Telegraph and Telephone Corporation
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:

 * Redistributions of source code must retain the above
   copyright notice, this list of conditions and the
   following disclaimer.

 * Redistributions in binary form must reproduce the above
   copyright notice, this list of conditions and the
   following disclaimer in the documentation and/or other
   materials provided with the distribution.

 * Neither the name of the Nippon Telegraph and Telegraph Corporation
   nor the names of its contributors may be used to endorse or
   promote products derived from this software without specific
   prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
```

A.31 memcached License

The following software may be included in this product:

memcached

A.32 Memcached.pm License

The following software may be included in this product:

Memcached.pm

```
of MySQL.

A copy of the GPLv2 and the Artistic License (Perl) 1.0 must be included with
any distribution.
```

This component is licensed under Section A.12, “GNU General Public License Version 2.0, June 1991”

This component is licensed under Section A.1, “Artistic License (Perl) 1.0”

A.33 mkpasswd.pl License

The following software may be included in this product:

mkpasswd.pl Perl module

```

Copyright (C) 2003-2004 by Chris Grau

This library is free software; you can redistribute it and/or modify it under
the same terms as Perl itself, either Perl version 5.8.1 or, at your option,
any later version of Perl 5 you may have available.

The Perl 5.8.1 license (from http://www.cpan.org/src/5.0/perl-5.8.1.tar.gz - main readme file):


    Perl Kit, Version 5

                Copyright (C) 1993, 1994, 1995, 1996, 1997, 1998
                   1999, 2000, 2001, by Larry Wall and others

    All rights reserved.

    This program is free software; you can redistribute it and/or modify
    it under the terms of either:

a) the GNU General Public License as published by the Free
Software Foundation; either version 1, or (at your option) any
later version, or

b) the "Artistic License" which comes with this Kit.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See either
    the GNU General Public License or the Artistic License for more details.

    You should have received a copy of the Artistic License with this
    Kit, in the file named "Artistic".  If not, I'll be glad to provide one.

    You should also have received a copy of the GNU General Public License
    along with this program in the file named "Copying". If not, write to the

    Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA
    02111-1307, USA or visit their web page on the internet at
    http://www.gnu.org/copyleft/gpl.html.

    For those of you that choose to use the GNU General Public License,
    my interpretation of the GNU General Public License is that no Perl
    script falls under the terms of the GPL unless you explicitly put
    said script under the terms of the GPL yourself.  Furthermore, any
    object code linked with perl does not automatically fall under the
    terms of the GPL, provided such object code only adds definitions
    of subroutines and variables, and does not otherwise impair the
    resulting interpreter from executing any standard Perl script.  I
    consider linking in C subroutines in this manner to be the moral
    equivalent of defining subroutines in the Perl language itself.  You
    may sell such an object file as proprietary provided that you provide
    or offer to provide the Perl source, as specified by the GNU General
    Public License.  (This is merely an alternate way of specifying input
    to the program.)  You may also sell a binary produced by the dumping of
    a running Perl script that belongs to you, provided that you provide or
    offer to provide the Perl source as specified by the GPL.  (The
    fact that a Perl interpreter and your code are in the same binary file
    is, in this case, a form of mere aggregation.)  This is my interpretation
    of the GPL.  If you still have concerns or difficulties understanding
    my intent, feel free to contact me.  Of course, the Artistic License
    spells all this out for your protection, so you may prefer to use that.

--------------------------------------------------------------------------

Perl is a language that combines some of the features of C, sed, awk
and shell.  See the manual page for more hype.  There are also many Perl
books available, covering a wide variety of topics, from various publishers.
See pod/perlbook.pod for more information.

Please read all the directions below before you proceed any further, and
then follow them carefully.

After you have unpacked your kit, you should have all the files listed
in MANIFEST.

Installation

1) Detailed instructions are in the file "INSTALL", which you should
read if you are either installing on a system resembling Unix
or porting perl to another platform.  For non-Unix platforms, see the
corresponding README.

2) Read the manual entries before running perl.

3) IMPORTANT!  Help save the world!  Communicate any problems and suggested
patches to perlbug@perl.org so we can keep the world in sync.
If you have a problem, there's someone else out there who either has had
or will have the same problem.  It's usually helpful if you send the
output of the "myconfig" script in the main perl directory.

If you've succeeded in compiling perl, the perlbug script in the "utils"
subdirectory can be used to help mail in a bug report.

If possible, send in patches such that the patch program will apply them.
Context diffs are the best, then normal diffs.  Don't send ed scripts--
I've probably changed my copy since the version you have.

The latest versions of perl are always available on the various CPAN
(Comprehensive Perl Archive Network) sites around the world.
See <URL:http://www.cpan.org/src/>.


Just a personal note:  I want you to know that I create nice things like this
because it pleases the Author of my story.  If this bothers you, then your
notion of Authorship needs some revision.  But you can use perl anyway. :-)

The author.
===============================================

The "Artistic License"

Preamble

The intent of this document is to state the conditions under which a
Package may be copied, such that the Copyright Holder maintains some
semblance of artistic control over the development of the package,
while giving the users of the package the right to use and distribute
```

```
the Package in a more-or-less customary fashion, plus the right to make
reasonable modifications.

Definitions:

"Package" refers to the collection of files distributed by the
Copyright Holder, and derivatives of that collection of files
created through textual modification.

"Standard Version" refers to such a Package if it has not been
modified, or has been modified in accordance with the wishes
of the Copyright Holder as specified below.

"Copyright Holder" is whoever is named in the copyright or
copyrights for the package.

"You" is you, if you're thinking about copying or distributing
this Package.

"Reasonable copying fee" is whatever you can justify on the
basis of media cost, duplication charges, time of people involved,
and so on.  (You will not be required to justify it to the
Copyright Holder, but only to the computing community at large
as a market that must bear the fee.)

"Freely Available" means that no fee is charged for the item
itself, though there may be fees involved in handling the item.
It also means that recipients of the item may redistribute it
under the same conditions they received it.

1. You may make and give away verbatim copies of the source form of the
Standard Version of this Package without restriction, provided that you
duplicate all of the original copyright notices and associated disclaimers.

2. You may apply bug fixes, portability fixes and other modifications
derived from the Public Domain or from the Copyright Holder.  A Package
modified in such a way shall still be considered the Standard Version.

3. You may otherwise modify your copy of this Package in any way, provided
that you insert a prominent notice in each changed file stating how and
when you changed that file, and provided that you do at least ONE of the
following:

    a) place your modifications in the Public Domain or otherwise make them
    Freely Available, such as by posting said modifications to Usenet or
    an equivalent medium, or placing the modifications on a major archive
    site such as uunet.uu.net, or by allowing the Copyright Holder to include
    your modifications in the Standard Version of the Package.

    b) use the modified Package only within your corporation or organization.

    c) rename any non-standard executables so the names do not conflict
    with standard executables, which must also be provided, and provide
    a separate manual page for each non-standard executable that clearly
    documents how it differs from the Standard Version.

    d) make other distribution arrangements with the Copyright Holder.

4. You may distribute the programs of this Package in object code or
executable form, provided that you do at least ONE of the following:

    a) distribute a Standard Version of the executables and library files,
    together with instructions (in the manual page or equivalent) on where
    to get the Standard Version.

    b) accompany the distribution with the machine-readable source of
    the Package with your modifications.
    c) give non-standard executables non-standard names, and clearly
    document the differences in manual pages (or equivalent), together
    with instructions on where to get the Standard Version.

    d) make other distribution arrangements with the Copyright Holder.

5. You may charge a reasonable copying fee for any distribution of this
Package.  You may charge any fee you choose for support of this
Package.  You may not charge a fee for this Package itself.  However,
you may distribute this Package in aggregate with other (possibly
commercial) programs as part of a larger (possibly commercial) software
distribution provided that you do not advertise this Package as a
product of your own.  You may embed this Package's interpreter within
an executable of yours (by linking); this shall be construed as a mere
form of aggregation, provided that the complete Standard Version of the
interpreter is so embedded.

6. The scripts and library files supplied as input to or produced as
output from the programs of this Package do not automatically fall
under the copyright of this Package, but belong to whoever generated
them, and may be sold commercially, and may be aggregated with this
Package.  If such scripts or library files are aggregated with this
Package via the so-called "undump" or "unexec" methods of producing a
binary executable image, then distribution of such an image shall
neither be construed as a distribution of this Package nor shall it
fall under the restrictions of Paragraphs 3 and 4, provided that you do
not represent such an executable image as a Standard Version of this
Package.

7. C subroutines (or comparably compiled subroutines in other
languages) supplied by you and linked into this Package in order to
emulate subroutines and variables of the language defined by this
Package shall not be considered part of this Package, but are the
equivalent of input as in Paragraph 6, provided these subroutines do
not change the language in any way that would cause it to fail the
regression tests for the language.

8. Aggregation of this Package with a commercial distribution is always
permitted provided that the use of this Package is embedded; that is,
when no overt attempt is made to make this Package's interfaces visible
to the end user of the commercial distribution.  Such use shall not be
construed as a distribution of this Package.

9. The name of the Copyright Holder may not be used to endorse or promote
products derived from this software without specific prior written
permission.

10. THIS PACKAGE IS PROVIDED "AS IS" AND WITHOUT ANY EXPRESS OR
IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED
WARRANTIES OF MERCHANTIBILITY AND FITNESS FOR A PARTICULAR PURPOSE.

The End
```

A.34 nt_servc (Windows NT Service class library) License

The following software may be included in this product:

nt_servc (Windows NT Service class library)

```
Windows NT Service class library
Copyright Abandoned 1998 Irena Pancirov - Irnet Snc
This file is public domain and comes with NO WARRANTY of any kind
```

A.35 OpenPAM License

The following software may be included in this product:

OpenPAM

```
Copyright (c) 2002-2003 Networks Associates Technology, Inc.
Copyright (c) 2004-2007 Dag-Erling Smørgrav
All rights reserved.

This software was developed for the FreeBSD Project by
ThinkSec AS and Network Associates Laboratories, the
Security Research Division of Network Associates, Inc.
under DARPA/SPAWAR contract N66001-01-C-8035 ("CBOSS"),
as part of the DARPA CHATS research program.

Redistribution and use in source and binary forms,
with or without modification, are permitted provided
that the following conditions are met:

1. Redistributions of source code must retain the above
   copyright notice, this list of conditions and the
   following disclaimer.
2. Redistributions in binary form must reproduce the
   above copyright notice, this list of conditions and
   the following disclaimer in the documentation and/or
   other materials provided with the distribution.
3. The name of the author may not be used to endorse or
   promote products derived from this software without
   specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY
AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED.  IN
NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE FOR
ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED
AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN
IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

A.36 OpenSSL v1.0 License

The following software may be included in this product:

```
NOTE: Does not apply to GPL licensed server (OpenSSL is not shipped with it)

OpenSSL v1.0

LICENSE ISSUES
==============
The OpenSSL toolkit stays under a dual license, i.e. both the conditions of
the OpenSSL License and the original SSLeay license apply to the toolkit. See
below for the actual license texts. Actually both licenses are BSD-style Open
Source licenses. In case of any license issues related to OpenSSL please
contact openssl-core@openssl.org.

OpenSSL License
---------------
```

```
/ ====================================================================
Copyright (c) 1998-2008 The OpenSSL Project.
All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:
.
1. Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice,
this list of conditions and the following disclaimer in the documentation
and/or other materials provided with the distribution.
3. All advertising materials mentioning features or use of this software must
display the following acknowledgment:  "This product includes software
developed by the OpenSSL Project for use in the OpenSSL Toolkit. (Link1 /)"
.
4. The names "OpenSSL Toolkit" and "OpenSSL Project" must not be used to
endorse or promote products derived from this software without prior written
permission. For written permission, please contact openssl-core@openssl.org.

5. Products derived from this software may not be called "OpenSSL" nor may
"OpenSSL" appear in their names without prior written permission of the
OpenSSL Project.
6. Redistributions of any form whatsoever must retain the following
acknowledgment:  "This product includes software developed by the OpenSSL
Project for use in the OpenSSL Toolkit (Link2 /)"
.
THIS SOFTWARE IS PROVIDED BY THE OpenSSL PROJECT ``AS IS'' AND ANY  EXPRESSED
OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE  IMPLIED WARRANTIES
OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR  PURPOSE ARE DISCLAIMED. IN
NO EVENT SHALL THE OpenSSL PROJECT OR  ITS CONTRIBUTORS BE LIABLE FOR ANY
DIRECT, INDIRECT, INCIDENTAL,  SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
(INCLUDING, BUT  NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
 LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)  HOWEVER CAUSED AND
ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,  STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE)  ARISING IN ANY WAY OUT OF THE USE OF
THIS SOFTWARE, EVEN IF ADVISED  OF THE POSSIBILITY OF SUCH DAMAGE.
====================================================================
This product includes cryptographic software written by Eric Young
(eay@cryptsoft.com). This product includes software written by Tim Hudson
(tjh@cryptsoft.com).

Original SSLeay License
-----------------------
/ Copyright (C) 1995-1998 Eric Young (eay@cryptsoft.com)
All rights reserved.
This package is an SSL implementation written by Eric Young
(eay@cryptsoft.com).  The implementation was written so as to conform with
Netscapes SSL.   This library is free for commercial and non-commercial use
as long as the following conditions are aheared to. The following conditions
apply to all code found in this distribution, be it the RC4, RSA,  lhash,
DES, etc., code; not just the SSL code. The SSL documentation included with
this distribution is covered by the same copyright terms except that the
holder is Tim Hudson (tjh@cryptsoft.com).   Copyright remains Eric Young's,
and as such any Copyright notices in the code are not to be removed.  If this
package is used in a product, Eric Young should be given attribution as the
author of the parts of the library used.  This can be in the form of a
textual message at program startup or in documentation (online or textual)
provided with the package.   Redistribution and use in source and binary
forms, with or without modification, are permitted provided that the
following conditions are met:  1. Redistributions of source code must retain
the copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice,
this list of conditions and the following disclaimer in the documentation
and/or other materials provided with the distribution.  3. All advertising
materials mentioning features or use of this software must display the
following acknowledgement:  "This product includes cryptographic software
written by Eric Young (eay@cryptsoft.com)" The word 'cryptographic' can be
```

```
left out if the routines from the library being used are not cryptographic
related :-).  4. If you include any Windows specific code (or a derivative
thereof) from the apps directory (application code) you must include an
acknowledgement:  "This product includes software written by Tim Hudson
(tjh@cryptsoft.com)"   THIS SOFTWARE IS PROVIDED BY ERIC YOUNG ``AS IS'' AND
ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE  ARE
DISCLAIMED. IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE  FOR ANY
DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL  DAMAGES
(INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS  OR SERVICES;
LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)  HOWEVER CAUSED AND
ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT  LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY  OUT OF THE USE OF
THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF  SUCH DAMAGE.   The
license and distribution terms for any publically available version or
derivative of this code cannot be changed. i.e. this code cannot simply be
copied and put under another distribution license  [including the GNU Public
License.]
```

A.37 PCRE License

The following software may be included in this product:

PCRE (Perl Compatible Regular Expressions) Library

```

PCRE LICENCE

PCRE is a library of functions to support regular expressions
whose syntax and semantics are as close as possible to those
of the Perl 5 language.

Release 7 of PCRE is distributed under the terms of the "BSD"
licence, as specified below. The documentation for PCRE,
supplied in the "doc" directory, is distributed under the same
terms as the software itself.

The basic library functions are written in C and are
freestanding. Also included in the distribution is a set
of C++ wrapper functions.

THE BASIC LIBRARY FUNCTIONS
---------------------------
Written by:       Philip Hazel
Email local part: ph10
Email domain:     cam.ac.uk

University of Cambridge Computing Service,
Cambridge, England. Phone: +44 1223 334714.

Copyright (c) 1997-2006 University of Cambridge
All rights reserved.

THE C++ WRAPPER FUNCTIONS
-------------------------
Contributed by:   Google Inc.

Copyright (c) 2006, Google Inc.
All rights reserved.

THE "BSD" LICENCE
-----------------

Redistribution and use in source and binary forms,
with or without modification, are permitted provided
that the following conditions are met:
```

```
* Redistributions of source code must retain the above
  copyright notice, this list of conditions and the
  following disclaimer.
* Redistributions in binary form must reproduce the
  above copyright notice, this list of conditions and
  the following disclaimer in the documentation and/or
  other materials provided with the distribution.
* Neither the name of the University of Cambridge nor
  the name of Google Inc. nor the names of their contributors
  may be used to endorse or promote products derived from
  this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES,
INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS
BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY,
OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT
OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS;
OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF
THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

End
```

A.38 Percona Multiple I/O Threads Patch License

The following software may be included in this product:

Percona Multiple I/O threads patch

```
Copyright (c) 2008, 2009 Percona Inc
All rights reserved.

Redistribution and use of this software in source and binary forms,
with or without modification, are permitted provided that the
following conditions are met:

 * Redistributions of source code must retain the above copyright
   notice, this list of conditions and the following disclaimer.
 * Redistributions in binary form must reproduce the above copyright
   notice, this list of conditions and the following disclaimer in the
   documentation and/or other materials provided with the distribution.
 * Neither the name of Percona Inc. nor the names of its contributors
   may be used to endorse or promote products derived from this software
   without specific prior written permission of Percona Inc.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN
ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
POSSIBILITY OF SUCH DAMAGE.
```

A.39 Red HAT RPM Spec File License

The following software may be included in this product:

Red Hat RPM Spec File

```
You are receiving a copy of the Red Hat spec file. The terms of the Oracle
license do NOT apply to the Red Hat spec file; it is licensed under the
following license, separately from the Oracle programs you receive.

                    GNU GENERAL PUBLIC LICENSE
                       Version 2, June 1991

Copyright (C) 1989, 1991 Free Software Foundation, Inc.,
51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA
Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.
[for rest of text, see following link]
```

This component is licensed under Section A.12, “GNU General Public License Version 2.0, June 1991”

A.40 RegEX-Spencer Library License

The following software may be included in this product: Henry Spencer's Regular-Expression Library (RegEX-Spencer)

```
Copyright 1992, 1993, 1994 Henry Spencer.  All rights reserved.
This software is not subject to any license of the American Telephone
and Telegraph Company or of the Regents of the University of California.

Permission is granted to anyone to use this software for any purpose on
any computer system, and to alter it and redistribute it, subject
to the following restrictions:

1. The author is not responsible for the consequences of use of this
   software, no matter how awful, even if they arise from flaws in it.

2. The origin of this software must not be misrepresented, either by
   explicit claim or by omission.  Since few users ever read sources,
   credits must appear in the documentation.

3. Altered versions must be plainly marked as such, and must not be
   misrepresented as being the original software.  Since few users
   ever read sources, credits must appear in the documentation.

4. This notice may not be removed or altered.
```

A.41 Richard A. O'Keefe String Library License

The following software may be included in this product:

Richard A. O'Keefe String Library

```
The Richard O’Keefe String Library is subject to the following notice:

These files are in the public domain.  This includes getopt.c, which
is the work of Henry Spencer, University of Toronto Zoology, who
says of it "None of this software is derived from Bell software. I
had no access to the source for Bell's versions at the time I wrote
it.  This software is hereby explicitly placed in the public domain.
```

```
It may be used for any purpose on any machine by anyone." I would
greatly prefer it if *my* material received no military use.

The t_ctype.h file is subject to the following notice:

Copyright (C) 1998, 1999 by Pruet Boonma, all rights reserved.
Copyright (C) 1998 by Theppitak Karoonboonyanan, all rights reserved.

  Permission to use, copy, modify, distribute and sell this software and its
documentation for any purpose is hereby granted without fee, provided that the above
copyright notice appear in all copies.

  Smaphan Raruenrom and Pruet Boonma makes no representations about
the suitability of this software for any purpose. It is provided
"as is" without express or implied warranty.
```

A.42 SHA-1 in C License

The following software may be included in this product:

SHA-1 in C

```

SHA-1 in C
By Steve Reid <steve@edmweb.com>
100% Public Domain
```

A.43 Unicode Data Files

The following software may be included in this product:

Unicode Data Files

```

Copyright 2001-2009 Unicode, Inc.

 Disclaimer

This source code is provided as is by Unicode, Inc. No claims are
made as to fitness for any particular purpose. No warranties of any
kind are expressed or implied. The recipient agrees to determine
applicability of information provided. If this file has been
purchased on magnetic or optical media from Unicode, Inc., the
sole remedy for any claim will be exchange of defective media
within 90 days of receipt.

 Limitations on Rights to Redistribute This Code

Unicode, Inc. hereby grants the right to freely use the information
supplied in this file in the creation of products supporting the
Unicode Standard, and to make copies of this file in any form
for internal or external distribution as long as this notice
remains attached.
```

A.44 `zlib` License

The following software may be included in this product:

zlib

Oracle gratefully acknowledges the contributions of Jean-loup Gailly and Mark Adler in creating the zlib general purpose compression library which is used in this product.

```
zlib.h -- interface of the 'zlib' general purpose compression library
Copyright (C) 1995-2004 Jean-loup Gailly and Mark Adler

zlib.h -- interface of the 'zlib' general purpose compression library
version 1.2.3, July 18th, 2005
Copyright (C) 1995-2005 Jean-loup Gailly and Mark Adler

zlib.h -- interface of the 'zlib' general purpose compression library
version 1.2.5, April 19th, 2010
Copyright (C) 1995-2010 Jean-loup Gailly and Mark Adler

This software is provided 'as-is', without any express or implied warranty.
In no event will the authors be held liable for any damages arising from the
use of this software. Permission is granted to anyone to use this software
for any purpose,including commercial applications, and to alter it and
redistribute it freely, subject to the following restrictions:

1. The origin of this software must not be misrepresented; you must not
   claim that you wrote the original software. If you use this software
   in a product, an acknowledgment in the product documentation would
   be appreciated but is not required.
2. Altered source versions must be plainly marked as such, and must not
   be misrepresented as being the original software.
3. This notice may not be removed or altered from any source distribution.

Jean-loup Gailly jloup@gzip.org
Mark Adler madler@alumni.caltech.edu
```

Appendix B MySQL 5.7 Frequently Asked Questions

Table of Contents

B.1 MySQL 5.7 FAQ: General

Questions

- B.1.1: [2913] Which version of MySQL is production-ready (GA)?
- B.1.2: [2914] What is the state of development (non-GA) versions?
- B.1.3: [2914] Can MySQL 5.7 do subqueries?
- B.1.4: [2914] Can MySQL 5.7 perform multiple-table inserts, updates, and deletes?
- B.1.5: [2914] Does MySQL 5.7 have a Query Cache? Does it work on Server, Instance or Database?
- B.1.6: [2914] Does MySQL 5.7 have Sequences?
- B.1.7: [2914] Does MySQL 5.7 have a `NOW()` function with fractions of seconds?
- B.1.8: [2914] Does MySQL 5.7 work with multi-core processors?
- B.1.9: [2915] Why do I see multiple processes for `mysqld`?
- B.1.10: [2915] Have there been there any improvements in error reporting when foreign keys fail? Does MySQL now report which column and reference failed?
- B.1.11: [2915] Can MySQL 5.7 perform ACID transactions?

Questions and Answers

B.1.1: Which version of MySQL is production-ready (GA)?

MySQL 5.6, MySQL 5.5, MySQL 5.1, and MySQL 5.0 are supported for production use.

MySQL 5.6 achieved General Availability (GA) status with MySQL 5.6.10, which was released for production use on 5 February 2013.

MySQL 5.5 achieved General Availability (GA) status with MySQL 5.5.8, which was released for production use on 3 December 2010.

MySQL 5.1 achieved General Availability (GA) status with MySQL 5.1.30, which was released for production use on 14 November 2008.

MySQL 5.0 achieved General Availability (GA) status with MySQL 5.0.15, which was released for production use on 19 October 2005. Note that active development for MySQL 5.0 has ended.

B.1.2: What is the state of development (non-GA) versions?

MySQL follows a milestone release model that introduces pre-production-quality features and stabilizes them to release quality (see http://dev.mysql.com/doc/mysql-development-cycle/en/index.html). This process then repeats, so releases cycle between pre-production and release quality status. Please check the change logs to identify the status of a given release.

MySQL 5.4 was a development series. Work on this series has ceased.

MySQL 5.7 is being actively developed using the milestone release methodology described above.

MySQL 6.0 was a development series. Work on this series has ceased.

B.1.3: Can MySQL 5.7 do subqueries?

Yes. See Section 13.2.10, “Subquery Syntax”.

B.1.4: Can MySQL 5.7 perform multiple-table inserts, updates, and deletes?

Yes. For the syntax required to perform multiple-table updates, see Section 13.2.11, “`UPDATE` Syntax”; for that required to perform multiple-table deletes, see Section 13.2.2, “`DELETE` Syntax”.

A multiple-table insert can be accomplished using a trigger whose `FOR EACH ROW` clause contains multiple `INSERT` statements within a `BEGIN ... END` block. See Section 18.3, “Using Triggers”.

B.1.5: Does MySQL 5.7 have a Query Cache? Does it work on Server, Instance or Database?

Yes. The query cache operates on the server level, caching complete result sets matched with the original query string. If an exactly identical query is made (which often happens, particularly in web applications), no parsing or execution is necessary; the result is sent directly from the cache. Various tuning options are available. See Section 8.9.3, “The MySQL Query Cache”.

B.1.6: Does MySQL 5.7 have Sequences?

No. However, MySQL has an `AUTO_INCREMENT` system, which in MySQL 5.7 can also handle inserts in a multi-master replication setup. With the `auto_increment_increment` and `auto_increment_offset` system variables, you can set each server to generate auto-increment values that don't conflict with other servers. The `auto_increment_increment` value should be greater than the number of servers, and each server should have a unique offset.

B.1.7: Does MySQL 5.7 have a `NOW()` function with fractions of seconds?

No. This is on the MySQL roadmap as a “rolling feature”. This means that it is not a flagship feature, but will be implemented, development time permitting. Specific customer demand may change this scheduling.

However, MySQL does parse time strings with a fractional component. See Section 11.3.2, “The `TIME` Type”.

B.1.8: Does MySQL 5.7 work with multi-core processors?

Yes. MySQL is fully multi-threaded, and will make use of multiple CPUs, provided that the operating system supports them.

B.1.9: Why do I see multiple processes for `mysqld`?

When using LinuxThreads, you should see a minimum of three `mysqld` processes running. These are in fact threads. There is one thread for the LinuxThreads manager, one thread to handle connections, and one thread to handle alarms and signals.

B.1.10: Have there been there any improvements in error reporting when foreign keys fail? Does MySQL now report which column and reference failed?

The foreign key support in `InnoDB` has seen improvements in each major version of MySQL. Foreign key support generic to all storage engines is scheduled for MySQL 6.x; this should resolve any inadequacies in the current storage engine specific implementation.

B.1.11: Can MySQL 5.7 perform ACID transactions?

Yes. All current MySQL versions support transactions. The `InnoDB` storage engine offers full ACID transactions with row-level locking, multi-versioning, nonlocking repeatable reads, and all four SQL standard isolation levels.

The `NDB` storage engine supports the `READ COMMITTED` transaction isolation level only.

B.2 MySQL 5.7 FAQ: Storage Engines

Questions

- B.2.1: [2915] Where can I obtain complete documentation for MySQL storage engines?
- B.2.2: [2915] Are there any new storage engines in MySQL 5.7?
- B.2.3: [2915] Have any storage engines been removed in MySQL 5.7?
- B.2.4: [2916] What are the unique benefits of the `ARCHIVE` storage engine?
- B.2.5: [2916] Do the new features in MySQL 5.7 apply to all storage engines?

Questions and Answers

B.2.1: Where can I obtain complete documentation for MySQL storage engines?

See Chapter 14, *Storage Engines*. That chapter contains information about all MySQL storage engines except for the `NDB` storage engine used for MySQL Cluster; `NDB` is covered in MySQL Cluster NDB 7.3.

B.2.2: Are there any new storage engines in MySQL 5.7?

The features from the optional `InnoDB Plugin` from MySQL 5.1 are folded into the built-in `InnoDB` storage engine, so you can take advantage of features such as the Barracuda file format, `InnoDB` table compression, and the new configuration options for performance. See Section 14.2, “The `InnoDB` Storage Engine” for details. `InnoDB` also becomes the default storage engine for new tables. See Section 14.2.1.1, “`InnoDB` as the Default MySQL Storage Engine” for details.

B.2.3: Have any storage engines been removed in MySQL 5.7?

No.

B.2.4: What are the unique benefits of the ARCHIVE storage engine?

The ARCHIVE storage engine is ideally suited for storing large amounts of data without indexes; it has a very small footprint, and performs selects using table scans. See Section 14.6, “The ARCHIVE Storage Engine”, for details.

B.2.5: Do the new features in MySQL 5.7 apply to all storage engines?

The general new features such as views, stored procedures, triggers, INFORMATION_SCHEMA, precision math (DECIMAL column type), and the BIT column type, apply to all storage engines. There are also additions and changes for specific storage engines.

B.3 MySQL 5.7 FAQ: Server SQL Mode

Questions

Questions and Answers

B.3.1: What are server SQL modes?

Server SQL modes define what SQL syntax MySQL should support and what kind of data validation checks it should perform. This makes it easier to use MySQL in different environments and to use MySQL together with other database servers. The MySQL Server apply these modes individually to different clients. For more information, see Section 5.1.7, “Server SQL Modes”.

B.3.2: How many server SQL modes are there?

Each mode can be independently switched on and off. See Section 5.1.7, “Server SQL Modes”, for a complete list of available modes.

B.3.3: How do you determine the server SQL mode?

You can set the default SQL mode (for mysqld startup) with the --sql-mode option. Using the statement SET [GLOBAL|SESSION] sql_mode='*modes*', you can change the settings from within a connection, either locally to the connection, or to take effect globally. You can retrieve the current mode by issuing a SELECT @@sql_mode statement.

B.3.4: Is the mode dependent on the database or connection?

A mode is not linked to a particular database. Modes can be set locally to the session (connection), or globally for the server. you can change these settings using SET [GLOBAL|SESSION] sql_mode='*modes*'.

B.3.5: Can the rules for strict mode be extended?

When we refer to *strict mode*, we mean a mode where at least one of the modes TRADITIONAL, STRICT_TRANS_TABLES, or STRICT_ALL_TABLES is enabled. Options can be combined, so you can add restrictions to a mode. See Section 5.1.7, “Server SQL Modes”, for more information.

B.3.6: Does strict mode impact performance?

The intensive validation of input data that some settings requires more time than if the validation is not done. While the performance impact is not that great, if you do not require such validation (perhaps your application already handles all of this), then MySQL gives you the option of leaving strict mode disabled. However—if you do require it—strict mode can provide such validation.

B.3.7: What is the default server SQL mode when MySQL 5.7 is installed?

The default SQL mode is NO_ENGINE_SUBSTITUTION. See Section 5.1.7, “Server SQL Modes”, for information about all available modes and MySQL's default behavior.

B.4 MySQL 5.7 FAQ: Stored Procedures and Functions

Questions

- B.4.1: [2918] Does MySQL 5.7 support stored procedures and functions?
- B.4.2: [2918] Where can I find documentation for MySQL stored procedures and stored functions?
- B.4.3: [2918] Is there a discussion forum for MySQL stored procedures?
- B.4.4: [2918] Where can I find the ANSI SQL 2003 specification for stored procedures?
- B.4.5: [2918] How do you manage stored routines?
- B.4.6: [2918] Is there a way to view all stored procedures and stored functions in a given database?
- B.4.7: [2919] Where are stored procedures stored?
- B.4.8: [2919] Is it possible to group stored procedures or stored functions into packages?
- B.4.9: [2919] Can a stored procedure call another stored procedure?
- B.4.10: [2919] Can a stored procedure call a trigger?
- B.4.11: [2919] Can a stored procedure access tables?
- B.4.12: [2919] Do stored procedures have a statement for raising application errors?
- B.4.13: [2919] Do stored procedures provide exception handling?
- B.4.14: [2919] Can MySQL 5.7 stored routines return result sets?
- B.4.15: [2919] Is WITH RECOMPILE supported for stored procedures?
- B.4.16: [2919] Is there a MySQL equivalent to using mod_plsql as a gateway on Apache to talk directly to a stored procedure in the database?
- B.4.17: [2919] Can I pass an array as input to a stored procedure?
- B.4.18: [2919] Can I pass a cursor as an IN parameter to a stored procedure?
- B.4.19: [2920] Can I return a cursor as an OUT parameter from a stored procedure?
- B.4.20: [2920] Can I print out a variable's value within a stored routine for debugging purposes?

Questions and Answers

B.4.1: Does MySQL 5.7 support stored procedures and functions?

Yes. MySQL 5.7 supports two types of stored routines—stored procedures and stored functions.

B.4.2: Where can I find documentation for MySQL stored procedures and stored functions?

See Section 18.2, “Using Stored Routines (Procedures and Functions)”.

B.4.3: Is there a discussion forum for MySQL stored procedures?

Yes. See http://forums.mysql.com/list.php?98.

B.4.4: Where can I find the ANSI SQL 2003 specification for stored procedures?

Unfortunately, the official specifications are not freely available (ANSI makes them available for purchase). However, there are books—such as *SQL-99 Complete, Really* by Peter Gulutzan and Trudy Pelzer—which give a comprehensive overview of the standard, including coverage of stored procedures.

B.4.5: How do you manage stored routines?

It is always good practice to use a clear naming scheme for your stored routines. You can manage stored procedures with `CREATE [FUNCTION|PROCEDURE]`, `ALTER [FUNCTION|PROCEDURE]`, `DROP [FUNCTION|PROCEDURE]`, and `SHOW CREATE [FUNCTION|PROCEDURE]`. You can obtain information about existing stored procedures using the `ROUTINES` table in the `INFORMATION_SCHEMA` database (see Section 19.19, “The `INFORMATION_SCHEMA ROUTINES` Table”).

B.4.6: Is there a way to view all stored procedures and stored functions in a given database?

Yes. For a database named *`dbname`*, use this query on the `INFORMATION_SCHEMA.ROUTINES` table:

```
SELECT ROUTINE_TYPE, ROUTINE_NAME
    FROM INFORMATION_SCHEMA.ROUTINES
    WHERE ROUTINE_SCHEMA='dbname';
```

For more information, see Section 19.19, “The `INFORMATION_SCHEMA ROUTINES` Table”.

The body of a stored routine can be viewed using `SHOW CREATE FUNCTION` (for a stored function) or `SHOW CREATE PROCEDURE` (for a stored procedure). See Section 13.7.5.9, “`SHOW CREATE PROCEDURE` Syntax”, for more information.

B.4.7: Where are stored procedures stored?

In the `proc` table of the `mysql` system database. However, you should not access the tables in the system database directly. Instead, use `SHOW CREATE FUNCTION` to obtain information about stored functions, and `SHOW CREATE PROCEDURE` to obtain information about stored procedures. See Section 13.7.5.9, “`SHOW CREATE PROCEDURE` Syntax”, for more information about these statements.

You can also query the `ROUTINES` table in the `INFORMATION_SCHEMA` database—see Section 19.19, “The `INFORMATION_SCHEMA ROUTINES` Table”, for information about this table.

B.4.8: Is it possible to group stored procedures or stored functions into packages?

No. This is not supported in MySQL 5.7.

B.4.9: Can a stored procedure call another stored procedure?

Yes.

B.4.10: Can a stored procedure call a trigger?

A stored procedure can execute an SQL statement, such as an `UPDATE`, that causes a trigger to activate.

B.4.11: Can a stored procedure access tables?

Yes. A stored procedure can access one or more tables as required.

B.4.12: Do stored procedures have a statement for raising application errors?

Yes. MySQL 5.7 implements the SQL standard `SIGNAL` and `RESIGNAL` statements. See Section 13.6.7, “Condition Handling”.

B.4.13: Do stored procedures provide exception handling?

MySQL implements `HANDLER` definitions according to the SQL standard. See Section 13.6.7.2, “`DECLARE ... HANDLER` Syntax”, for details.

B.4.14: Can MySQL 5.7 stored routines return result sets?

Stored procedures can, but stored functions cannot. If you perform an ordinary `SELECT` inside a stored procedure, the result set is returned directly to the client. You need to use the MySQL 4.1 (or above) client/server protocol for this to work. This means that—for instance—in PHP, you need to use the `mysqli` extension rather than the old `mysql` extension.

B.4.15: Is `WITH RECOMPILE` supported for stored procedures?

Not in MySQL 5.7.

B.4.16: Is there a MySQL equivalent to using `mod_plsql` as a gateway on Apache to talk directly to a stored procedure in the database?

There is no equivalent in MySQL 5.7.

B.4.17: Can I pass an array as input to a stored procedure?

Not in MySQL 5.7.

B.4.18: Can I pass a cursor as an `IN` parameter to a stored procedure?

In MySQL 5.7, cursors are available inside stored procedures only.

B.4.19: Can I return a cursor as an `OUT` parameter from a stored procedure?

In MySQL 5.7, cursors are available inside stored procedures only. However, if you do not open a cursor on a `SELECT`, the result will be sent directly to the client. You can also `SELECT INTO` variables. See Section 13.2.9, “`SELECT` Syntax”.

B.4.20: Can I print out a variable's value within a stored routine for debugging purposes?

Yes, you can do this in a *stored procedure*, but not in a stored function. If you perform an ordinary `SELECT` inside a stored procedure, the result set is returned directly to the client. You will need to use the MySQL 4.1 (or above) client/server protocol for this to work. This means that—for instance—in PHP, you need to use the `mysqli` extension rather than the old `mysql` extension.

B.4.21: Can I commit or roll back transactions inside a stored procedure?

Yes. However, you cannot perform transactional operations within a stored function.

B.4.22: Do MySQL 5.7 stored procedures and functions work with replication?

Yes, standard actions carried out in stored procedures and functions are replicated from a master MySQL server to a slave server. There are a few limitations that are described in detail in Section 18.7, “Binary Logging of Stored Programs”.

B.4.23: Are stored procedures and functions created on a master server replicated to a slave?

Yes, creation of stored procedures and functions carried out through normal DDL statements on a master server are replicated to a slave, so the objects will exist on both servers. `ALTER` and `DROP` statements for stored procedures and functions are also replicated.

B.4.24: How are actions that take place inside stored procedures and functions replicated?

MySQL records each DML event that occurs in a stored procedure and replicates those individual actions to a slave server. The actual calls made to execute stored procedures are not replicated.

Stored functions that change data are logged as function invocations, not as the DML events that occur inside each function.

B.4.25: Are there special security requirements for using stored procedures and functions together with replication?

Yes. Because a slave server has authority to execute any statement read from a master's binary log, special security constraints exist for using stored functions with replication. If replication or binary logging in general (for the purpose of point-in-time recovery) is active, then MySQL DBAs have two security options open to them:

1. Any user wishing to create stored functions must be granted the `SUPER` privilege.

2. Alternatively, a DBA can set the `log_bin_trust_function_creators` system variable to 1, which enables anyone with the standard `CREATE ROUTINE` privilege to create stored functions.

B.4.26: What limitations exist for replicating stored procedure and function actions?

Nondeterministic (random) or time-based actions embedded in stored procedures may not replicate properly. By their very nature, randomly produced results are not predictable and cannot be exactly reproduced, and therefore, random actions replicated to a slave will not mirror those performed on a master. Note that declaring stored functions to be `DETERMINISTIC` or setting the log_bin_trust_function_creators system variable to 0 will not allow random-valued operations to be invoked.

In addition, time-based actions cannot be reproduced on a slave because the timing of such actions in a stored procedure is not reproducible through the binary log used for replication. It records only DML events and does not factor in timing constraints.

Finally, nontransactional tables for which errors occur during large DML actions (such as bulk inserts) may experience replication issues in that a master may be partially updated from DML activity, but no updates are done to the slave because of the errors that occurred. A workaround is for a function's DML actions to be carried out with the IGNORE keyword so that updates on the master that cause errors are ignored and updates that do not cause errors are replicated to the slave.

B.4.27: Do the preceding limitations affect MySQL's ability to do point-in-time recovery?

The same limitations that affect replication do affect point-in-time recovery.

B.4.28: What is being done to correct the aforementioned limitations?

You can choose either statement-based replication or row-based replication. The original replication implementation is based on statement-based binary logging. Row-based binary logging resolves the limitations mentioned earlier.

Mixed replication is also available (by starting the server with --binlog-format=mixed). This hybrid, “smart” form of replication “knows” whether statement-level replication can safely be used, or row-level replication is required.

For additional information, see Section 16.1.2, “Replication Formats”.

B.5 MySQL 5.7 FAQ: Triggers

Questions

- B.5.14: [2923] How are actions carried out through triggers on a master replicated to a slave?

Questions and Answers

B.5.1: Where can I find the documentation for MySQL 5.7 triggers?

See Section 18.3, "Using Triggers".

B.5.2: Is there a discussion forum for MySQL Triggers?

Yes. It is available at http://forums.mysql.com/list.php?99.

B.5.3: Does MySQL 5.7 have statement-level or row-level triggers?

In MySQL 5.7, all triggers are FOR EACH ROW—that is, the trigger is activated for each row that is inserted, updated, or deleted. MySQL 5.7 does not support triggers using FOR EACH STATEMENT.

B.5.4: Are there any default triggers?

Not explicitly. MySQL does have specific special behavior for some TIMESTAMP columns, as well as for columns which are defined using AUTO_INCREMENT.

B.5.5: How are triggers managed in MySQL?

In MySQL 5.7, triggers can be created using the CREATE TRIGGER statement, and dropped using DROP TRIGGER. See Section 13.1.15, "CREATE TRIGGER Syntax", and Section 13.1.24, "DROP TRIGGER Syntax", for more about these statements.

Information about triggers can be obtained by querying the INFORMATION_SCHEMA.TRIGGERS table. See Section 19.27, "The INFORMATION_SCHEMA TRIGGERS Table".

B.5.6: Is there a way to view all triggers in a given database?

Yes. You can obtain a listing of all triggers defined on database dbname using a query on the INFORMATION_SCHEMA.TRIGGERS table such as the one shown here:

```
SELECT TRIGGER_NAME, EVENT_MANIPULATION, EVENT_OBJECT_TABLE, ACTION_STATEMENT
    FROM INFORMATION_SCHEMA.TRIGGERS
    WHERE TRIGGER_SCHEMA='dbname';
```

For more information about this table, see Section 19.27, "The INFORMATION_SCHEMA TRIGGERS Table".

You can also use the SHOW TRIGGERS statement, which is specific to MySQL. See Section 13.7.5.37, "SHOW TRIGGERS Syntax".

B.5.7: Where are triggers stored?

Triggers for a table are currently stored in .TRG files, with one such file one per table.

B.5.8: Can a trigger call a stored procedure?

Yes.

B.5.9: Can triggers access tables?

A trigger can access both old and new data in its own table. A trigger can also affect other tables, but it is not permitted to modify a table that is already being used (for reading or writing) by the statement that invoked the function or trigger.

B.5.10: Can a table have multiple triggers with the same trigger event and action time?

As of MySQL 5.7.2, it is possible to define multiple triggers for a given table that have the same trigger event and action time. For example, you can have two `BEFORE UPDATE` triggers for a table. By default, triggers that have the same trigger event and action time activate in the order they were created. To affect trigger order, specify a clause after `FOR EACH ROW` that indicates `FOLLOWS` or `PRECEDES` and the name of an existing trigger that also has the same trigger event and action time. With `FOLLOWS`, the new trigger activates after the existing trigger. With `PRECEDES`, the new trigger activates before the existing trigger.

B.5.11: Can triggers call an external application through a UDF?

Yes. For example, a trigger could invoke the `sys_exec()` UDF.

B.5.12: Is it possible for a trigger to update tables on a remote server?

Yes. A table on a remote server could be updated using the `FEDERATED` storage engine. (See Section 14.9, “The `FEDERATED` Storage Engine”).

B.5.13: Do triggers work with replication?

Yes. However, the way in which they work depends whether you are using MySQL's “classic” statement-based replication available in all versions of MySQL, or the row-based replication format introduced in MySQL 5.1.

When using statement-based replication, triggers on the slave are executed by statements that are executed on the master (and replicated to the slave).

When using row-based replication, triggers are not executed on the slave due to statements that were run on the master and then replicated to the slave. Instead, when using row-based replication, the changes caused by executing the trigger on the master are applied on the slave.

For more information, see Section 16.4.1.32, “Replication and Triggers”.

B.5.14: How are actions carried out through triggers on a master replicated to a slave?

Again, this depends on whether you are using statement-based or row-based replication.

Statement-based replication. First, the triggers that exist on a master must be re-created on the slave server. Once this is done, the replication flow works as any other standard DML statement that participates in replication. For example, consider a table `EMP` that has an `AFTER` insert trigger, which exists on a master MySQL server. The same `EMP` table and `AFTER` insert trigger exist on the slave server as well. The replication flow would be:

1. An `INSERT` statement is made to `EMP`.
2. The `AFTER` trigger on `EMP` activates.
3. The `INSERT` statement is written to the binary log.
4. The replication slave picks up the `INSERT` statement to `EMP` and executes it.
5. The `AFTER` trigger on `EMP` that exists on the slave activates.

Row-based replication. When you use row-based replication, the changes caused by executing the trigger on the master are applied on the slave. However, the triggers themselves are not actually executed on the slave under row-based replication. This is because, if both the master and the slave applied the changes from the master and—in addition—the trigger causing these changes were applied on the slave, the changes would in effect be applied twice on the slave, leading to different data on the master and the slave.

In most cases, the outcome is the same for both row-based and statement-based replication. However, if you use different triggers on the master and slave, you cannot use row-based replication. (This is because the row-based format replicates the changes made by triggers executing on the master to the slaves, rather than the statements that caused the triggers to execute, and the corresponding triggers on the slave are not executed.) Instead, any statements causing such triggers to be executed must be replicated using statement-based replication.

For more information, see Section 16.4.1.32, “Replication and Triggers”.

B.6 MySQL 5.7 FAQ: Views

Questions

- B.6.1: [2924] Where can I find documentation covering MySQL Views?
- B.6.2: [2924] Is there a discussion forum for MySQL Views?
- B.6.3: [2924] What happens to a view if an underlying table is dropped or renamed?
- B.6.4: [2924] Does MySQL 5.7 have table snapshots?
- B.6.5: [2924] Does MySQL 5.7 have materialized views?
- B.6.6: [2924] Can you insert into views that are based on joins?

Questions and Answers

B.6.1: Where can I find documentation covering MySQL Views?

See Section 18.5, “Using Views”.

B.6.2: Is there a discussion forum for MySQL Views?

Yes. See http://forums.mysql.com/list.php?100

B.6.3: What happens to a view if an underlying table is dropped or renamed?

After a view has been created, it is possible to drop or alter a table or view to which the definition refers. To check a view definition for problems of this kind, use the `CHECK TABLE` statement. (See Section 13.7.2.2, “`CHECK TABLE` Syntax”.)

B.6.4: Does MySQL 5.7 have table snapshots?

No.

B.6.5: Does MySQL 5.7 have materialized views?

No.

B.6.6: Can you insert into views that are based on joins?

It is possible, provided that your `INSERT` statement has a column list that makes it clear there is only one table involved.

You *cannot* insert into multiple tables with a single insert on a view.

B.7 MySQL 5.7 FAQ: INFORMATION_SCHEMA

Questions

- B.7.1: [2925] Where can I find documentation for the MySQL INFORMATION_SCHEMA database?
- B.7.2: [2925] Is there a discussion forum for INFORMATION_SCHEMA?
- B.7.3: [2925] Where can I find the ANSI SQL 2003 specification for INFORMATION_SCHEMA?
- B.7.4: [2925] What is the difference between the Oracle Data Dictionary and MySQL's INFORMATION_SCHEMA?
- B.7.5: [2925] Can I add to or otherwise modify the tables found in the INFORMATION_SCHEMA database?

Questions and Answers

B.7.1: Where can I find documentation for the MySQL INFORMATION_SCHEMA database?

See Chapter 19, *INFORMATION_SCHEMA Tables*

B.7.2: Is there a discussion forum for INFORMATION_SCHEMA?

See http://forums.mysql.com/list.php?101.

B.7.3: Where can I find the ANSI SQL 2003 specification for INFORMATION_SCHEMA?

Unfortunately, the official specifications are not freely available. (ANSI makes them available for purchase.) However, there are books available—such as *SQL-99 Complete, Really* by Peter Gulutzan and Trudy Pelzer—which give a comprehensive overview of the standard, including INFORMATION_SCHEMA.

B.7.4: What is the difference between the Oracle Data Dictionary and MySQL's INFORMATION_SCHEMA?

Both Oracle and MySQL provide metadata in tables. However, Oracle and MySQL use different table names and column names. MySQL's implementation is more similar to those found in DB2 and SQL Server, which also support INFORMATION_SCHEMA as defined in the SQL standard.

B.7.5: Can I add to or otherwise modify the tables found in the INFORMATION_SCHEMA database?

No. Since applications may rely on a certain standard structure, this should not be modified. For this reason, *we cannot support bugs or other issues which result from modifying INFORMATION_SCHEMA tables or data*.

B.8 MySQL 5.7 FAQ: Migration

Questions

- B.8.1: [2925] Where can I find information on how to migrate from MySQL 5.6 to MySQL 5.7?
- B.8.2: [2926] How has storage engine (table type) support changed in MySQL 5.7 from previous versions?

Questions and Answers

B.8.1: Where can I find information on how to migrate from MySQL 5.6 to MySQL 5.7?

For detailed upgrade information, see Section 2.10.1, “Upgrading MySQL”. Do not skip a major version when upgrading, but rather complete the process in steps, upgrading from one major version to the next in each step. This may seem more complicated, but it will you save time and trouble—if you encounter problems during the upgrade, their origin will be easier to identify, either by you or—if you have a MySQL Enterprise subscription—by MySQL support.

B.8.2: How has storage engine (table type) support changed in MySQL 5.7 from previous versions?

Storage engine support has changed as follows:

- Support for `ISAM` tables was removed in MySQL 5.0 and you should now use the `MyISAM` storage engine in place of `ISAM`. To convert a table *`tblname`* from `ISAM` to `MyISAM`, simply issue a statement such as this one:

  ```
  ALTER TABLE tblname ENGINE=MYISAM;
  ```

- Internal `RAID` for `MyISAM` tables was also removed in MySQL 5.0. This was formerly used to allow large tables in file systems that did not support file sizes greater than 2GB. All modern file systems allow for larger tables; in addition, there are now other solutions such as `MERGE` tables and views.
- The `VARCHAR` column type now retains trailing spaces in all storage engines.
- `MEMORY` tables (formerly known as `HEAP` tables) can also contain `VARCHAR` columns.

B.9 MySQL 5.7 FAQ: Security

Questions

- B.9.1: [2926] Where can I find documentation that addresses security issues for MySQL?
- B.9.2: [2927] Does MySQL 5.7 have native support for SSL?
- B.9.3: [2927] Is SSL support be built into MySQL binaries, or must I recompile the binary myself to enable it?
- B.9.4: [2927] Does MySQL 5.7 have built-in authentication against LDAP directories?
- B.9.5: [2927] Does MySQL 5.7 include support for Roles Based Access Control (RBAC)?

Questions and Answers

B.9.1: Where can I find documentation that addresses security issues for MySQL?

The best place to start is Chapter 6, *Security*.

Other portions of the MySQL Documentation which you may find useful with regard to specific security concerns include the following:

- Section 6.1.1, “Security Guidelines”.
- Section 6.1.3, “Making MySQL Secure Against Attackers”.
- Section C.5.4.1, “How to Reset the Root Password”.
- Section 6.1.5, “How to Run MySQL as a Normal User”.
- Section 22.3.2.6, “User-Defined Function Security Precautions”.
- Section 6.1.4, “Security-Related `mysqld` Options and Variables”.

- Section 6.1.6, “Security Issues with `LOAD DATA LOCAL`”.
- Section 2.9, “Postinstallation Setup and Testing”.
- Section 6.3.11.1, “Basic SSL Concepts”.

B.9.2: Does MySQL 5.7 have native support for SSL?

Most 5.7 binaries have support for SSL connections between the client and server. See Section 6.3.11, “Using SSL for Secure Connections”.

You can also tunnel a connection using SSH, if (for example) the client application does not support SSL connections. For an example, see Section 6.3.12, “Connecting to MySQL Remotely from Windows with SSH”.

B.9.3: Is SSL support be built into MySQL binaries, or must I recompile the binary myself to enable it?

Most 5.7 binaries have SSL enabled for client-server connections that are secured, authenticated, or both. See Section 6.3.11, “Using SSL for Secure Connections”.

B.9.4: Does MySQL 5.7 have built-in authentication against LDAP directories?

Not at this time.

B.9.5: Does MySQL 5.7 include support for Roles Based Access Control (RBAC)?

Not at this time.

B.10 MySQL 5.7 FAQ: MySQL Cluster

Questions

- B.10.1: [2927] Which versions of the MySQL software support Cluster? Do I have to compile from source?

Questions and Answers

B.10.1: Which versions of the MySQL software support Cluster? Do I have to compile from source?

MySQL Cluster is not supported in MySQL Server 5.7 releases. Instead, MySQL Cluster is released as a separate product, available as MySQL Cluster NDB 7.2 and MySQL Cluster NDB 7.3. You should use MySQL Cluster NDB 7.3 for new deployments, and plan to upgrade if you are using a previous version of MySQL Cluster. For an overview of improvements made in MySQL Cluster NDB 7.2, see MySQL Cluster Development in MySQL Cluster NDB 7.2; for information about improvements made in MySQL Cluster NDB 7.3, see MySQL Cluster Development in MySQL Cluster NDB 7.3.

For detailed information about deploying and using MySQL Cluster, see MySQL Cluster NDB 7.2 and MySQL Cluster NDB 7.3.

B.11 MySQL 5.7 FAQ: MySQL Chinese, Japanese, and Korean Character Sets

This set of Frequently Asked Questions derives from the experience of MySQL's Support and Development groups in handling many inquiries about CJK (Chinese-Japanese-Korean) issues.

Questions

- B.11.1: [2928] What CJK character sets are available in MySQL?
- B.11.2: [2929] I have inserted CJK characters into my table. Why does SELECT display them as “?” characters?
- B.11.3: [2931] What problems should I be aware of when working with the Big5 Chinese character set?
- B.11.4: [2931] Why do Japanese character set conversions fail?
- B.11.5: [2932] What should I do if I want to convert SJIS 81CA to cp932?
- B.11.6: [2932] How does MySQL represent the Yen (¥) sign?
- B.11.7: [2933] Does MySQL plan to make a separate character set where 5C is the Yen sign, as at least one other major DBMS does?
- B.11.8: [2933] Of what issues should I be aware when working with Korean character sets in MySQL?
- B.11.9: [2933] Why do I get Incorrect string value error messages?
- B.11.10: [2934] Why does my GUI front end or browser not display CJK characters correctly in my application using Access, PHP, or another API?
- B.11.11: [2935] I've upgraded to MySQL 5.7. How can I revert to behavior like that in MySQL 4.0 with regard to character sets?
- B.11.12: [2936] Why do some LIKE and FULLTEXT searches with CJK characters fail?
- B.11.13: [2937] How do I know whether character *X* is available in all character sets?
- B.11.14: [2938] Why do CJK strings sort incorrectly in Unicode? (I)
- B.11.15: [2939] Why do CJK strings sort incorrectly in Unicode? (II)
- B.11.16: [2939] Why are my supplementary characters rejected by MySQL?
- B.11.17: [2940] Shouldn't it be “CJKV”?
- B.11.18: [2940] Does MySQL allow CJK characters to be used in database and table names?
- B.11.19: [2940] Where can I find translations of the MySQL Manual into Chinese, Japanese, and Korean?
- B.11.20: [2940] Where can I get help with CJK and related issues in MySQL?

Questions and Answers

B.11.1: What CJK character sets are available in MySQL?

The list of CJK character sets may vary depending on your MySQL version. For example, the eucjpms character set was not supported prior to MySQL 5.0.3. However, since the name of the applicable language appears in the DESCRIPTION column for every entry in the INFORMATION_SCHEMA.CHARACTER_SETS table, you can obtain a current list of all the non-Unicode CJK character sets using this query:

```
mysql> SELECT CHARACTER_SET_NAME, DESCRIPTION
    -> FROM INFORMATION_SCHEMA.CHARACTER_SETS
    -> WHERE DESCRIPTION LIKE '%Chinese%'
    -> OR DESCRIPTION LIKE '%Japanese%'
    -> OR DESCRIPTION LIKE '%Korean%'
    -> ORDER BY CHARACTER_SET_NAME;
+--------------------+---------------------------+
| CHARACTER_SET_NAME | DESCRIPTION               |
+--------------------+---------------------------+
| big5               | Big5 Traditional Chinese  |
| cp932              | SJIS for Windows Japanese |
| eucjpms            | UJIS for Windows Japanese |
| euckr              | EUC-KR Korean             |
| gb2312             | GB2312 Simplified Chinese |
| gbk                | GBK Simplified Chinese    |
| sjis               | Shift-JIS Japanese        |
| ujis               | EUC-JP Japanese           |
+--------------------+---------------------------+
8 rows in set (0.01 sec)
```

(See Section 19.1, “The INFORMATION_SCHEMA CHARACTER_SETS Table”, for more information.)

MySQL supports the two common variants of the GB (*Guojia Biaozhun*, or *National Standard*, or *Simplified Chinese*) character sets which are official in the People's Republic of China: gb2312 and gbk. Sometimes people try to insert gbk characters into gb2312, and it works most of the time because gbk is a superset of gb2312—but eventually they try to insert a rarer Chinese character and it doesn't work. (See Bug #16072 for an example).

Here, we try to clarify exactly what characters are legitimate in gb2312 or gbk, with reference to the official documents. Please check these references before reporting gb2312 or gbk bugs.

- For a complete listing of the gb2312 characters, ordered according to the gb2312_chinese_ci collation: gb2312
- MySQL's gbk is in reality “Microsoft code page 936”. This differs from the official gbk for characters A1A4 (middle dot), A1AA (em dash), A6E0-A6F5, and A8BB-A8C0.
- For a listing of gbk/Unicode mappings, see http://www.unicode.org/Public/MAPPINGS/VENDORS/MICSFT/WINDOWS/CP936.TXT.
- For MySQL's listing of gbk characters, see gbk.

B.11.2: I have inserted CJK characters into my table. Why does SELECT display them as “?” characters?

This problem is usually due to a setting in MySQL that doesn't match the settings for the application program or the operating system. Here are some common steps for correcting these types of issues:

- *Be certain of what MySQL version you are using.*

  Use the statement SELECT VERSION(); to determine this.

- *Make sure that the database is actually using the desired character set.*

  People often think that the client character set is always the same as either the server character set or the character set used for display purposes. However, both of these are false assumptions. You can make sure by checking the result of SHOW CREATE TABLE *tablename* or—better yet—by using this statement:

  ```
  SELECT character_set_name, collation_name
      FROM information_schema.columns
      WHERE table_schema = your_database_name
  ```

```
        AND table_name = your_table_name
        AND column_name = your_column_name;
```

- *Determine the hexadecimal value of the character or characters that are not being displayed correctly*.

  You can obtain this information for a column *`column_name`* in the table *`table_name`* using the following query:

  ```
  SELECT HEX(column_name)
  FROM table_name;
  ```

  `3F` is the encoding for the `?` character; this means that `?` is the character actually stored in the column. This most often happens because of a problem converting a particular character from your client character set to the target character set.

- *Make sure that a round trip possible—that is, when you select `literal` (or `_introducer hexadecimal-value`), you obtain `literal` as a result*.

  For example, the Japanese *Katakana* character *Pe* (`ペ'`) exists in all CJK character sets, and has the code point value (hexadecimal coding) `0x30da`. To test a round trip for this character, use this query:

  ```
  SELECT 'ペ' AS `ペ`;         /* or SELECT _ucs2 0x30da; */
  ```

  If the result is not also `ペ`, then the round trip has failed.

  For bug reports regarding such failures, we might ask you to follow up with `SELECT HEX('ペ');`. Then we can determine whether the client encoding is correct.

- *Make sure that the problem is not with the browser or other application, rather than with MySQL*.

  Use the `mysql` client program (on Windows: `mysql.exe`) to accomplish this task. If `mysql` displays correctly but your application doesn't, then your problem is probably due to system settings.

  To find out what your settings are, use the `SHOW VARIABLES` statement, whose output should resemble what is shown here:

  ```
  mysql> SHOW VARIABLES LIKE 'char%';
  +--------------------------+----------------------------------------+
  | Variable_name            | Value                                  |
  +--------------------------+----------------------------------------+
  | character_set_client     | utf8                                   |
  | character_set_connection | utf8                                   |
  | character_set_database   | latin1                                 |
  | character_set_filesystem | binary                                 |
  | character_set_results    | utf8                                   |
  | character_set_server     | latin1                                 |
  | character_set_system     | utf8                                   |
  | character_sets_dir       | /usr/local/mysql/share/mysql/charsets/ |
  +--------------------------+----------------------------------------+
  8 rows in set (0.03 sec)
  ```

  These are typical character-set settings for an international-oriented client (notice the use of `utf8` Unicode) connected to a server in the West (`latin1` is a West Europe character set and a default for MySQL).

  Although Unicode (usually the `utf8` variant on Unix, and the `ucs2` variant on Windows) is preferable to Latin, it is often not what your operating system utilities support best. Many Windows users find that a Microsoft character set, such as `cp932` for Japanese Windows, is suitable.

If you cannot control the server settings, and you have no idea what your underlying computer is, then try changing to a common character set for the country that you're in (euckr = Korea; gb2312 or gbk = People's Republic of China; big5 = Taiwan; sjis, ujis, cp932, or eucjpms = Japan; ucs2 or utf8 = anywhere). Usually it is necessary to change only the client and connection and results settings. There is a simple statement which changes all three at once: SET NAMES. For example:

```
SET NAMES 'big5';
```

Once the setting is correct, you can make it permanent by editing my.cnf or my.ini. For example you might add lines looking like these:

```
[mysqld]
character-set-server=big5
[client]
default-character-set=big5
```

It is also possible that there are issues with the API configuration setting being used in your application; see *Why does my GUI front end or browser not display CJK characters correctly...?* for more information.

B.11.3: What problems should I be aware of when working with the Big5 Chinese character set?

MySQL supports the Big5 character set which is common in Hong Kong and Taiwan (Republic of China). MySQL's big5 is in reality Microsoft code page 950, which is very similar to the original big5 character set. We changed to this character set starting with MySQL version 4.1.16 / 5.0.16 (as a result of Bug #12476). For example, the following statements work in current versions of MySQL, but not in old versions:

```
mysql> CREATE TABLE big5 (BIG5 CHAR(1) CHARACTER SET BIG5);
Query OK, 0 rows affected (0.13 sec)

mysql> INSERT INTO big5 VALUES (0xf9dc);
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM big5;
+------+
| big5 |
+------+
| 媦   |
+------+
1 row in set (0.02 sec)
```

A feature request for adding HKSCS extensions has been filed. People who need this extension may find the suggested patch for Bug #13577 to be of interest.

B.11.4: Why do Japanese character set conversions fail?

MySQL supports the sjis, ujis, cp932, and eucjpms character sets, as well as Unicode. A common need is to convert between character sets. For example, there might be a Unix server (typically with sjis or ujis) and a Windows client (typically with cp932).

In the following conversion table, the ucs2 column represents the source, and the sjis, cp932, ujis, and eucjpms columns represent the destinations—that is, the last 4 columns provide the hexadecimal result when we use CONVERT(ucs2) or we assign a ucs2 column containing the value to an sjis, cp932, ujis, or eucjpms column.

Character Name	ucs2	sjis	cp932	ujis	eucjpms
BROKEN BAR	00A6	3F	3F	8FA2C3	3F

Character Name	ucs2	sjis	cp932	ujis	eucjpms
FULLWIDTH BROKEN BAR	FFE4	3F	FA55	3F	8FA2
YEN SIGN	00A5	3F	3F	20	3F
FULLWIDTH YEN SIGN	FFE5	818F	818F	A1EF	3F
TILDE	007E	7E	7E	7E	7E
OVERLINE	203E	3F	3F	20	3F
HORIZONTAL BAR	2015	815C	815C	A1BD	A1BD
EM DASH	2014	3F	3F	3F	3F
REVERSE SOLIDUS	005C	815F	5C	5C	5C
FULLWIDTH ""	FF3C	3F	815F	3F	A1C0
WAVE DASH	301C	8160	3F	A1C1	3F
FULLWIDTH TILDE	FF5E	3F	8160	3F	A1C1
DOUBLE VERTICAL LINE	2016	8161	3F	A1C2	3F
PARALLEL TO	2225	3F	8161	3F	A1C2
MINUS SIGN	2212	817C	3F	A1DD	3F
FULLWIDTH HYPHEN-MINUS	FF0D	3F	817C	3F	A1DD
CENT SIGN	00A2	8191	3F	A1F1	3F
FULLWIDTH CENT SIGN	FFE0	3F	8191	3F	A1F1
POUND SIGN	00A3	8192	3F	A1F2	3F
FULLWIDTH POUND SIGN	FFE1	3F	8192	3F	A1F2
NOT SIGN	00AC	81CA	3F	A2CC	3F
FULLWIDTH NOT SIGN	FFE2	3F	81CA	3F	A2CC

Now consider the following portion of the table.

	ucs2	sjis	cp932
NOT SIGN	00AC	81CA	3F
FULLWIDTH NOT SIGN	FFE2	3F	81CA

This means that MySQL converts the `NOT SIGN` (Unicode `U+00AC`) to `sjis` code point `0x81CA` and to `cp932` code point `3F`. (`3F` is the question mark (“?”)—this is what is always used when the conversion cannot be performed.

B.11.5: What should I do if I want to convert SJIS `81CA` to `cp932`?

Our answer is: “?”. There are serious complaints about this: many people would prefer a “loose” conversion, so that `81CA (NOT SIGN)` in `sjis` becomes `81CA (FULLWIDTH NOT SIGN)` in `cp932`. We are considering a change to this behavior.

B.11.6: How does MySQL represent the Yen (`¥`) sign?

A problem arises because some versions of Japanese character sets (both `sjis` and `euc`) treat `5C` as a *reverse solidus* (`\`—also known as a backslash), and others treat it as a yen sign (`¥`).

MySQL follows only one version of the JIS (Japanese Industrial Standards) standard description. In MySQL, *`5C` is always the reverse solidus (`\`)*.

B.11.7: Does MySQL plan to make a separate character set where `5C` is the Yen sign, as at least one other major DBMS does?

This is one possible solution to the Yen sign issue; however, this will not happen in MySQL 5.1 or 6.0.

B.11.8: Of what issues should I be aware when working with Korean character sets in MySQL?

In theory, while there have been several versions of the `euckr` (*Extended Unix Code Korea*) character set, only one problem has been noted.

We use the “ASCII” variant of EUC-KR, in which the code point `0x5c` is REVERSE SOLIDUS, that is `\`, instead of the “KS-Roman” variant of EUC-KR, in which the code point `0x5c` is `WON SIGN`(₩). This means that you cannot convert Unicode `U+20A9` to `euckr`:

```
mysql> SELECT
    ->     CONVERT('₩' USING euckr) AS euckr,
    ->     HEX(CONVERT('₩' USING euckr)) AS hexeuckr;
+-------+----------+
| euckr | hexeuckr |
+-------+----------+
| ?     | 3F       |
+-------+----------+
1 row in set (0.00 sec)
```

MySQL's graphic Korean chart is here: euckr.

B.11.9: Why do I get `Incorrect string value` error messages?

For illustration, we'll create a table with one Unicode (`ucs2`) column and one Chinese (`gb2312`) column.

```
mysql> CREATE TABLE ch
    -> (ucs2 CHAR(3) CHARACTER SET ucs2,
    -> gb2312 CHAR(3) CHARACTER SET gb2312);
Query OK, 0 rows affected (0.05 sec)
```

We'll try to place the rare character 汌 in both columns.

```
mysql> INSERT INTO ch VALUES ('A汌B','A汌B');
Query OK, 1 row affected, 1 warning (0.00 sec)
```

Ah, there is a warning. Use the following statement to see what it is:

```
mysql> SHOW WARNINGS\G
*************************** 1. row ***************************
  Level: Warning
   Code: 1366
Message: Incorrect string value: '\xE6\xB1\x8CB' for column 'gb2312' at row 1
1 row in set (0.00 sec)
```

So it is a warning about the `gb2312` column only.

```
mysql> SELECT ucs2,HEX(ucs2),gb2312,HEX(gb2312) FROM ch;
+-------+--------------+--------+-------------+
| ucs2  | HEX(ucs2)    | gb2312 | HEX(gb2312) |
+-------+--------------+--------+-------------+
| A汌B | 00416C4C0042 | A?B    | 413F42      |
+-------+--------------+--------+-------------+
1 row in set (0.00 sec)
```

Several things need explanation here:

1. The fact that it is a “warning” rather than an “error” is characteristic of MySQL. We like to try to do what we can, to get the best fit, rather than give up.
2. The 汌 character is not in the `gb2312` character set. We described that problem earlier.
3. If you are using an old version of MySQL, you will probably see a different message.
4. With `sql_mode=TRADITIONAL`, there would be an error message, rather than a warning.

B.11.10: Why does my GUI front end or browser not display CJK characters correctly in my application using Access, PHP, or another API?

Obtain a direct connection to the server using the `mysql` client (Windows: `mysql.exe`), and try the same query there. If `mysql` responds correctly, then the trouble may be that your application interface requires initialization. Use `mysql` to tell you what character set or sets it uses with the statement `SHOW VARIABLES LIKE 'char%';`. If you are using Access, then you are most likely connecting with Connector/ODBC. In this case, you should check Configuring Connector/ODBC. If, for instance, you use `big5`, you would enter `SET NAMES 'big5'`. (Note that no `;` is required in this case). If you are using ASP, you might need to add `SET NAMES` in the code. Here is an example that has worked in the past:

```
<%
Session.CodePage=0
Dim strConnection
Dim Conn
strConnection="driver={MySQL ODBC 3.51 Driver};server=server;uid=username;" \
               & "pwd=password;database=database;stmt=SET NAMES 'big5';"
Set Conn = Server.CreateObject("ADODB.Connection")
Conn.Open strConnection
%>
```

In much the same way, if you are using any character set other than `latin1` with Connector/Net, then you must specify the character set in the connection string. See Connecting to MySQL Using Connector/Net, for more information.

If you are using PHP, try this:

```
<?php
  $link = mysql_connect($host, $usr, $pwd);

  mysql_select_db($db);

  if( mysql_error() ) { print "Database ERROR: " . mysql_error(); }
  mysql_query("SET NAMES 'utf8'", $link);
?>
```

In this case, we used `SET NAMES` to change `character_set_client` and `character_set_connection` and `character_set_results`.

We encourage the use of the newer `mysqli` extension, rather than `mysql`. Using `mysqli`, the previous example could be rewritten as shown here:

```
<?php
  $link = new mysqli($host, $usr, $pwd, $db);
  if( mysqli_connect_errno() )
  {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
  }

  $link->query("SET NAMES 'utf8'");
?>
```

Another issue often encountered in PHP applications has to do with assumptions made by the browser. Sometimes adding or changing a `<meta>` tag suffices to correct the problem: for example, to insure that the user agent interprets page content as `UTF-8`, you should include `<meta http-equiv="Content-Type" content="text/html; charset=utf-8">` in the `<head>` of the HTML page.

If you are using Connector/J, see Using Character Sets and Unicode.

B.11.11: I've upgraded to MySQL 5.7. How can I revert to behavior like that in MySQL 4.0 with regard to character sets?

In MySQL Version 4.0, there was a single “global” character set for both server and client, and the decision as to which character to use was made by the server administrator. This changed starting with MySQL Version 4.1. What happens now is a “handshake”, as described in Section 10.1.4, “Connection Character Sets and Collations”:

> When a client connects, it sends to the server the name of the character set that it wants to use. The server uses the name to set the `character_set_client`, `character_set_results`, and `character_set_connection` system variables. In effect, the server performs a `SET NAMES` operation using the character set name.

The effect of this is that you cannot control the client character set by starting `mysqld` with `--character-set-server=utf8`. However, some of our Asian customers have said that they prefer the MySQL 4.0 behavior. To make it possible to retain this behavior, we added a `mysqld` switch, `--character-set-client-handshake`, which can be turned off with `--skip-character-set-client-handshake`. If you start `mysqld` with `--skip-character-set-client-handshake`, then, when a client connects, it sends to the server the name of the character set that it wants to use—however, *the server ignores this request from the client*.

By way of example, suppose that your favorite server character set is `latin1` (unlikely in a CJK area, but this is the default value). Suppose further that the client uses `utf8` because this is what the client's operating system supports. Now, start the server with `latin1` as its default character set:

```
mysqld --character-set-server=latin1
```

And then start the client with the default character set `utf8`:

```
mysql --default-character-set=utf8
```

The current settings can be seen by viewing the output of `SHOW VARIABLES`:

```
mysql> SHOW VARIABLES LIKE 'char%';
+--------------------------+----------------------------------------+
| Variable_name            | Value                                  |
+--------------------------+----------------------------------------+
| character_set_client     | utf8                                   |
| character_set_connection | utf8                                   |
| character_set_database   | latin1                                 |
| character_set_filesystem | binary                                 |
| character_set_results    | utf8                                   |
| character_set_server     | latin1                                 |
| character_set_system     | utf8                                   |
| character_sets_dir       | /usr/local/mysql/share/mysql/charsets/ |
+--------------------------+----------------------------------------+
8 rows in set (0.01 sec)
```

Now stop the client, and then stop the server using `mysqladmin`. Then start the server again, but this time tell it to skip the handshake like so:

```
mysqld --character-set-server=utf8 --skip-character-set-client-handshake
```

Start the client with `utf8` once again as the default character set, then display the current settings:

```
mysql> SHOW VARIABLES LIKE 'char%';
+--------------------------+----------------------------------------+
| Variable_name            | Value                                  |
+--------------------------+----------------------------------------+
| character_set_client     | latin1                                 |
| character_set_connection | latin1                                 |
| character_set_database   | latin1                                 |
| character_set_filesystem | binary                                 |
| character_set_results    | latin1                                 |
| character_set_server     | latin1                                 |
| character_set_system     | utf8                                   |
| character_sets_dir       | /usr/local/mysql/share/mysql/charsets/ |
+--------------------------+----------------------------------------+
8 rows in set (0.01 sec)
```

As you can see by comparing the differing results from `SHOW VARIABLES`, the server ignores the client's initial settings if the `--skip-character-set-client-handshake` is used.

B.11.12: Why do some `LIKE` and `FULLTEXT` searches with CJK characters fail?

There is a very simple problem with `LIKE` searches on `BINARY` and `BLOB` columns: we need to know the end of a character. With multi-byte character sets, different characters might have different octet lengths. For example, in `utf8`, `A` requires one byte but `ペ` requires three bytes, as shown here:

```
+-------------------------+--------------------------+
| OCTET_LENGTH(_utf8 'A') | OCTET_LENGTH(_utf8 'ペ') |
+-------------------------+--------------------------+
|                       1 |                        3 |
+-------------------------+--------------------------+
1 row in set (0.00 sec)
```

If we don't know where the first character ends, then we don't know where the second character begins, in which case even very simple searches such as `LIKE '_A%'` fail. The solution is to use a regular CJK character set in the first place, or to convert to a CJK character set before comparing.

This is one reason why MySQL cannot allow encodings of nonexistent characters. If it is not strict about rejecting bad input, then it has no way of knowing where characters end.

For `FULLTEXT` searches, we need to know where words begin and end. With Western languages, this is rarely a problem because most (if not all) of these use an easy-to-identify word boundary—the space character. However, this is not usually the case with Asian writing. We could use arbitrary halfway measures, like assuming that all Han characters represent words, or (for Japanese) depending on changes from Katakana to Hiragana due to grammatical endings. However, the only sure solution requires a comprehensive word list, which means that we would have to include a dictionary in the server for each Asian language supported. This is simply not feasible.

B.11.13: How do I know whether character *X* is available in all character sets?

The majority of simplified Chinese and basic nonhalfwidth Japanese *Kana* characters appear in all CJK character sets. This stored procedure accepts a UCS-2 Unicode character, converts it to all other character sets, and displays the results in hexadecimal.

```
DELIMITER //

CREATE PROCEDURE p_convert(ucs2_char CHAR(1) CHARACTER SET ucs2)
BEGIN

CREATE TABLE tj
             (ucs2 CHAR(1) character set ucs2,
              utf8 CHAR(1) character set utf8,
              big5 CHAR(1) character set big5,
              cp932 CHAR(1) character set cp932,
              eucjpms CHAR(1) character set eucjpms,
              euckr CHAR(1) character set euckr,
              gb2312 CHAR(1) character set gb2312,
              gbk CHAR(1) character set gbk,
              sjis CHAR(1) character set sjis,
              ujis CHAR(1) character set ujis);

INSERT INTO tj (ucs2) VALUES (ucs2_char);

UPDATE tj SET utf8=ucs2,
              big5=ucs2,
              cp932=ucs2,
              eucjpms=ucs2,
              euckr=ucs2,
              gb2312=ucs2,
              gbk=ucs2,
              sjis=ucs2,
              ujis=ucs2;

/* If there is a conversion problem, UPDATE will produce a warning. */

SELECT hex(ucs2) AS ucs2,
       hex(utf8) AS utf8,
       hex(big5) AS big5,
       hex(cp932) AS cp932,
       hex(eucjpms) AS eucjpms,
       hex(euckr) AS euckr,
       hex(gb2312) AS gb2312,
       hex(gbk) AS gbk,
       hex(sjis) AS sjis,
       hex(ujis) AS ujis
FROM tj;

DROP TABLE tj;

END//
```

The input can be any single ucs2 character, or it can be the code point value (hexadecimal representation) of that character. For example, from Unicode's list of ucs2 encodings and names (http://www.unicode.org/Public/UNIDATA/UnicodeData.txt), we know that the *Katakana* character *Pe* appears in all CJK character sets, and that its code point value is 0x30da. If we use this value as the argument to p_convert(), the result is as shown here:

```
mysql> CALL p_convert(0x30da)//
```

```
+------+--------+------+-------+---------+-------+--------+------+------+------+
| ucs2 | utf8   | big5 | cp932 | eucjpms | euckr | gb2312 | gbk  | sjis | ujis |
+------+--------+------+-------+---------+-------+--------+------+------+------+
| 30DA | E3839A | C772 | 8379  | A5DA    | ABDA  | A5DA   | A5DA | 8379 | A5DA |
+------+--------+------+-------+---------+-------+--------+------+------+------+
1 row in set (0.04 sec)
```

Since none of the column values is 3F—that is, the question mark character (?)—we know that every conversion worked.

B.11.14: Why do CJK strings sort incorrectly in Unicode? (I)

Sometimes people observe that the result of a utf8_unicode_ci or ucs2_unicode_ci search, or of an ORDER BY sort is not what they think a native would expect. Although we never rule out the possibility that there is a bug, we have found in the past that many people do not read correctly the standard table of weights for the Unicode Collation Algorithm. MySQL uses the table found at http://www.unicode.org/Public/UCA/4.0.0/allkeys-4.0.0.txt. This is not the first table you will find by navigating from the unicode.org home page, because MySQL uses the older 4.0.0 “allkeys” table, rather than the more recent 4.1.0 table. (The newer '520' collations in MySQL 5.6 use the 5.2 “allkeys” table.) This is because we are very wary about changing ordering which affects indexes, lest we bring about situations such as that reported in Bug #16526, illustrated as follows:

```
mysql< CREATE TABLE tj (s1 CHAR(1) CHARACTER SET utf8 COLLATE utf8_unicode_ci);
Query OK, 0 rows affected (0.05 sec)

mysql> INSERT INTO tj VALUES ('が'),('か');
Query OK, 2 rows affected (0.00 sec)
Records: 2  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM tj WHERE s1 = 'か';
+------+
| s1   |
+------+
| が   |
| か   |
+------+
2 rows in set (0.00 sec)
```

The character in the first result row is not the one that we searched for. Why did MySQL retrieve it? First we look for the Unicode code point value, which is possible by reading the hexadecimal number for the ucs2 version of the characters:

```
mysql> SELECT s1, HEX(CONVERT(s1 USING ucs2)) FROM tj;
+------+-----------------------------+
| s1   | HEX(CONVERT(s1 USING ucs2)) |
+------+-----------------------------+
| が   | 304C                        |
| か   | 304B                        |
+------+-----------------------------+
2 rows in set (0.03 sec)
```

Now we search for 304B and 304C in the 4.0.0 allkeys table, and find these lines:

```
304B  ; [.1E57.0020.000E.304B] # HIRAGANA LETTER KA
304C  ; [.1E57.0020.000E.304B][.0000.0140.0002.3099] # HIRAGANA LETTER GA; QQCM
```

The official Unicode names (following the “#” mark) tell us the Japanese syllabary (Hiragana), the informal classification (letter, digit, or punctuation mark), and the Western identifier (KA or GA, which happen to be voiced and unvoiced components of the same letter pair). More importantly, the *primary weight* (the first hexadecimal number inside the square brackets) is 1E57 on both lines. For comparisons in both searching and sorting, MySQL pays attention to the primary weight only, ignoring all the other numbers. This means that we are sorting が and か correctly according to the Unicode specification. If we wanted to distinguish them, we'd have to use a non-UCA (Unicode Collation Algorithm) collation (utf8_bin or utf8_general_ci), or to compare the HEX() values, or use ORDER BY CONVERT(s1 USING sjis). Being correct "according to Unicode" isn't enough, of course: the person who submitted the bug was equally correct. We plan to add another collation for Japanese according to the JIS X 4061 standard, in which voiced/unvoiced letter pairs like KA/GA are distinguishable for ordering purposes.

B.11.15: Why do CJK strings sort incorrectly in Unicode? (II)

If you are using Unicode (ucs2 or utf8), and you know what the Unicode sort order is (see Section B.11, "MySQL 5.7 FAQ: MySQL Chinese, Japanese, and Korean Character Sets"), but MySQL still seems to sort your table incorrectly, then you should first verify the table character set:

```
mysql> SHOW CREATE TABLE t\G
******************** 1. row ******************
Table: t
Create Table: CREATE TABLE `t` (
`s1` char(1) CHARACTER SET ucs2 DEFAULT NULL
) ENGINE=MyISAM DEFAULT CHARSET=latin1
1 row in set (0.00 sec)
```

Since the character set appears to be correct, let's see what information the INFORMATION_SCHEMA.COLUMNS table can provide about this column:

```
mysql> SELECT COLUMN_NAME, CHARACTER_SET_NAME, COLLATION_NAME
    -> FROM INFORMATION_SCHEMA.COLUMNS
    -> WHERE COLUMN_NAME = 's1'
    -> AND TABLE_NAME = 't';
+-------------+--------------------+-----------------+
| COLUMN_NAME | CHARACTER_SET_NAME | COLLATION_NAME  |
+-------------+--------------------+-----------------+
| s1          | ucs2               | ucs2_general_ci |
+-------------+--------------------+-----------------+
1 row in set (0.01 sec)
```

(See Section 19.4, "The INFORMATION_SCHEMA COLUMNS Table", for more information.)

You can see that the collation is ucs2_general_ci instead of ucs2_unicode_ci. The reason why this is so can be found using SHOW CHARSET, as shown here:

```
mysql> SHOW CHARSET LIKE 'ucs2%';
+---------+---------------+-------------------+--------+
| Charset | Description   | Default collation | Maxlen |
+---------+---------------+-------------------+--------+
| ucs2    | UCS-2 Unicode | ucs2_general_ci   |      2 |
+---------+---------------+-------------------+--------+
1 row in set (0.00 sec)
```

For ucs2 and utf8, the default collation is "general". To specify a Unicode collation, use COLLATE ucs2_unicode_ci.

B.11.16: Why are my supplementary characters rejected by MySQL?

Before MySQL 5.5.3, MySQL does not support supplementary characters—that is, characters which need more than 3 bytes—for UTF-8. We support only what Unicode calls the *Basic Multilingual Plane / Plane 0*. Only a few very rare Han characters are supplementary; support for them is uncommon. This has led to reports such as that found in Bug #12600, which we rejected as “not a bug”. With `utf8`, we must truncate an input string when we encounter bytes that we don't understand. Otherwise, we wouldn't know how long the bad multi-byte character is.

One possible workaround is to use `ucs2` instead of `utf8`, in which case the “bad” characters are changed to question marks; however, no truncation takes place. You can also change the data type to `BLOB` or `BINARY`, which perform no validity checking.

As of MySQL 5.5.3, Unicode support is extended to include supplementary characters by means of additional Unicode character sets: `utf16`, `utf32`, and 4-byte `utf8mb4`. These character sets support supplementary Unicode characters outside the Basic Multilingual Plane (BMP).

B.11.17: Shouldn't it be “CJKV”?

No. The term “CJKV” (*Chinese Japanese Korean Vietnamese*) refers to Vietnamese character sets which contain Han (originally Chinese) characters. MySQL has no plan to support the old Vietnamese script using Han characters. MySQL does of course support the modern Vietnamese script with Western characters.

As of MySQL 5.6, there are Vietnamese collations for Unicode character sets, as described in Section 10.1.14.1, “Unicode Character Sets”.

B.11.18: Does MySQL allow CJK characters to be used in database and table names?

This issue is fixed in MySQL 5.1, by automatically rewriting the names of the corresponding directories and files.

For example, if you create a database named `楮` on a server whose operating system does not support CJK in directory names, MySQL creates a directory named `@0w@00a5@00ae`. which is just a fancy way of encoding `E6A5AE`—that is, the Unicode hexadecimal representation for the `楮` character. However, if you run a `SHOW DATABASES` statement, you can see that the database is listed as `楮`.

B.11.19: Where can I find translations of the MySQL Manual into Chinese, Japanese, and Korean?

A Simplified Chinese version of the Manual, current for MySQL 5.1.12, can be found at http://dev.mysql.com/doc/. The Japanese translation of the MySQL 4.1 manual can be downloaded from http://dev.mysql.com/doc/.

B.11.20: Where can I get help with CJK and related issues in MySQL?

The following resources are available:

- A listing of MySQL user groups can be found at http://dev.mysql.com/user-groups/.
- View feature requests relating to character set issues at http://tinyurl.com/y6xcuf.
- Visit the MySQL Character Sets, Collation, Unicode Forum. We are also in the process of adding foreign-language forums at http://forums.mysql.com/.

B.12 MySQL 5.7 FAQ: Connectors & APIs

For common questions, issues, and answers relating to the MySQL Connectors and other APIs, see the following areas of the Manual:

- Section 21.8.15, “Common Questions and Problems When Using the C API”
- Common Problems with MySQL and PHP

- Connector/ODBC Notes and Tips
- Connector/Net Programming
- MySQL Connector/J Developer Guide

B.13 MySQL 5.7 FAQ: Replication

In the following section, we provide answers to questions that are most frequently asked about MySQL Replication.

Questions

- B.13.1: [2941] Must the slave be connected to the master all the time?
- B.13.2: [2942] Must I enable networking on my master and slave to enable replication?
- B.13.3: [2942] How do I know how late a slave is compared to the master? In other words, how do I know the date of the last statement replicated by the slave?
- B.13.4: [2942] How do I force the master to block updates until the slave catches up?
- B.13.5: [2942] What issues should I be aware of when setting up two-way replication?
- B.13.6: [2943] How can I use replication to improve performance of my system?
- B.13.7: [2943] What should I do to prepare client code in my own applications to use performance-enhancing replication?
- B.13.8: [2943] When and how much can MySQL replication improve the performance of my system?
- B.13.9: [2944] How can I use replication to provide redundancy or high availability?
- B.13.10: [2944] How do I tell whether a master server is using statement-based or row-based binary logging format?
- B.13.11: [2944] How do I tell a slave to use row-based replication?
- B.13.12: [2944] How do I prevent `GRANT` and `REVOKE` statements from replicating to slave machines?
- B.13.13: [2945] Does replication work on mixed operating systems (for example, the master runs on Linux while slaves run on Mac OS X and Windows)?
- B.13.14: [2945] Does replication work on mixed hardware architectures (for example, the master runs on a 64-bit machine while slaves run on 32-bit machines)?

Questions and Answers

B.13.1: Must the slave be connected to the master all the time?

No, it does not. The slave can go down or stay disconnected for hours or even days, and then reconnect and catch up on updates. For example, you can set up a master/slave relationship over a dial-up link where the link is up only sporadically and for short periods of time. The implication of this is that, at any given time, the slave is not guaranteed to be in synchrony with the master unless you take some special measures.

To ensure that catchup can occur for a slave that has been disconnected, you must not remove binary log files from the master that contain information that has not yet been replicated to the slaves. Asynchronous replication can work only if the slave is able to continue reading the binary log from the point where it last read events.

B.13.2: Must I enable networking on my master and slave to enable replication?

Yes, networking must be enabled on the master and slave. If networking is not enabled, the slave cannot connect to the master and transfer the binary log. Check that the `skip-networking` option has not been enabled in the configuration file for either server.

B.13.3: How do I know how late a slave is compared to the master? In other words, how do I know the date of the last statement replicated by the slave?

Check the `Seconds_Behind_Master` column in the output from `SHOW SLAVE STATUS`. See Section 16.1.5.1, “Checking Replication Status”.

When the slave SQL thread executes an event read from the master, it modifies its own time to the event timestamp. (This is why `TIMESTAMP` is well replicated.) In the `Time` column in the output of `SHOW PROCESSLIST`, the number of seconds displayed for the slave SQL thread is the number of seconds between the timestamp of the last replicated event and the real time of the slave machine. You can use this to determine the date of the last replicated event. Note that if your slave has been disconnected from the master for one hour, and then reconnects, you may immediately see large `Time` values such as 3600 for the slave SQL thread in `SHOW PROCESSLIST`. This is because the slave is executing statements that are one hour old. See Section 16.2.1, “Replication Implementation Details”.

B.13.4: How do I force the master to block updates until the slave catches up?

Use the following procedure:

1. On the master, execute these statements:

   ```
   mysql> FLUSH TABLES WITH READ LOCK;
   mysql> SHOW MASTER STATUS;
   ```

   Record the replication coordinates (the current binary log file name and position) from the output of the `SHOW` statement.

2. On the slave, issue the following statement, where the arguments to the `MASTER_POS_WAIT()` function are the replication coordinate values obtained in the previous step:

   ```
   mysql> SELECT MASTER_POS_WAIT('log_name', log_pos);
   ```

   The `SELECT` statement blocks until the slave reaches the specified log file and position. At that point, the slave is in synchrony with the master and the statement returns.

3. On the master, issue the following statement to enable the master to begin processing updates again:

   ```
   mysql> UNLOCK TABLES;
   ```

B.13.5: What issues should I be aware of when setting up two-way replication?

MySQL replication currently does not support any locking protocol between master and slave to guarantee the atomicity of a distributed (cross-server) update. In other words, it is possible for client A to make an update to co-master 1, and in the meantime, before it propagates to co-master 2, client B could make an update to co-master 2 that makes the update of client A work differently than it did on co-master 1. Thus, when the update of client A makes it to co-master 2, it produces tables that are different from what you have on co-master 1, even after all the updates from co-master 2 have also propagated. This means that you should not chain two servers together in a two-way replication relationship unless you are sure that your updates can safely happen in any order, or unless you take care of mis-ordered updates somehow in the client code.

You should also realize that two-way replication actually does not improve performance very much (if at all) as far as updates are concerned. Each server must do the same number of updates, just as you would have a single server do. The only difference is that there is a little less lock contention because the updates originating on another server are serialized in one slave thread. Even this benefit might be offset by network delays.

B.13.6: How can I use replication to improve performance of my system?

Set up one server as the master and direct all writes to it. Then configure as many slaves as you have the budget and rackspace for, and distribute the reads among the master and the slaves. You can also start the slaves with the `--skip-innodb`, `--low-priority-updates`, and `--delay-key-write=ALL` options to get speed improvements on the slave end. In this case, the slave uses nontransactional `MyISAM` tables instead of `InnoDB` tables to get more speed by eliminating transactional overhead.

B.13.7: What should I do to prepare client code in my own applications to use performance-enhancing replication?

See the guide to using replication as a scale-out solution, Section 16.3.3, “Using Replication for Scale-Out”.

B.13.8: When and how much can MySQL replication improve the performance of my system?

MySQL replication is most beneficial for a system that processes frequent reads and infrequent writes. In theory, by using a single-master/multiple-slave setup, you can scale the system by adding more slaves until you either run out of network bandwidth, or your update load grows to the point that the master cannot handle it.

To determine how many slaves you can use before the added benefits begin to level out, and how much you can improve performance of your site, you must know your query patterns, and determine empirically by benchmarking the relationship between the throughput for reads and writes on a typical master and a typical slave. The example here shows a rather simplified calculation of what you can get with replication for a hypothetical system. Let `reads` and `writes` denote the number of reads and writes per second, respectively.

Let's say that system load consists of 10% writes and 90% reads, and we have determined by benchmarking that `reads` is 1200 - 2 * `writes`. In other words, the system can do 1,200 reads per second with no writes, the average write is twice as slow as the average read, and the relationship is linear. Suppose that the master and each slave have the same capacity, and that we have one master and *`N`* slaves. Then we have for each server (master or slave):

`reads` = 1200 - 2 * `writes`

`reads` = 9 * `writes` / (*`N`* + 1) (reads are split, but writes replicated to all slaves)

9 * `writes` / (*`N`* + 1) + 2 * `writes` = 1200

`writes` = 1200 / (2 + 9/(*`N`* + 1))

The last equation indicates the maximum number of writes for *`N`* slaves, given a maximum possible read rate of 1,200 per second and a ratio of nine reads per write.

This analysis yields the following conclusions:

- If *N* = 0 (which means we have no replication), our system can handle about 1200/11 = 109 writes per second.
- If *N* = 1, we get up to 184 writes per second.
- If *N* = 8, we get up to 400 writes per second.
- If *N* = 17, we get up to 480 writes per second.
- Eventually, as *N* approaches infinity (and our budget negative infinity), we can get very close to 600 writes per second, increasing system throughput about 5.5 times. However, with only eight servers, we increase it nearly four times.

Note that these computations assume infinite network bandwidth and neglect several other factors that could be significant on your system. In many cases, you may not be able to perform a computation similar to the one just shown that accurately predicts what will happen on your system if you add *N* replication slaves. However, answering the following questions should help you decide whether and by how much replication will improve the performance of your system:

- What is the read/write ratio on your system?
- How much more write load can one server handle if you reduce the reads?
- For how many slaves do you have bandwidth available on your network?

B.13.9: How can I use replication to provide redundancy or high availability?

How you implement redundancy is entirely dependent on your application and circumstances. High-availability solutions (with automatic failover) require active monitoring and either custom scripts or third party tools to provide the failover support from the original MySQL server to the slave.

To handle the process manually, you should be able to switch from a failed master to a pre-configured slave by altering your application to talk to the new server or by adjusting the DNS for the MySQL server from the failed server to the new server.

For more information and some example solutions, see Section 16.3.6, “Switching Masters During Failover”.

B.13.10: How do I tell whether a master server is using statement-based or row-based binary logging format?

Check the value of the `binlog_format` system variable:

```
mysql> SHOW VARIABLES LIKE 'binlog_format';
```

The value shown will be one of `STATEMENT`, `ROW`, or `MIXED`. For `MIXED` mode, row-based logging is preferred but replication switches automatically to statement-based logging under certain conditions; for information about when this may occur, see Section 5.2.4.3, “Mixed Binary Logging Format”.

B.13.11: How do I tell a slave to use row-based replication?

Slaves automatically know which format to use.

B.13.12: How do I prevent `GRANT` and `REVOKE` statements from replicating to slave machines?

Start the server with the `--replicate-wild-ignore-table=mysql.%` option to ignore replication for tables in the `mysql` database.

B.13.13: Does replication work on mixed operating systems (for example, the master runs on Linux while slaves run on Mac OS X and Windows)?

Yes.

B.13.14: Does replication work on mixed hardware architectures (for example, the master runs on a 64-bit machine while slaves run on 32-bit machines)?

Yes.

Appendix C Errors, Error Codes, and Common Problems

Table of Contents

This appendix lists common problems and errors that may occur and potential resolutions, in addition to listing the errors that may appear when you call MySQL from any host language. The first section covers problems and resolutions. Detailed information on errors is provided: One list displays server error messages. Another list displays client program messages.

C.1 Sources of Error Information

There are several sources of error information in MySQL:

- Each SQL statement executed results in an error code, an SQLSTATE value, and an error message, as described in Section C.2, “Types of Error Values”. These errors are returned from the server side; see Section C.3, “Server Error Codes and Messages”.

- Errors can occur on the client side, usually involving problems communicating with the server; see Section C.4, “Client Error Codes and Messages”.

- SQL statement warning and error information is available through the `SHOW WARNINGS` and `SHOW ERRORS` statements. The `warning_count` system variable indicates the number of errors, warnings, and notes. The `error_count` system variable indicates the number of errors. Its value excludes warnings and notes.

- The `GET DIAGNOSTICS` statement may be used to inspect the diagnostic information in the diagnostics area. See Section 13.6.7.3, “`GET DIAGNOSTICS` Syntax”.

- `SHOW SLAVE STATUS` statement output includes information about replication errors occurring on the slave side.

- `SHOW ENGINE INNODB STATUS` statement output includes information about the most recent foreign key error if a `CREATE TABLE` statement for an `InnoDB` table fails.

- The `perror` program provides information from the command line about error numbers. See Section 4.8.1, “`perror` — Explain Error Codes”.

Descriptions of server and client errors are provided later in this Appendix. For information about errors related to `InnoDB`, see Section 14.2.17.4, “`InnoDB` Error Handling”.

C.2 Types of Error Values

When an error occurs in MySQL, the server returns two types of error values:

- A MySQL-specific error code. This value is numeric. It is not portable to other database systems.
- An SQLSTATE value. The value is a five-character string (for example, `'42S02'`). The values are taken from ANSI SQL and ODBC and are more standardized.

A message string that provides a textual description of the error is also available.

When an error occurs, the MySQL error code, SQLSTATE value, and message string are available using C API functions:

- MySQL error code: Call `mysql_errno()`
- SQLSTATE value: Call `mysql_sqlstate()`
- Error message: Call `mysql_error()`

For prepared statements, the corresponding error functions are `mysql_stmt_errno()`, `mysql_stmt_sqlstate()`, and `mysql_stmt_error()`. All error functions are described in Section 21.8, “MySQL C API”.

The number of errors, warnings, and notes for the previous statement can be obtained by calling `mysql_warning_count()`. See Section 21.8.7.77, “`mysql_warning_count()`”.

The first two characters of an SQLSTATE value indicate the error class:

- Class = `'00'` indicates success.
- Class = `'01'` indicates a warning.
- Class = `'02'` indicates “not found.” This is relevant within the context of cursors and is used to control what happens when a cursor reaches the end of a data set. This condition also occurs for `SELECT ... INTO` *`var_list`* statements that retrieve no rows.
- Class > `'02'` indicates an exception.

C.3 Server Error Codes and Messages

MySQL programs have access to several types of error information when the server returns an error. For example, the `mysql` client program displays errors using the following format:

```
shell> SELECT * FROM no_such_table;
ERROR 1146 (42S02): Table 'test.no_such_table' doesn't exist
```

The message displayed contains three types of information:

- A numeric error code (`1146`). This number is MySQL-specific and is not portable to other database systems.
- A five-character SQLSTATE value (`'42S02'`). The values are taken from ANSI SQL and ODBC and are more standardized. Not all MySQL error numbers have corresponding SQLSTATE values. In these cases, `'HY000'` (general error) is used.
- A message string that provides a textual description of the error.

For error checking, use error codes, not error messages. Error messages do not change often, but it is possible. Also if the database administrator changes the language setting, that affects the language of error messages.

Error codes are stable across GA releases of a given MySQL series. Before a series reaches GA status, new codes may still be under development and subject to change.

Server error information comes from the following source files. For details about the way that error information is defined, see the MySQL Internals Manual.

- Error message information is listed in the `share/errmsg.txt` file. `%d` and `%s` represent numbers and strings, respectively, that are substituted into the Message values when they are displayed.
- The Error values listed in `share/errmsg.txt` are used to generate the definitions in the `include/mysqld_error.h` and `include/mysqld_ername.h` MySQL source files.
- The SQLSTATE values listed in `share/errmsg.txt` are used to generate the definitions in the `include/sql_state.h` MySQL source file.

Because updates are frequent, it is possible that those files will contain additional error information not listed here.

- Error: `1000` SQLSTATE: `HY000` (`ER_HASHCHK`)

  Message: hashchk

  Unused.

- Error: `1001` SQLSTATE: `HY000` (`ER_NISAMCHK`)

  Message: isamchk

  Unused.

- Error: `1002` SQLSTATE: `HY000` (`ER_NO`)

  Message: NO

  Used in the construction of other messages.

- Error: `1003` SQLSTATE: `HY000` (`ER_YES`)

  Message: YES

  Used in the construction of other messages.

  Extended `EXPLAIN` format generates Note messages. `ER_YES` is used in the `Code` column for these messages in subsequent `SHOW WARNINGS` output.

- Error: `1004` SQLSTATE: `HY000` (`ER_CANT_CREATE_FILE`)

  Message: Can't create file '%s' (errno: %d - %s)

  Occurs for failure to copy an `.frm` file to a new location, during execution of a `CREATE TABLE` *`dst`* `LIKE` *`src`* statement when the server tries to copy the source table `.frm` file to the destination table `.frm` file.

  Possible causes: Permissions problem for source `.frm` file; destination `.frm` file already exists but is not writeable.

- Error: `1005` SQLSTATE: `HY000` (`ER_CANT_CREATE_TABLE`)

  Message: Can't create table '%s' (errno: %d)

- Error: `1006` SQLSTATE: `HY000` (`ER_CANT_CREATE_DB`)

  Message: Can't create database '%s' (errno: %d)

- Error: `1007` SQLSTATE: `HY000` (`ER_DB_CREATE_EXISTS`)

  Message: Can't create database '%s'; database exists

  An attempt to create a database failed because the database already exists.

  Drop the database first if you really want to replace an existing database, or add an `IF NOT EXISTS` clause to the `CREATE DATABASE` statement if to retain an existing database without having the statement produce an error.

- Error: `1008` SQLSTATE: `HY000` (`ER_DB_DROP_EXISTS`)

  Message: Can't drop database '%s'; database doesn't exist

- Error: `1009` SQLSTATE: `HY000` (`ER_DB_DROP_DELETE`)

  Message: Error dropping database (can't delete '%s', errno: %d)

- Error: `1010` SQLSTATE: `HY000` (`ER_DB_DROP_RMDIR`)

  Message: Error dropping database (can't rmdir '%s', errno: %d)

- Error: `1011` SQLSTATE: `HY000` (`ER_CANT_DELETE_FILE`)

  Message: Error on delete of '%s' (errno: %d - %s)

- Error: `1012` SQLSTATE: `HY000` (`ER_CANT_FIND_SYSTEM_REC`)

  Message: Can't read record in system table

  Returned by `InnoDB` for attempts to access `InnoDB` `INFORMATION_SCHEMA` tables when `InnoDB` is unavailable.

- Error: `1013` SQLSTATE: `HY000` (`ER_CANT_GET_STAT`)

  Message: Can't get status of '%s' (errno: %d - %s)

- Error: `1014` SQLSTATE: `HY000` (`ER_CANT_GET_WD`)

  Message: Can't get working directory (errno: %d - %s)

- Error: `1015` SQLSTATE: `HY000` (`ER_CANT_LOCK`)

  Message: Can't lock file (errno: %d - %s)

- Error: `1016` SQLSTATE: `HY000` (`ER_CANT_OPEN_FILE`)

  Message: Can't open file: '%s' (errno: %d - %s)

- Error: `1017` SQLSTATE: `HY000` (`ER_FILE_NOT_FOUND`)

  Message: Can't find file: '%s' (errno: %d - %s)

- Error: 1018 SQLSTATE: HY000 (ER_CANT_READ_DIR)

  Message: Can't read dir of '%s' (errno: %d - %s)

- Error: 1019 SQLSTATE: HY000 (ER_CANT_SET_WD)

  Message: Can't change dir to '%s' (errno: %d - %s)

- Error: 1020 SQLSTATE: HY000 (ER_CHECKREAD)

  Message: Record has changed since last read in table '%s'

- Error: 1021 SQLSTATE: HY000 (ER_DISK_FULL)

  Message: Disk full (%s); waiting for someone to free some space... (errno: %d - %s)

- Error: 1022 SQLSTATE: 23000 (ER_DUP_KEY)

  Message: Can't write; duplicate key in table '%s'

- Error: 1023 SQLSTATE: HY000 (ER_ERROR_ON_CLOSE)

  Message: Error on close of '%s' (errno: %d - %s)

- Error: 1024 SQLSTATE: HY000 (ER_ERROR_ON_READ)

  Message: Error reading file '%s' (errno: %d - %s)

- Error: 1025 SQLSTATE: HY000 (ER_ERROR_ON_RENAME)

  Message: Error on rename of '%s' to '%s' (errno: %d - %s)

- Error: 1026 SQLSTATE: HY000 (ER_ERROR_ON_WRITE)

  Message: Error writing file '%s' (errno: %d - %s)

- Error: 1027 SQLSTATE: HY000 (ER_FILE_USED)

  Message: '%s' is locked against change

- Error: 1028 SQLSTATE: HY000 (ER_FILSORT_ABORT)

  Message: Sort aborted

- Error: 1029 SQLSTATE: HY000 (ER_FORM_NOT_FOUND)

  Message: View '%s' doesn't exist for '%s'

- Error: 1030 SQLSTATE: HY000 (ER_GET_ERRNO)

  Message: Got error %d from storage engine

  Check the %d value to see what the OS error means. For example, 28 indicates that you have run out of disk space.

- Error: 1031 SQLSTATE: HY000 (ER_ILLEGAL_HA)

  Message: Table storage engine for '%s' doesn't have this option

- Error: 1032 SQLSTATE: HY000 (ER_KEY_NOT_FOUND)

Message: Can't find record in '%s'

- Error: `1033` SQLSTATE: `HY000` (`ER_NOT_FORM_FILE`)

  Message: Incorrect information in file: '%s'

- Error: `1034` SQLSTATE: `HY000` (`ER_NOT_KEYFILE`)

  Message: Incorrect key file for table '%s'; try to repair it

- Error: `1035` SQLSTATE: `HY000` (`ER_OLD_KEYFILE`)

  Message: Old key file for table '%s'; repair it!

- Error: `1036` SQLSTATE: `HY000` (`ER_OPEN_AS_READONLY`)

  Message: Table '%s' is read only

- Error: `1037` SQLSTATE: `HY001` (`ER_OUTOFMEMORY`)

  Message: Out of memory; restart server and try again (needed %d bytes)

- Error: `1038` SQLSTATE: `HY001` (`ER_OUT_OF_SORTMEMORY`)

  Message: Out of sort memory, consider increasing server sort buffer size

- Error: `1039` SQLSTATE: `HY000` (`ER_UNEXPECTED_EOF`)

  Message: Unexpected EOF found when reading file '%s' (errno: %d - %s)

- Error: `1040` SQLSTATE: `08004` (`ER_CON_COUNT_ERROR`)

  Message: Too many connections

- Error: `1041` SQLSTATE: `HY000` (`ER_OUT_OF_RESOURCES`)

  Message: Out of memory; check if mysqld or some other process uses all available memory; if not, you may have to use 'ulimit' to allow mysqld to use more memory or you can add more swap space

- Error: `1042` SQLSTATE: `08S01` (`ER_BAD_HOST_ERROR`)

  Message: Can't get hostname for your address

- Error: `1043` SQLSTATE: `08S01` (`ER_HANDSHAKE_ERROR`)

  Message: Bad handshake

- Error: `1044` SQLSTATE: `42000` (`ER_DBACCESS_DENIED_ERROR`)

  Message: Access denied for user '%s'@'%s' to database '%s'

- Error: `1045` SQLSTATE: `28000` (`ER_ACCESS_DENIED_ERROR`)

  Message: Access denied for user '%s'@'%s' (using password: %s)

- Error: `1046` SQLSTATE: `3D000` (`ER_NO_DB_ERROR`)

  Message: No database selected

- Error: `1047` SQLSTATE: `08S01` (`ER_UNKNOWN_COM_ERROR`)

Message: Unknown command

- Error: 1048 SQLSTATE: 23000 (ER_BAD_NULL_ERROR)

  Message: Column '%s' cannot be null

- Error: 1049 SQLSTATE: 42000 (ER_BAD_DB_ERROR)

  Message: Unknown database '%s'

- Error: 1050 SQLSTATE: 42S01 (ER_TABLE_EXISTS_ERROR)

  Message: Table '%s' already exists

- Error: 1051 SQLSTATE: 42S02 (ER_BAD_TABLE_ERROR)

  Message: Unknown table '%s'

- Error: 1052 SQLSTATE: 23000 (ER_NON_UNIQ_ERROR)

  Message: Column '%s' in %s is ambiguous

  ```
  %s = column name
  %s = location of column (for example, "field list")
  ```

  Likely cause: A column appears in a query without appropriate qualification, such as in a select list or ON clause.

  Examples:

  ```
  mysql> SELECT i FROM t INNER JOIN t AS t2;
  ERROR 1052 (23000): Column 'i' in field list is ambiguous

  mysql> SELECT * FROM t LEFT JOIN t AS t2 ON i = i;
  ERROR 1052 (23000): Column 'i' in on clause is ambiguous
  ```

  Resolution:

  - Qualify the column with the appropriate table name:

    ```
    mysql> SELECT t2.i FROM t INNER JOIN t AS t2;
    ```

  - Modify the query to avoid the need for qualification:

    ```
    mysql> SELECT * FROM t LEFT JOIN t AS t2 USING (i);
    ```

- Error: 1053 SQLSTATE: 08S01 (ER_SERVER_SHUTDOWN)

  Message: Server shutdown in progress

- Error: 1054 SQLSTATE: 42S22 (ER_BAD_FIELD_ERROR)

  Message: Unknown column '%s' in '%s'

- Error: 1055 SQLSTATE: 42000 (ER_WRONG_FIELD_WITH_GROUP)

  Message: '%s' isn't in GROUP BY

- Error: `1056` SQLSTATE: `42000` (`ER_WRONG_GROUP_FIELD`)

  Message: Can't group on '%s'
- Error: `1057` SQLSTATE: `42000` (`ER_WRONG_SUM_SELECT`)

  Message: Statement has sum functions and columns in same statement
- Error: `1058` SQLSTATE: `21S01` (`ER_WRONG_VALUE_COUNT`)

  Message: Column count doesn't match value count
- Error: `1059` SQLSTATE: `42000` (`ER_TOO_LONG_IDENT`)

  Message: Identifier name '%s' is too long
- Error: `1060` SQLSTATE: `42S21` (`ER_DUP_FIELDNAME`)

  Message: Duplicate column name '%s'
- Error: `1061` SQLSTATE: `42000` (`ER_DUP_KEYNAME`)

  Message: Duplicate key name '%s'
- Error: `1062` SQLSTATE: `23000` (`ER_DUP_ENTRY`)

  Message: Duplicate entry '%s' for key %d

  The message returned with this error uses the format string for `ER_DUP_ENTRY_WITH_KEY_NAME`.
- Error: `1063` SQLSTATE: `42000` (`ER_WRONG_FIELD_SPEC`)

  Message: Incorrect column specifier for column '%s'
- Error: `1064` SQLSTATE: `42000` (`ER_PARSE_ERROR`)

  Message: %s near '%s' at line %d
- Error: `1065` SQLSTATE: `42000` (`ER_EMPTY_QUERY`)

  Message: Query was empty
- Error: `1066` SQLSTATE: `42000` (`ER_NONUNIQ_TABLE`)

  Message: Not unique table/alias: '%s'
- Error: `1067` SQLSTATE: `42000` (`ER_INVALID_DEFAULT`)

  Message: Invalid default value for '%s'
- Error: `1068` SQLSTATE: `42000` (`ER_MULTIPLE_PRI_KEY`)

  Message: Multiple primary key defined
- Error: `1069` SQLSTATE: `42000` (`ER_TOO_MANY_KEYS`)

  Message: Too many keys specified; max %d keys allowed
- Error: `1070` SQLSTATE: `42000` (`ER_TOO_MANY_KEY_PARTS`)

  Message: Too many key parts specified; max %d parts allowed

- Error: 1071 SQLSTATE: 42000 (ER_TOO_LONG_KEY)

  Message: Specified key was too long; max key length is %d bytes
- Error: 1072 SQLSTATE: 42000 (ER_KEY_COLUMN_DOES_NOT_EXITS)

  Message: Key column '%s' doesn't exist in table
- Error: 1073 SQLSTATE: 42000 (ER_BLOB_USED_AS_KEY)

  Message: BLOB column '%s' can't be used in key specification with the used table type
- Error: 1074 SQLSTATE: 42000 (ER_TOO_BIG_FIELDLENGTH)

  Message: Column length too big for column '%s' (max = %lu); use BLOB or TEXT instead
- Error: 1075 SQLSTATE: 42000 (ER_WRONG_AUTO_KEY)

  Message: Incorrect table definition; there can be only one auto column and it must be defined as a key
- Error: 1076 SQLSTATE: HY000 (ER_READY)

  Message: %s: ready for connections. Version: '%s' socket: '%s' port: %d
- Error: 1077 SQLSTATE: HY000 (ER_NORMAL_SHUTDOWN)

  Message: %s: Normal shutdown
- Error: 1078 SQLSTATE: HY000 (ER_GOT_SIGNAL)

  Message: %s: Got signal %d. Aborting!
- Error: 1079 SQLSTATE: HY000 (ER_SHUTDOWN_COMPLETE)

  Message: %s: Shutdown complete
- Error: 1080 SQLSTATE: 08S01 (ER_FORCING_CLOSE)

  Message: %s: Forcing close of thread %ld user: '%s'
- Error: 1081 SQLSTATE: 08S01 (ER_IPSOCK_ERROR)

  Message: Can't create IP socket
- Error: 1082 SQLSTATE: 42S12 (ER_NO_SUCH_INDEX)

  Message: Table '%s' has no index like the one used in CREATE INDEX; recreate the table
- Error: 1083 SQLSTATE: 42000 (ER_WRONG_FIELD_TERMINATORS)

  Message: Field separator argument is not what is expected; check the manual
- Error: 1084 SQLSTATE: 42000 (ER_BLOBS_AND_NO_TERMINATED)

  Message: You can't use fixed rowlength with BLOBs; please use 'fields terminated by'
- Error: 1085 SQLSTATE: HY000 (ER_TEXTFILE_NOT_READABLE)

  Message: The file '%s' must be in the database directory or be readable by all

- Error: 1086 SQLSTATE: HY000 (ER_FILE_EXISTS_ERROR)

  Message: File '%s' already exists

- Error: 1087 SQLSTATE: HY000 (ER_LOAD_INFO)

  Message: Records: %ld Deleted: %ld Skipped: %ld Warnings: %ld

- Error: 1088 SQLSTATE: HY000 (ER_ALTER_INFO)

  Message: Records: %ld Duplicates: %ld

- Error: 1089 SQLSTATE: HY000 (ER_WRONG_SUB_KEY)

  Message: Incorrect prefix key; the used key part isn't a string, the used length is longer than the key part, or the storage engine doesn't support unique prefix keys

- Error: 1090 SQLSTATE: 42000 (ER_CANT_REMOVE_ALL_FIELDS)

  Message: You can't delete all columns with ALTER TABLE; use DROP TABLE instead

- Error: 1091 SQLSTATE: 42000 (ER_CANT_DROP_FIELD_OR_KEY)

  Message: Can't DROP '%s'; check that column/key exists

- Error: 1092 SQLSTATE: HY000 (ER_INSERT_INFO)

  Message: Records: %ld Duplicates: %ld Warnings: %ld

- Error: 1093 SQLSTATE: HY000 (ER_UPDATE_TABLE_USED)

  Message: You can't specify target table '%s' for update in FROM clause

- Error: 1094 SQLSTATE: HY000 (ER_NO_SUCH_THREAD)

  Message: Unknown thread id: %lu

- Error: 1095 SQLSTATE: HY000 (ER_KILL_DENIED_ERROR)

  Message: You are not owner of thread %lu

- Error: 1096 SQLSTATE: HY000 (ER_NO_TABLES_USED)

  Message: No tables used

- Error: 1097 SQLSTATE: HY000 (ER_TOO_BIG_SET)

  Message: Too many strings for column %s and SET

- Error: 1098 SQLSTATE: HY000 (ER_NO_UNIQUE_LOGFILE)

  Message: Can't generate a unique log-filename %s.(1-999)

- Error: 1099 SQLSTATE: HY000 (ER_TABLE_NOT_LOCKED_FOR_WRITE)

  Message: Table '%s' was locked with a READ lock and can't be updated

- Error: 1100 SQLSTATE: HY000 (ER_TABLE_NOT_LOCKED)

  Message: Table '%s' was not locked with LOCK TABLES

- Error: 1101 SQLSTATE: 42000 (ER_BLOB_CANT_HAVE_DEFAULT)

  Message: BLOB/TEXT column '%s' can't have a default value
- Error: 1102 SQLSTATE: 42000 (ER_WRONG_DB_NAME)

  Message: Incorrect database name '%s'
- Error: 1103 SQLSTATE: 42000 (ER_WRONG_TABLE_NAME)

  Message: Incorrect table name '%s'
- Error: 1104 SQLSTATE: 42000 (ER_TOO_BIG_SELECT)

  Message: The SELECT would examine more than MAX_JOIN_SIZE rows; check your WHERE and use SET SQL_BIG_SELECTS=1 or SET MAX_JOIN_SIZE=# if the SELECT is okay
- Error: 1105 SQLSTATE: HY000 (ER_UNKNOWN_ERROR)

  Message: Unknown error
- Error: 1106 SQLSTATE: 42000 (ER_UNKNOWN_PROCEDURE)

  Message: Unknown procedure '%s'
- Error: 1107 SQLSTATE: 42000 (ER_WRONG_PARAMCOUNT_TO_PROCEDURE)

  Message: Incorrect parameter count to procedure '%s'
- Error: 1108 SQLSTATE: HY000 (ER_WRONG_PARAMETERS_TO_PROCEDURE)

  Message: Incorrect parameters to procedure '%s'
- Error: 1109 SQLSTATE: 42S02 (ER_UNKNOWN_TABLE)

  Message: Unknown table '%s' in %s
- Error: 1110 SQLSTATE: 42000 (ER_FIELD_SPECIFIED_TWICE)

  Message: Column '%s' specified twice
- Error: 1111 SQLSTATE: HY000 (ER_INVALID_GROUP_FUNC_USE)

  Message: Invalid use of group function
- Error: 1112 SQLSTATE: 42000 (ER_UNSUPPORTED_EXTENSION)

  Message: Table '%s' uses an extension that doesn't exist in this MySQL version
- Error: 1113 SQLSTATE: 42000 (ER_TABLE_MUST_HAVE_COLUMNS)

  Message: A table must have at least 1 column
- Error: 1114 SQLSTATE: HY000 (ER_RECORD_FILE_FULL)

  Message: The table '%s' is full
- Error: 1115 SQLSTATE: 42000 (ER_UNKNOWN_CHARACTER_SET)

  Message: Unknown character set: '%s'

- Error: `1116` SQLSTATE: `HY000` (`ER_TOO_MANY_TABLES`)

  Message: Too many tables; MySQL can only use %d tables in a join

- Error: `1117` SQLSTATE: `HY000` (`ER_TOO_MANY_FIELDS`)

  Message: Too many columns

- Error: `1118` SQLSTATE: `42000` (`ER_TOO_BIG_ROWSIZE`)

  Message: Row size too large. The maximum row size for the used table type, not counting BLOBs, is %ld. This includes storage overhead, check the manual. You have to change some columns to TEXT or BLOBs

- Error: `1119` SQLSTATE: `HY000` (`ER_STACK_OVERRUN`)

  Message: Thread stack overrun: Used: %ld of a %ld stack. Use 'mysqld --thread_stack=#' to specify a bigger stack if needed

- Error: `1120` SQLSTATE: `42000` (`ER_WRONG_OUTER_JOIN`)

  Message: Cross dependency found in OUTER JOIN; examine your ON conditions

- Error: `1121` SQLSTATE: `42000` (`ER_NULL_COLUMN_IN_INDEX`)

  Message: Table handler doesn't support NULL in given index. Please change column '%s' to be NOT NULL or use another handler

- Error: `1122` SQLSTATE: `HY000` (`ER_CANT_FIND_UDF`)

  Message: Can't load function '%s'

- Error: `1123` SQLSTATE: `HY000` (`ER_CANT_INITIALIZE_UDF`)

  Message: Can't initialize function '%s'; %s

- Error: `1124` SQLSTATE: `HY000` (`ER_UDF_NO_PATHS`)

  Message: No paths allowed for shared library

- Error: `1125` SQLSTATE: `HY000` (`ER_UDF_EXISTS`)

  Message: Function '%s' already exists

- Error: `1126` SQLSTATE: `HY000` (`ER_CANT_OPEN_LIBRARY`)

  Message: Can't open shared library '%s' (errno: %d %s)

- Error: `1127` SQLSTATE: `HY000` (`ER_CANT_FIND_DL_ENTRY`)

  Message: Can't find symbol '%s' in library

- Error: `1128` SQLSTATE: `HY000` (`ER_FUNCTION_NOT_DEFINED`)

  Message: Function '%s' is not defined

- Error: `1129` SQLSTATE: `HY000` (`ER_HOST_IS_BLOCKED`)

  Message: Host '%s' is blocked because of many connection errors; unblock with 'mysqladmin flush-hosts'

- Error: 1130 SQLSTATE: HY000 (ER_HOST_NOT_PRIVILEGED)

  Message: Host '%s' is not allowed to connect to this MySQL server

- Error: 1131 SQLSTATE: 42000 (ER_PASSWORD_ANONYMOUS_USER)

  Message: You are using MySQL as an anonymous user and anonymous users are not allowed to change passwords

- Error: 1132 SQLSTATE: 42000 (ER_PASSWORD_NOT_ALLOWED)

  Message: You must have privileges to update tables in the mysql database to be able to change passwords for others

- Error: 1133 SQLSTATE: 42000 (ER_PASSWORD_NO_MATCH)

  Message: Can't find any matching row in the user table

- Error: 1134 SQLSTATE: HY000 (ER_UPDATE_INFO)

  Message: Rows matched: %ld Changed: %ld Warnings: %ld

- Error: 1135 SQLSTATE: HY000 (ER_CANT_CREATE_THREAD)

  Message: Can't create a new thread (errno %d); if you are not out of available memory, you can consult the manual for a possible OS-dependent bug

- Error: 1136 SQLSTATE: 21S01 (ER_WRONG_VALUE_COUNT_ON_ROW)

  Message: Column count doesn't match value count at row %ld

- Error: 1137 SQLSTATE: HY000 (ER_CANT_REOPEN_TABLE)

  Message: Can't reopen table: '%s'

- Error: 1138 SQLSTATE: 22004 (ER_INVALID_USE_OF_NULL)

  Message: Invalid use of NULL value

- Error: 1139 SQLSTATE: 42000 (ER_REGEXP_ERROR)

  Message: Got error '%s' from regexp

- Error: 1140 SQLSTATE: 42000 (ER_MIX_OF_GROUP_FUNC_AND_FIELDS)

  Message: Mixing of GROUP columns (MIN(),MAX(),COUNT(),...) with no GROUP columns is illegal if there is no GROUP BY clause

- Error: 1141 SQLSTATE: 42000 (ER_NONEXISTING_GRANT)

  Message: There is no such grant defined for user '%s' on host '%s'

- Error: 1142 SQLSTATE: 42000 (ER_TABLEACCESS_DENIED_ERROR)

  Message: %s command denied to user '%s'@'%s' for table '%s'

- Error: 1143 SQLSTATE: 42000 (ER_COLUMNACCESS_DENIED_ERROR)

  Message: %s command denied to user '%s'@'%s' for column '%s' in table '%s'

- Error: `1144` SQLSTATE: `42000` (`ER_ILLEGAL_GRANT_FOR_TABLE`)

  Message: Illegal GRANT/REVOKE command; please consult the manual to see which privileges can be used

- Error: `1145` SQLSTATE: `42000` (`ER_GRANT_WRONG_HOST_OR_USER`)

  Message: The host or user argument to GRANT is too long

- Error: `1146` SQLSTATE: `42S02` (`ER_NO_SUCH_TABLE`)

  Message: Table '%s.%s' doesn't exist

- Error: `1147` SQLSTATE: `42000` (`ER_NONEXISTING_TABLE_GRANT`)

  Message: There is no such grant defined for user '%s' on host '%s' on table '%s'

- Error: `1148` SQLSTATE: `42000` (`ER_NOT_ALLOWED_COMMAND`)

  Message: The used command is not allowed with this MySQL version

- Error: `1149` SQLSTATE: `42000` (`ER_SYNTAX_ERROR`)

  Message: You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use

- Error: `1150` SQLSTATE: `HY000` (`ER_UNUSED1`)

  Message: Delayed insert thread couldn't get requested lock for table %s

- Error: `1151` SQLSTATE: `HY000` (`ER_UNUSED2`)

  Message: Too many delayed threads in use

- Error: `1152` SQLSTATE: `08S01` (`ER_ABORTING_CONNECTION`)

  Message: Aborted connection %ld to db: '%s' user: '%s' (%s)

- Error: `1153` SQLSTATE: `08S01` (`ER_NET_PACKET_TOO_LARGE`)

  Message: Got a packet bigger than 'max_allowed_packet' bytes

- Error: `1154` SQLSTATE: `08S01` (`ER_NET_READ_ERROR_FROM_PIPE`)

  Message: Got a read error from the connection pipe

- Error: `1155` SQLSTATE: `08S01` (`ER_NET_FCNTL_ERROR`)

  Message: Got an error from fcntl()

- Error: `1156` SQLSTATE: `08S01` (`ER_NET_PACKETS_OUT_OF_ORDER`)

  Message: Got packets out of order

- Error: `1157` SQLSTATE: `08S01` (`ER_NET_UNCOMPRESS_ERROR`)

  Message: Couldn't uncompress communication packet

- Error: `1158` SQLSTATE: `08S01` (`ER_NET_READ_ERROR`)

Message: Got an error reading communication packets

- Error: 1159 SQLSTATE: 08S01 (ER_NET_READ_INTERRUPTED)

  Message: Got timeout reading communication packets

- Error: 1160 SQLSTATE: 08S01 (ER_NET_ERROR_ON_WRITE)

  Message: Got an error writing communication packets

- Error: 1161 SQLSTATE: 08S01 (ER_NET_WRITE_INTERRUPTED)

  Message: Got timeout writing communication packets

- Error: 1162 SQLSTATE: 42000 (ER_TOO_LONG_STRING)

  Message: Result string is longer than 'max_allowed_packet' bytes

- Error: 1163 SQLSTATE: 42000 (ER_TABLE_CANT_HANDLE_BLOB)

  Message: The used table type doesn't support BLOB/TEXT columns

- Error: 1164 SQLSTATE: 42000 (ER_TABLE_CANT_HANDLE_AUTO_INCREMENT)

  Message: The used table type doesn't support AUTO_INCREMENT columns

- Error: 1165 SQLSTATE: HY000 (ER_UNUSED3)

  Message: INSERT DELAYED can't be used with table '%s' because it is locked with LOCK TABLES

- Error: 1166 SQLSTATE: 42000 (ER_WRONG_COLUMN_NAME)

  Message: Incorrect column name '%s'

- Error: 1167 SQLSTATE: 42000 (ER_WRONG_KEY_COLUMN)

  Message: The used storage engine can't index column '%s'

- Error: 1168 SQLSTATE: HY000 (ER_WRONG_MRG_TABLE)

  Message: Unable to open underlying table which is differently defined or of non-MyISAM type or doesn't exist

- Error: 1169 SQLSTATE: 23000 (ER_DUP_UNIQUE)

  Message: Can't write, because of unique constraint, to table '%s'

- Error: 1170 SQLSTATE: 42000 (ER_BLOB_KEY_WITHOUT_LENGTH)

  Message: BLOB/TEXT column '%s' used in key specification without a key length

- Error: 1171 SQLSTATE: 42000 (ER_PRIMARY_CANT_HAVE_NULL)

  Message: All parts of a PRIMARY KEY must be NOT NULL; if you need NULL in a key, use UNIQUE instead

- Error: 1172 SQLSTATE: 42000 (ER_TOO_MANY_ROWS)

  Message: Result consisted of more than one row

- Error: 1173 SQLSTATE: 42000 (ER_REQUIRES_PRIMARY_KEY)

  Message: This table type requires a primary key

- Error: 1174 SQLSTATE: HY000 (ER_NO_RAID_COMPILED)

  Message: This version of MySQL is not compiled with RAID support

- Error: 1175 SQLSTATE: HY000 (ER_UPDATE_WITHOUT_KEY_IN_SAFE_MODE)

  Message: You are using safe update mode and you tried to update a table without a WHERE that uses a KEY column

- Error: 1176 SQLSTATE: 42000 (ER_KEY_DOES_NOT_EXITS)

  Message: Key '%s' doesn't exist in table '%s'

- Error: 1177 SQLSTATE: 42000 (ER_CHECK_NO_SUCH_TABLE)

  Message: Can't open table

- Error: 1178 SQLSTATE: 42000 (ER_CHECK_NOT_IMPLEMENTED)

  Message: The storage engine for the table doesn't support %s

- Error: 1179 SQLSTATE: 25000 (ER_CANT_DO_THIS_DURING_AN_TRANSACTION)

  Message: You are not allowed to execute this command in a transaction

- Error: 1180 SQLSTATE: HY000 (ER_ERROR_DURING_COMMIT)

  Message: Got error %d during COMMIT

- Error: 1181 SQLSTATE: HY000 (ER_ERROR_DURING_ROLLBACK)

  Message: Got error %d during ROLLBACK

- Error: 1182 SQLSTATE: HY000 (ER_ERROR_DURING_FLUSH_LOGS)

  Message: Got error %d during FLUSH_LOGS

- Error: 1183 SQLSTATE: HY000 (ER_ERROR_DURING_CHECKPOINT)

  Message: Got error %d during CHECKPOINT

- Error: 1184 SQLSTATE: 08S01 (ER_NEW_ABORTING_CONNECTION)

  Message: Aborted connection %ld to db: '%s' user: '%s' host: '%s' (%s)

- Error: 1185 SQLSTATE: HY000 (ER_DUMP_NOT_IMPLEMENTED)

  Message: The storage engine for the table does not support binary table dump

- Error: 1186 SQLSTATE: HY000 (ER_FLUSH_MASTER_BINLOG_CLOSED)

  Message: Binlog closed, cannot RESET MASTER

- Error: 1187 SQLSTATE: HY000 (ER_INDEX_REBUILD)

  Message: Failed rebuilding the index of dumped table '%s'

- Error: `1188` SQLSTATE: `HY000` (`ER_MASTER`)

  Message: Error from master: '%s'

- Error: `1189` SQLSTATE: `08S01` (`ER_MASTER_NET_READ`)

  Message: Net error reading from master

- Error: `1190` SQLSTATE: `08S01` (`ER_MASTER_NET_WRITE`)

  Message: Net error writing to master

- Error: `1191` SQLSTATE: `HY000` (`ER_FT_MATCHING_KEY_NOT_FOUND`)

  Message: Can't find FULLTEXT index matching the column list

- Error: `1192` SQLSTATE: `HY000` (`ER_LOCK_OR_ACTIVE_TRANSACTION`)

  Message: Can't execute the given command because you have active locked tables or an active transaction

- Error: `1193` SQLSTATE: `HY000` (`ER_UNKNOWN_SYSTEM_VARIABLE`)

  Message: Unknown system variable '%s'

- Error: `1194` SQLSTATE: `HY000` (`ER_CRASHED_ON_USAGE`)

  Message: Table '%s' is marked as crashed and should be repaired

- Error: `1195` SQLSTATE: `HY000` (`ER_CRASHED_ON_REPAIR`)

  Message: Table '%s' is marked as crashed and last (automatic?) repair failed

- Error: `1196` SQLSTATE: `HY000` (`ER_WARNING_NOT_COMPLETE_ROLLBACK`)

  Message: Some non-transactional changed tables couldn't be rolled back

- Error: `1197` SQLSTATE: `HY000` (`ER_TRANS_CACHE_FULL`)

  Message: Multi-statement transaction required more than 'max_binlog_cache_size' bytes of storage; increase this mysqld variable and try again

- Error: `1198` SQLSTATE: `HY000` (`ER_SLAVE_MUST_STOP`)

  Message: This operation cannot be performed with a running slave; run STOP SLAVE first

- Error: `1199` SQLSTATE: `HY000` (`ER_SLAVE_NOT_RUNNING`)

  Message: This operation requires a running slave; configure slave and do START SLAVE

- Error: `1200` SQLSTATE: `HY000` (`ER_BAD_SLAVE`)

  Message: The server is not configured as slave; fix in config file or with CHANGE MASTER TO

- Error: `1201` SQLSTATE: `HY000` (`ER_MASTER_INFO`)

  Message: Could not initialize master info structure; more error messages can be found in the MySQL error log

- Error: `1202` SQLSTATE: `HY000` (`ER_SLAVE_THREAD`)

Message: Could not create slave thread; check system resources

- Error: `1203` SQLSTATE: `42000` (`ER_TOO_MANY_USER_CONNECTIONS`)

  Message: User %s already has more than 'max_user_connections' active connections

- Error: `1204` SQLSTATE: `HY000` (`ER_SET_CONSTANTS_ONLY`)

  Message: You may only use constant expressions with SET

- Error: `1205` SQLSTATE: `HY000` (`ER_LOCK_WAIT_TIMEOUT`)

  Message: Lock wait timeout exceeded; try restarting transaction

- Error: `1206` SQLSTATE: `HY000` (`ER_LOCK_TABLE_FULL`)

  Message: The total number of locks exceeds the lock table size

- Error: `1207` SQLSTATE: `25000` (`ER_READ_ONLY_TRANSACTION`)

  Message: Update locks cannot be acquired during a READ UNCOMMITTED transaction

- Error: `1208` SQLSTATE: `HY000` (`ER_DROP_DB_WITH_READ_LOCK`)

  Message: DROP DATABASE not allowed while thread is holding global read lock

- Error: `1209` SQLSTATE: `HY000` (`ER_CREATE_DB_WITH_READ_LOCK`)

  Message: CREATE DATABASE not allowed while thread is holding global read lock

- Error: `1210` SQLSTATE: `HY000` (`ER_WRONG_ARGUMENTS`)

  Message: Incorrect arguments to %s

- Error: `1211` SQLSTATE: `42000` (`ER_NO_PERMISSION_TO_CREATE_USER`)

  Message: '%s'@'%s' is not allowed to create new users

- Error: `1212` SQLSTATE: `HY000` (`ER_UNION_TABLES_IN_DIFFERENT_DIR`)

  Message: Incorrect table definition; all MERGE tables must be in the same database

- Error: `1213` SQLSTATE: `40001` (`ER_LOCK_DEADLOCK`)

  Message: Deadlock found when trying to get lock; try restarting transaction

- Error: `1214` SQLSTATE: `HY000` (`ER_TABLE_CANT_HANDLE_FT`)

  Message: The used table type doesn't support FULLTEXT indexes

- Error: `1215` SQLSTATE: `HY000` (`ER_CANNOT_ADD_FOREIGN`)

  Message: Cannot add foreign key constraint

- Error: `1216` SQLSTATE: `23000` (`ER_NO_REFERENCED_ROW`)

  Message: Cannot add or update a child row: a foreign key constraint fails

- Error: `1217` SQLSTATE: `23000` (`ER_ROW_IS_REFERENCED`)

Message: Cannot delete or update a parent row: a foreign key constraint fails

- Error: 1218 SQLSTATE: 08S01 (ER_CONNECT_TO_MASTER)

  Message: Error connecting to master: %s

- Error: 1219 SQLSTATE: HY000 (ER_QUERY_ON_MASTER)

  Message: Error running query on master: %s

- Error: 1220 SQLSTATE: HY000 (ER_ERROR_WHEN_EXECUTING_COMMAND)

  Message: Error when executing command %s: %s

- Error: 1221 SQLSTATE: HY000 (ER_WRONG_USAGE)

  Message: Incorrect usage of %s and %s

- Error: 1222 SQLSTATE: 21000 (ER_WRONG_NUMBER_OF_COLUMNS_IN_SELECT)

  Message: The used SELECT statements have a different number of columns

- Error: 1223 SQLSTATE: HY000 (ER_CANT_UPDATE_WITH_READLOCK)

  Message: Can't execute the query because you have a conflicting read lock

- Error: 1224 SQLSTATE: HY000 (ER_MIXING_NOT_ALLOWED)

  Message: Mixing of transactional and non-transactional tables is disabled

- Error: 1225 SQLSTATE: HY000 (ER_DUP_ARGUMENT)

  Message: Option '%s' used twice in statement

- Error: 1226 SQLSTATE: 42000 (ER_USER_LIMIT_REACHED)

  Message: User '%s' has exceeded the '%s' resource (current value: %ld)

- Error: 1227 SQLSTATE: 42000 (ER_SPECIFIC_ACCESS_DENIED_ERROR)

  Message: Access denied; you need (at least one of) the %s privilege(s) for this operation

- Error: 1228 SQLSTATE: HY000 (ER_LOCAL_VARIABLE)

  Message: Variable '%s' is a SESSION variable and can't be used with SET GLOBAL

- Error: 1229 SQLSTATE: HY000 (ER_GLOBAL_VARIABLE)

  Message: Variable '%s' is a GLOBAL variable and should be set with SET GLOBAL

- Error: 1230 SQLSTATE: 42000 (ER_NO_DEFAULT)

  Message: Variable '%s' doesn't have a default value

- Error: 1231 SQLSTATE: 42000 (ER_WRONG_VALUE_FOR_VAR)

  Message: Variable '%s' can't be set to the value of '%s'

- Error: 1232 SQLSTATE: 42000 (ER_WRONG_TYPE_FOR_VAR)

Message: Incorrect argument type to variable '%s'

- Error: 1233 SQLSTATE: HY000 (ER_VAR_CANT_BE_READ)

  Message: Variable '%s' can only be set, not read

- Error: 1234 SQLSTATE: 42000 (ER_CANT_USE_OPTION_HERE)

  Message: Incorrect usage/placement of '%s'

- Error: 1235 SQLSTATE: 42000 (ER_NOT_SUPPORTED_YET)

  Message: This version of MySQL doesn't yet support '%s'

- Error: 1236 SQLSTATE: HY000 (ER_MASTER_FATAL_ERROR_READING_BINLOG)

  Message: Got fatal error %d from master when reading data from binary log: '%s'

- Error: 1237 SQLSTATE: HY000 (ER_SLAVE_IGNORED_TABLE)

  Message: Slave SQL thread ignored the query because of replicate-*-table rules

- Error: 1238 SQLSTATE: HY000 (ER_INCORRECT_GLOBAL_LOCAL_VAR)

  Message: Variable '%s' is a %s variable

- Error: 1239 SQLSTATE: 42000 (ER_WRONG_FK_DEF)

  Message: Incorrect foreign key definition for '%s': %s

- Error: 1240 SQLSTATE: HY000 (ER_KEY_REF_DO_NOT_MATCH_TABLE_REF)

  Message: Key reference and table reference don't match

- Error: 1241 SQLSTATE: 21000 (ER_OPERAND_COLUMNS)

  Message: Operand should contain %d column(s)

- Error: 1242 SQLSTATE: 21000 (ER_SUBQUERY_NO_1_ROW)

  Message: Subquery returns more than 1 row

- Error: 1243 SQLSTATE: HY000 (ER_UNKNOWN_STMT_HANDLER)

  Message: Unknown prepared statement handler (%.*s) given to %s

- Error: 1244 SQLSTATE: HY000 (ER_CORRUPT_HELP_DB)

  Message: Help database is corrupt or does not exist

- Error: 1245 SQLSTATE: HY000 (ER_CYCLIC_REFERENCE)

  Message: Cyclic reference on subqueries

- Error: 1246 SQLSTATE: HY000 (ER_AUTO_CONVERT)

  Message: Converting column '%s' from %s to %s

- Error: 1247 SQLSTATE: 42S22 (ER_ILLEGAL_REFERENCE)

Message: Reference '%s' not supported (%s)

- Error: 1248 SQLSTATE: 42000 (ER_DERIVED_MUST_HAVE_ALIAS)

  Message: Every derived table must have its own alias

- Error: 1249 SQLSTATE: 01000 (ER_SELECT_REDUCED)

  Message: Select %u was reduced during optimization

- Error: 1250 SQLSTATE: 42000 (ER_TABLENAME_NOT_ALLOWED_HERE)

  Message: Table '%s' from one of the SELECTs cannot be used in %s

- Error: 1251 SQLSTATE: 08004 (ER_NOT_SUPPORTED_AUTH_MODE)

  Message: Client does not support authentication protocol requested by server; consider upgrading MySQL client

- Error: 1252 SQLSTATE: 42000 (ER_SPATIAL_CANT_HAVE_NULL)

  Message: All parts of a SPATIAL index must be NOT NULL

- Error: 1253 SQLSTATE: 42000 (ER_COLLATION_CHARSET_MISMATCH)

  Message: COLLATION '%s' is not valid for CHARACTER SET '%s'

- Error: 1254 SQLSTATE: HY000 (ER_SLAVE_WAS_RUNNING)

  Message: Slave is already running

- Error: 1255 SQLSTATE: HY000 (ER_SLAVE_WAS_NOT_RUNNING)

  Message: Slave already has been stopped

- Error: 1256 SQLSTATE: HY000 (ER_TOO_BIG_FOR_UNCOMPRESS)

  Message: Uncompressed data size too large; the maximum size is %d (probably, length of uncompressed data was corrupted)

- Error: 1257 SQLSTATE: HY000 (ER_ZLIB_Z_MEM_ERROR)

  Message: ZLIB: Not enough memory

- Error: 1258 SQLSTATE: HY000 (ER_ZLIB_Z_BUF_ERROR)

  Message: ZLIB: Not enough room in the output buffer (probably, length of uncompressed data was corrupted)

- Error: 1259 SQLSTATE: HY000 (ER_ZLIB_Z_DATA_ERROR)

  Message: ZLIB: Input data corrupted

- Error: 1260 SQLSTATE: HY000 (ER_CUT_VALUE_GROUP_CONCAT)

  Message: Row %u was cut by GROUP_CONCAT()

- Error: 1261 SQLSTATE: 01000 (ER_WARN_TOO_FEW_RECORDS)

Message: Row %ld doesn't contain data for all columns

- Error: 1262 SQLSTATE: 01000 (ER_WARN_TOO_MANY_RECORDS)

  Message: Row %ld was truncated; it contained more data than there were input columns

- Error: 1263 SQLSTATE: 22004 (ER_WARN_NULL_TO_NOTNULL)

  Message: Column set to default value; NULL supplied to NOT NULL column '%s' at row %ld

- Error: 1264 SQLSTATE: 22003 (ER_WARN_DATA_OUT_OF_RANGE)

  Message: Out of range value for column '%s' at row %ld

- Error: 1265 SQLSTATE: 01000 (WARN_DATA_TRUNCATED)

  Message: Data truncated for column '%s' at row %ld

- Error: 1266 SQLSTATE: HY000 (ER_WARN_USING_OTHER_HANDLER)

  Message: Using storage engine %s for table '%s'

- Error: 1267 SQLSTATE: HY000 (ER_CANT_AGGREGATE_2COLLATIONS)

  Message: Illegal mix of collations (%s,%s) and (%s,%s) for operation '%s'

- Error: 1268 SQLSTATE: HY000 (ER_DROP_USER)

  Message: Cannot drop one or more of the requested users

- Error: 1269 SQLSTATE: HY000 (ER_REVOKE_GRANTS)

  Message: Can't revoke all privileges for one or more of the requested users

- Error: 1270 SQLSTATE: HY000 (ER_CANT_AGGREGATE_3COLLATIONS)

  Message: Illegal mix of collations (%s,%s), (%s,%s), (%s,%s) for operation '%s'

- Error: 1271 SQLSTATE: HY000 (ER_CANT_AGGREGATE_NCOLLATIONS)

  Message: Illegal mix of collations for operation '%s'

- Error: 1272 SQLSTATE: HY000 (ER_VARIABLE_IS_NOT_STRUCT)

  Message: Variable '%s' is not a variable component (can't be used as XXXX.variable_name)

- Error: 1273 SQLSTATE: HY000 (ER_UNKNOWN_COLLATION)

  Message: Unknown collation: '%s'

- Error: 1274 SQLSTATE: HY000 (ER_SLAVE_IGNORED_SSL_PARAMS)

  Message: SSL parameters in CHANGE MASTER are ignored because this MySQL slave was compiled without SSL support; they can be used later if MySQL slave with SSL is started

- Error: 1275 SQLSTATE: HY000 (ER_SERVER_IS_IN_SECURE_AUTH_MODE)

  Message: Server is running in --secure-auth mode, but '%s'@'%s' has a password in the old format; please change the password to the new format

- Error: 1276 SQLSTATE: HY000 (ER_WARN_FIELD_RESOLVED)

  Message: Field or reference '%s%s%s%s%s' of SELECT #%d was resolved in SELECT #%d

- Error: 1277 SQLSTATE: HY000 (ER_BAD_SLAVE_UNTIL_COND)

  Message: Incorrect parameter or combination of parameters for START SLAVE UNTIL

- Error: 1278 SQLSTATE: HY000 (ER_MISSING_SKIP_SLAVE)

  Message: It is recommended to use --skip-slave-start when doing step-by-step replication with START SLAVE UNTIL; otherwise, you will get problems if you get an unexpected slave's mysqld restart

- Error: 1279 SQLSTATE: HY000 (ER_UNTIL_COND_IGNORED)

  Message: SQL thread is not to be started so UNTIL options are ignored

- Error: 1280 SQLSTATE: 42000 (ER_WRONG_NAME_FOR_INDEX)

  Message: Incorrect index name '%s'

- Error: 1281 SQLSTATE: 42000 (ER_WRONG_NAME_FOR_CATALOG)

  Message: Incorrect catalog name '%s'

- Error: 1282 SQLSTATE: HY000 (ER_WARN_QC_RESIZE)

  Message: Query cache failed to set size %lu; new query cache size is %lu

- Error: 1283 SQLSTATE: HY000 (ER_BAD_FT_COLUMN)

  Message: Column '%s' cannot be part of FULLTEXT index

- Error: 1284 SQLSTATE: HY000 (ER_UNKNOWN_KEY_CACHE)

  Message: Unknown key cache '%s'

- Error: 1285 SQLSTATE: HY000 (ER_WARN_HOSTNAME_WONT_WORK)

  Message: MySQL is started in --skip-name-resolve mode; you must restart it without this switch for this grant to work

- Error: 1286 SQLSTATE: 42000 (ER_UNKNOWN_STORAGE_ENGINE)

  Message: Unknown storage engine '%s'

- Error: 1287 SQLSTATE: HY000 (ER_WARN_DEPRECATED_SYNTAX)

  Message: '%s' is deprecated and will be removed in a future release. Please use %s instead

- Error: 1288 SQLSTATE: HY000 (ER_NON_UPDATABLE_TABLE)

  Message: The target table %s of the %s is not updatable

- Error: 1289 SQLSTATE: HY000 (ER_FEATURE_DISABLED)

  Message: The '%s' feature is disabled; you need MySQL built with '%s' to have it working

- Error: 1290 SQLSTATE: HY000 (ER_OPTION_PREVENTS_STATEMENT)

Message: The MySQL server is running with the %s option so it cannot execute this statement

- Error: `1291` SQLSTATE: `HY000` (`ER_DUPLICATED_VALUE_IN_TYPE`)

  Message: Column '%s' has duplicated value '%s' in %s

- Error: `1292` SQLSTATE: `22007` (`ER_TRUNCATED_WRONG_VALUE`)

  Message: Truncated incorrect %s value: '%s'

- Error: `1293` SQLSTATE: `HY000` (`ER_TOO_MUCH_AUTO_TIMESTAMP_COLS`)

  Message: Incorrect table definition; there can be only one TIMESTAMP column with CURRENT_TIMESTAMP in DEFAULT or ON UPDATE clause

- Error: `1294` SQLSTATE: `HY000` (`ER_INVALID_ON_UPDATE`)

  Message: Invalid ON UPDATE clause for '%s' column

- Error: `1295` SQLSTATE: `HY000` (`ER_UNSUPPORTED_PS`)

  Message: This command is not supported in the prepared statement protocol yet

- Error: `1296` SQLSTATE: `HY000` (`ER_GET_ERRMSG`)

  Message: Got error %d '%s' from %s

- Error: `1297` SQLSTATE: `HY000` (`ER_GET_TEMPORARY_ERRMSG`)

  Message: Got temporary error %d '%s' from %s

- Error: `1298` SQLSTATE: `HY000` (`ER_UNKNOWN_TIME_ZONE`)

  Message: Unknown or incorrect time zone: '%s'

- Error: `1299` SQLSTATE: `HY000` (`ER_WARN_INVALID_TIMESTAMP`)

  Message: Invalid TIMESTAMP value in column '%s' at row %ld

- Error: `1300` SQLSTATE: `HY000` (`ER_INVALID_CHARACTER_STRING`)

  Message: Invalid %s character string: '%s'

- Error: `1301` SQLSTATE: `HY000` (`ER_WARN_ALLOWED_PACKET_OVERFLOWED`)

  Message: Result of %s() was larger than max_allowed_packet (%ld) - truncated

- Error: `1302` SQLSTATE: `HY000` (`ER_CONFLICTING_DECLARATIONS`)

  Message: Conflicting declarations: '%s%s' and '%s%s'

- Error: `1303` SQLSTATE: `2F003` (`ER_SP_NO_RECURSIVE_CREATE`)

  Message: Can't create a %s from within another stored routine

- Error: `1304` SQLSTATE: `42000` (`ER_SP_ALREADY_EXISTS`)

  Message: %s %s already exists

- Error: `1305` SQLSTATE: `42000` (`ER_SP_DOES_NOT_EXIST`)

Message: %s %s does not exist

- Error: 1306 SQLSTATE: HY000 (ER_SP_DROP_FAILED)

  Message: Failed to DROP %s %s

- Error: 1307 SQLSTATE: HY000 (ER_SP_STORE_FAILED)

  Message: Failed to CREATE %s %s

- Error: 1308 SQLSTATE: 42000 (ER_SP_LILABEL_MISMATCH)

  Message: %s with no matching label: %s

- Error: 1309 SQLSTATE: 42000 (ER_SP_LABEL_REDEFINE)

  Message: Redefining label %s

- Error: 1310 SQLSTATE: 42000 (ER_SP_LABEL_MISMATCH)

  Message: End-label %s without match

- Error: 1311 SQLSTATE: 01000 (ER_SP_UNINIT_VAR)

  Message: Referring to uninitialized variable %s

- Error: 1312 SQLSTATE: 0A000 (ER_SP_BADSELECT)

  Message: PROCEDURE %s can't return a result set in the given context

- Error: 1313 SQLSTATE: 42000 (ER_SP_BADRETURN)

  Message: RETURN is only allowed in a FUNCTION

- Error: 1314 SQLSTATE: 0A000 (ER_SP_BADSTATEMENT)

  Message: %s is not allowed in stored procedures

- Error: 1315 SQLSTATE: 42000 (ER_UPDATE_LOG_DEPRECATED_IGNORED)

  Message: The update log is deprecated and replaced by the binary log; SET SQL_LOG_UPDATE has been ignored.

- Error: 1316 SQLSTATE: 42000 (ER_UPDATE_LOG_DEPRECATED_TRANSLATED)

  Message: The update log is deprecated and replaced by the binary log; SET SQL_LOG_UPDATE has been translated to SET SQL_LOG_BIN.

- Error: 1317 SQLSTATE: 70100 (ER_QUERY_INTERRUPTED)

  Message: Query execution was interrupted

- Error: 1318 SQLSTATE: 42000 (ER_SP_WRONG_NO_OF_ARGS)

  Message: Incorrect number of arguments for %s %s; expected %u, got %u

- Error: 1319 SQLSTATE: 42000 (ER_SP_COND_MISMATCH)

  Message: Undefined CONDITION: %s

- Error: 1320 SQLSTATE: 42000 (ER_SP_NORETURN)

  Message: No RETURN found in FUNCTION %s
- Error: 1321 SQLSTATE: 2F005 (ER_SP_NORETURNEND)

  Message: FUNCTION %s ended without RETURN
- Error: 1322 SQLSTATE: 42000 (ER_SP_BAD_CURSOR_QUERY)

  Message: Cursor statement must be a SELECT
- Error: 1323 SQLSTATE: 42000 (ER_SP_BAD_CURSOR_SELECT)

  Message: Cursor SELECT must not have INTO
- Error: 1324 SQLSTATE: 42000 (ER_SP_CURSOR_MISMATCH)

  Message: Undefined CURSOR: %s
- Error: 1325 SQLSTATE: 24000 (ER_SP_CURSOR_ALREADY_OPEN)

  Message: Cursor is already open
- Error: 1326 SQLSTATE: 24000 (ER_SP_CURSOR_NOT_OPEN)

  Message: Cursor is not open
- Error: 1327 SQLSTATE: 42000 (ER_SP_UNDECLARED_VAR)

  Message: Undeclared variable: %s
- Error: 1328 SQLSTATE: HY000 (ER_SP_WRONG_NO_OF_FETCH_ARGS)

  Message: Incorrect number of FETCH variables
- Error: 1329 SQLSTATE: 02000 (ER_SP_FETCH_NO_DATA)

  Message: No data - zero rows fetched, selected, or processed
- Error: 1330 SQLSTATE: 42000 (ER_SP_DUP_PARAM)

  Message: Duplicate parameter: %s
- Error: 1331 SQLSTATE: 42000 (ER_SP_DUP_VAR)

  Message: Duplicate variable: %s
- Error: 1332 SQLSTATE: 42000 (ER_SP_DUP_COND)

  Message: Duplicate condition: %s
- Error: 1333 SQLSTATE: 42000 (ER_SP_DUP_CURS)

  Message: Duplicate cursor: %s
- Error: 1334 SQLSTATE: HY000 (ER_SP_CANT_ALTER)

  Message: Failed to ALTER %s %s

- Error: 1335 SQLSTATE: 0A000 (ER_SP_SUBSELECT_NYI)

  Message: Subquery value not supported

- Error: 1336 SQLSTATE: 0A000 (ER_STMT_NOT_ALLOWED_IN_SF_OR_TRG)

  Message: %s is not allowed in stored function or trigger

- Error: 1337 SQLSTATE: 42000 (ER_SP_VARCOND_AFTER_CURSHNDLR)

  Message: Variable or condition declaration after cursor or handler declaration

- Error: 1338 SQLSTATE: 42000 (ER_SP_CURSOR_AFTER_HANDLER)

  Message: Cursor declaration after handler declaration

- Error: 1339 SQLSTATE: 20000 (ER_SP_CASE_NOT_FOUND)

  Message: Case not found for CASE statement

- Error: 1340 SQLSTATE: HY000 (ER_FPARSER_TOO_BIG_FILE)

  Message: Configuration file '%s' is too big

- Error: 1341 SQLSTATE: HY000 (ER_FPARSER_BAD_HEADER)

  Message: Malformed file type header in file '%s'

- Error: 1342 SQLSTATE: HY000 (ER_FPARSER_EOF_IN_COMMENT)

  Message: Unexpected end of file while parsing comment '%s'

- Error: 1343 SQLSTATE: HY000 (ER_FPARSER_ERROR_IN_PARAMETER)

  Message: Error while parsing parameter '%s' (line: '%s')

- Error: 1344 SQLSTATE: HY000 (ER_FPARSER_EOF_IN_UNKNOWN_PARAMETER)

  Message: Unexpected end of file while skipping unknown parameter '%s'

- Error: 1345 SQLSTATE: HY000 (ER_VIEW_NO_EXPLAIN)

  Message: EXPLAIN/SHOW can not be issued; lacking privileges for underlying table

- Error: 1346 SQLSTATE: HY000 (ER_FRM_UNKNOWN_TYPE)

  Message: File '%s' has unknown type '%s' in its header

- Error: 1347 SQLSTATE: HY000 (ER_WRONG_OBJECT)

  Message: '%s.%s' is not %s

- Error: 1348 SQLSTATE: HY000 (ER_NONUPDATEABLE_COLUMN)

  Message: Column '%s' is not updatable

- Error: 1349 SQLSTATE: HY000 (ER_VIEW_SELECT_DERIVED)

  Message: View's SELECT contains a subquery in the FROM clause

- Error: 1350 SQLSTATE: HY000 (ER_VIEW_SELECT_CLAUSE)

  Message: View's SELECT contains a '%s' clause

- Error: 1351 SQLSTATE: HY000 (ER_VIEW_SELECT_VARIABLE)

  Message: View's SELECT contains a variable or parameter

- Error: 1352 SQLSTATE: HY000 (ER_VIEW_SELECT_TMPTABLE)

  Message: View's SELECT refers to a temporary table '%s'

- Error: 1353 SQLSTATE: HY000 (ER_VIEW_WRONG_LIST)

  Message: View's SELECT and view's field list have different column counts

- Error: 1354 SQLSTATE: HY000 (ER_WARN_VIEW_MERGE)

  Message: View merge algorithm can't be used here for now (assumed undefined algorithm)

- Error: 1355 SQLSTATE: HY000 (ER_WARN_VIEW_WITHOUT_KEY)

  Message: View being updated does not have complete key of underlying table in it

- Error: 1356 SQLSTATE: HY000 (ER_VIEW_INVALID)

  Message: View '%s.%s' references invalid table(s) or column(s) or function(s) or definer/invoker of view lack rights to use them

- Error: 1357 SQLSTATE: HY000 (ER_SP_NO_DROP_SP)

  Message: Can't drop or alter a %s from within another stored routine

- Error: 1358 SQLSTATE: HY000 (ER_SP_GOTO_IN_HNDLR)

  Message: GOTO is not allowed in a stored procedure handler

- Error: 1359 SQLSTATE: HY000 (ER_TRG_ALREADY_EXISTS)

  Message: Trigger already exists

- Error: 1360 SQLSTATE: HY000 (ER_TRG_DOES_NOT_EXIST)

  Message: Trigger does not exist

- Error: 1361 SQLSTATE: HY000 (ER_TRG_ON_VIEW_OR_TEMP_TABLE)

  Message: Trigger's '%s' is view or temporary table

- Error: 1362 SQLSTATE: HY000 (ER_TRG_CANT_CHANGE_ROW)

  Message: Updating of %s row is not allowed in %strigger

- Error: 1363 SQLSTATE: HY000 (ER_TRG_NO_SUCH_ROW_IN_TRG)

  Message: There is no %s row in %s trigger

- Error: 1364 SQLSTATE: HY000 (ER_NO_DEFAULT_FOR_FIELD)

  Message: Field '%s' doesn't have a default value

- Error: 1365 SQLSTATE: 22012 (ER_DIVISION_BY_ZERO)

  Message: Division by 0

- Error: 1366 SQLSTATE: HY000 (ER_TRUNCATED_WRONG_VALUE_FOR_FIELD)

  Message: Incorrect %s value: '%s' for column '%s' at row %ld

- Error: 1367 SQLSTATE: 22007 (ER_ILLEGAL_VALUE_FOR_TYPE)

  Message: Illegal %s '%s' value found during parsing

- Error: 1368 SQLSTATE: HY000 (ER_VIEW_NONUPD_CHECK)

  Message: CHECK OPTION on non-updatable view '%s.%s'

- Error: 1369 SQLSTATE: HY000 (ER_VIEW_CHECK_FAILED)

  Message: CHECK OPTION failed '%s.%s'

- Error: 1370 SQLSTATE: 42000 (ER_PROCACCESS_DENIED_ERROR)

  Message: %s command denied to user '%s'@'%s' for routine '%s'

- Error: 1371 SQLSTATE: HY000 (ER_RELAY_LOG_FAIL)

  Message: Failed purging old relay logs: %s

- Error: 1372 SQLSTATE: HY000 (ER_PASSWD_LENGTH)

  Message: Password hash should be a %d-digit hexadecimal number

- Error: 1373 SQLSTATE: HY000 (ER_UNKNOWN_TARGET_BINLOG)

  Message: Target log not found in binlog index

- Error: 1374 SQLSTATE: HY000 (ER_IO_ERR_LOG_INDEX_READ)

  Message: I/O error reading log index file

- Error: 1375 SQLSTATE: HY000 (ER_BINLOG_PURGE_PROHIBITED)

  Message: Server configuration does not permit binlog purge

- Error: 1376 SQLSTATE: HY000 (ER_FSEEK_FAIL)

  Message: Failed on fseek()

- Error: 1377 SQLSTATE: HY000 (ER_BINLOG_PURGE_FATAL_ERR)

  Message: Fatal error during log purge

- Error: 1378 SQLSTATE: HY000 (ER_LOG_IN_USE)

  Message: A purgeable log is in use, will not purge

- Error: 1379 SQLSTATE: HY000 (ER_LOG_PURGE_UNKNOWN_ERR)

  Message: Unknown error during log purge

- Error: 1380 SQLSTATE: HY000 (ER_RELAY_LOG_INIT)

  Message: Failed initializing relay log position: %s

- Error: 1381 SQLSTATE: HY000 (ER_NO_BINARY_LOGGING)

  Message: You are not using binary logging

- Error: 1382 SQLSTATE: HY000 (ER_RESERVED_SYNTAX)

  Message: The '%s' syntax is reserved for purposes internal to the MySQL server

- Error: 1383 SQLSTATE: HY000 (ER_WSAS_FAILED)

  Message: WSAStartup Failed

- Error: 1384 SQLSTATE: HY000 (ER_DIFF_GROUPS_PROC)

  Message: Can't handle procedures with different groups yet

- Error: 1385 SQLSTATE: HY000 (ER_NO_GROUP_FOR_PROC)

  Message: Select must have a group with this procedure

- Error: 1386 SQLSTATE: HY000 (ER_ORDER_WITH_PROC)

  Message: Can't use ORDER clause with this procedure

- Error: 1387 SQLSTATE: HY000 (ER_LOGGING_PROHIBIT_CHANGING_OF)

  Message: Binary logging and replication forbid changing the global server %s

- Error: 1388 SQLSTATE: HY000 (ER_NO_FILE_MAPPING)

  Message: Can't map file: %s, errno: %d

- Error: 1389 SQLSTATE: HY000 (ER_WRONG_MAGIC)

  Message: Wrong magic in %s

- Error: 1390 SQLSTATE: HY000 (ER_PS_MANY_PARAM)

  Message: Prepared statement contains too many placeholders

- Error: 1391 SQLSTATE: HY000 (ER_KEY_PART_0)

  Message: Key part '%s' length cannot be 0

- Error: 1392 SQLSTATE: HY000 (ER_VIEW_CHECKSUM)

  Message: View text checksum failed

- Error: 1393 SQLSTATE: HY000 (ER_VIEW_MULTIUPDATE)

  Message: Can not modify more than one base table through a join view '%s.%s'

- Error: 1394 SQLSTATE: HY000 (ER_VIEW_NO_INSERT_FIELD_LIST)

  Message: Can not insert into join view '%s.%s' without fields list

- Error: 1395 SQLSTATE: HY000 (ER_VIEW_DELETE_MERGE_VIEW)

  Message: Can not delete from join view '%s.%s'

- Error: 1396 SQLSTATE: HY000 (ER_CANNOT_USER)

  Message: Operation %s failed for %s

- Error: 1397 SQLSTATE: XAE04 (ER_XAER_NOTA)

  Message: XAER_NOTA: Unknown XID

- Error: 1398 SQLSTATE: XAE05 (ER_XAER_INVAL)

  Message: XAER_INVAL: Invalid arguments (or unsupported command)

- Error: 1399 SQLSTATE: XAE07 (ER_XAER_RMFAIL)

  Message: XAER_RMFAIL: The command cannot be executed when global transaction is in the %s state

- Error: 1400 SQLSTATE: XAE09 (ER_XAER_OUTSIDE)

  Message: XAER_OUTSIDE: Some work is done outside global transaction

- Error: 1401 SQLSTATE: XAE03 (ER_XAER_RMERR)

  Message: XAER_RMERR: Fatal error occurred in the transaction branch - check your data for consistency

- Error: 1402 SQLSTATE: XA100 (ER_XA_RBROLLBACK)

  Message: XA_RBROLLBACK: Transaction branch was rolled back

- Error: 1403 SQLSTATE: 42000 (ER_NONEXISTING_PROC_GRANT)

  Message: There is no such grant defined for user '%s' on host '%s' on routine '%s'

- Error: 1404 SQLSTATE: HY000 (ER_PROC_AUTO_GRANT_FAIL)

  Message: Failed to grant EXECUTE and ALTER ROUTINE privileges

- Error: 1405 SQLSTATE: HY000 (ER_PROC_AUTO_REVOKE_FAIL)

  Message: Failed to revoke all privileges to dropped routine

- Error: 1406 SQLSTATE: 22001 (ER_DATA_TOO_LONG)

  Message: Data too long for column '%s' at row %ld

- Error: 1407 SQLSTATE: 42000 (ER_SP_BAD_SQLSTATE)

  Message: Bad SQLSTATE: '%s'

- Error: 1408 SQLSTATE: HY000 (ER_STARTUP)

  Message: %s: ready for connections. Version: '%s' socket: '%s' port: %d %s

- Error: 1409 SQLSTATE: HY000 (ER_LOAD_FROM_FIXED_SIZE_ROWS_TO_VAR)

  Message: Can't load value from file with fixed size rows to variable

- Error: 1410 SQLSTATE: 42000 (ER_CANT_CREATE_USER_WITH_GRANT)

  Message: You are not allowed to create a user with GRANT

- Error: 1411 SQLSTATE: HY000 (ER_WRONG_VALUE_FOR_TYPE)

  Message: Incorrect %s value: '%s' for function %s

- Error: 1412 SQLSTATE: HY000 (ER_TABLE_DEF_CHANGED)

  Message: Table definition has changed, please retry transaction

- Error: 1413 SQLSTATE: 42000 (ER_SP_DUP_HANDLER)

  Message: Duplicate handler declared in the same block

- Error: 1414 SQLSTATE: 42000 (ER_SP_NOT_VAR_ARG)

  Message: OUT or INOUT argument %d for routine %s is not a variable or NEW pseudo-variable in BEFORE trigger

- Error: 1415 SQLSTATE: 0A000 (ER_SP_NO_RETSET)

  Message: Not allowed to return a result set from a %s

- Error: 1416 SQLSTATE: 22003 (ER_CANT_CREATE_GEOMETRY_OBJECT)

  Message: Cannot get geometry object from data you send to the GEOMETRY field

- Error: 1417 SQLSTATE: HY000 (ER_FAILED_ROUTINE_BREAK_BINLOG)

  Message: A routine failed and has neither NO SQL nor READS SQL DATA in its declaration and binary logging is enabled; if non-transactional tables were updated, the binary log will miss their changes

- Error: 1418 SQLSTATE: HY000 (ER_BINLOG_UNSAFE_ROUTINE)

  Message: This function has none of DETERMINISTIC, NO SQL, or READS SQL DATA in its declaration and binary logging is enabled (you *might* want to use the less safe log_bin_trust_function_creators variable)

- Error: 1419 SQLSTATE: HY000 (ER_BINLOG_CREATE_ROUTINE_NEED_SUPER)

  Message: You do not have the SUPER privilege and binary logging is enabled (you *might* want to use the less safe log_bin_trust_function_creators variable)

- Error: 1420 SQLSTATE: HY000 (ER_EXEC_STMT_WITH_OPEN_CURSOR)

  Message: You can't execute a prepared statement which has an open cursor associated with it. Reset the statement to re-execute it.

- Error: 1421 SQLSTATE: HY000 (ER_STMT_HAS_NO_OPEN_CURSOR)

  Message: The statement (%lu) has no open cursor.

- Error: 1422 SQLSTATE: HY000 (ER_COMMIT_NOT_ALLOWED_IN_SF_OR_TRG)

  Message: Explicit or implicit commit is not allowed in stored function or trigger.

- Error: 1423 SQLSTATE: HY000 (ER_NO_DEFAULT_FOR_VIEW_FIELD)

Message: Field of view '%s.%s' underlying table doesn't have a default value

- Error: 1424 SQLSTATE: HY000 (ER_SP_NO_RECURSION)

  Message: Recursive stored functions and triggers are not allowed.

- Error: 1425 SQLSTATE: 42000 (ER_TOO_BIG_SCALE)

  Message: Too big scale %d specified for column '%s'. Maximum is %lu.

- Error: 1426 SQLSTATE: 42000 (ER_TOO_BIG_PRECISION)

  Message: Too big precision %d specified for column '%s'. Maximum is %lu.

- Error: 1427 SQLSTATE: 42000 (ER_M_BIGGER_THAN_D)

  Message: For float(M,D), double(M,D) or decimal(M,D), M must be >= D (column '%s').

- Error: 1428 SQLSTATE: HY000 (ER_WRONG_LOCK_OF_SYSTEM_TABLE)

  Message: You can't combine write-locking of system tables with other tables or lock types

- Error: 1429 SQLSTATE: HY000 (ER_CONNECT_TO_FOREIGN_DATA_SOURCE)

  Message: Unable to connect to foreign data source: %s

- Error: 1430 SQLSTATE: HY000 (ER_QUERY_ON_FOREIGN_DATA_SOURCE)

  Message: There was a problem processing the query on the foreign data source. Data source error: %s

- Error: 1431 SQLSTATE: HY000 (ER_FOREIGN_DATA_SOURCE_DOESNT_EXIST)

  Message: The foreign data source you are trying to reference does not exist. Data source error: %s

- Error: 1432 SQLSTATE: HY000 (ER_FOREIGN_DATA_STRING_INVALID_CANT_CREATE)

  Message: Can't create federated table. The data source connection string '%s' is not in the correct format

- Error: 1433 SQLSTATE: HY000 (ER_FOREIGN_DATA_STRING_INVALID)

  Message: The data source connection string '%s' is not in the correct format

- Error: 1434 SQLSTATE: HY000 (ER_CANT_CREATE_FEDERATED_TABLE)

  Message: Can't create federated table. Foreign data src error: %s

- Error: 1435 SQLSTATE: HY000 (ER_TRG_IN_WRONG_SCHEMA)

  Message: Trigger in wrong schema

- Error: 1436 SQLSTATE: HY000 (ER_STACK_OVERRUN_NEED_MORE)

  Message: Thread stack overrun: %ld bytes used of a %ld byte stack, and %ld bytes needed. Use 'mysqld --thread_stack=#' to specify a bigger stack.

- Error: 1437 SQLSTATE: 42000 (ER_TOO_LONG_BODY)

  Message: Routine body for '%s' is too long

- Error: 1438 SQLSTATE: HY000 (ER_WARN_CANT_DROP_DEFAULT_KEYCACHE)

  Message: Cannot drop default keycache

- Error: 1439 SQLSTATE: 42000 (ER_TOO_BIG_DISPLAYWIDTH)

  Message: Display width out of range for column '%s' (max = %lu)

- Error: 1440 SQLSTATE: XAE08 (ER_XAER_DUPID)

  Message: XAER_DUPID: The XID already exists

- Error: 1441 SQLSTATE: 22008 (ER_DATETIME_FUNCTION_OVERFLOW)

  Message: Datetime function: %s field overflow

- Error: 1442 SQLSTATE: HY000 (ER_CANT_UPDATE_USED_TABLE_IN_SF_OR_TRG)

  Message: Can't update table '%s' in stored function/trigger because it is already used by statement which invoked this stored function/trigger.

- Error: 1443 SQLSTATE: HY000 (ER_VIEW_PREVENT_UPDATE)

  Message: The definition of table '%s' prevents operation %s on table '%s'.

- Error: 1444 SQLSTATE: HY000 (ER_PS_NO_RECURSION)

  Message: The prepared statement contains a stored routine call that refers to that same statement. It's not allowed to execute a prepared statement in such a recursive manner

- Error: 1445 SQLSTATE: HY000 (ER_SP_CANT_SET_AUTOCOMMIT)

  Message: Not allowed to set autocommit from a stored function or trigger

- Error: 1446 SQLSTATE: HY000 (ER_MALFORMED_DEFINER)

  Message: Definer is not fully qualified

- Error: 1447 SQLSTATE: HY000 (ER_VIEW_FRM_NO_USER)

  Message: View '%s'.'%s' has no definer information (old table format). Current user is used as definer. Please recreate the view!

- Error: 1448 SQLSTATE: HY000 (ER_VIEW_OTHER_USER)

  Message: You need the SUPER privilege for creation view with '%s'@'%s' definer

- Error: 1449 SQLSTATE: HY000 (ER_NO_SUCH_USER)

  Message: The user specified as a definer ('%s'@'%s') does not exist

- Error: 1450 SQLSTATE: HY000 (ER_FORBID_SCHEMA_CHANGE)

  Message: Changing schema from '%s' to '%s' is not allowed.

- Error: 1451 SQLSTATE: 23000 (ER_ROW_IS_REFERENCED_2)

  Message: Cannot delete or update a parent row: a foreign key constraint fails (%s)

- Error: 1452 SQLSTATE: 23000 (ER_NO_REFERENCED_ROW_2)

Message: Cannot add or update a child row: a foreign key constraint fails (%s)

- Error: `1453` SQLSTATE: `42000` (`ER_SP_BAD_VAR_SHADOW`)

  Message: Variable '%s' must be quoted with `...`, or renamed

- Error: `1454` SQLSTATE: `HY000` (`ER_TRG_NO_DEFINER`)

  Message: No definer attribute for trigger '%s'.'%s'. The trigger will be activated under the authorization of the invoker, which may have insufficient privileges. Please recreate the trigger.

- Error: `1455` SQLSTATE: `HY000` (`ER_OLD_FILE_FORMAT`)

  Message: '%s' has an old format, you should re-create the '%s' object(s)

- Error: `1456` SQLSTATE: `HY000` (`ER_SP_RECURSION_LIMIT`)

  Message: Recursive limit %d (as set by the max_sp_recursion_depth variable) was exceeded for routine %s

- Error: `1457` SQLSTATE: `HY000` (`ER_SP_PROC_TABLE_CORRUPT`)

  Message: Failed to load routine %s. The table mysql.proc is missing, corrupt, or contains bad data (internal code %d)

- Error: `1458` SQLSTATE: `42000` (`ER_SP_WRONG_NAME`)

  Message: Incorrect routine name '%s'

- Error: `1459` SQLSTATE: `HY000` (`ER_TABLE_NEEDS_UPGRADE`)

  Message: Table upgrade required. Please do "REPAIR TABLE `%s`" or dump/reload to fix it!

- Error: `1460` SQLSTATE: `42000` (`ER_SP_NO_AGGREGATE`)

  Message: AGGREGATE is not supported for stored functions

- Error: `1461` SQLSTATE: `42000` (`ER_MAX_PREPARED_STMT_COUNT_REACHED`)

  Message: Can't create more than max_prepared_stmt_count statements (current value: %lu)

- Error: `1462` SQLSTATE: `HY000` (`ER_VIEW_RECURSIVE`)

  Message: `%s`.`%s` contains view recursion

- Error: `1463` SQLSTATE: `42000` (`ER_NON_GROUPING_FIELD_USED`)

  Message: Non-grouping field '%s' is used in %s clause

- Error: `1464` SQLSTATE: `HY000` (`ER_TABLE_CANT_HANDLE_SPKEYS`)

  Message: The used table type doesn't support SPATIAL indexes

- Error: `1465` SQLSTATE: `HY000` (`ER_NO_TRIGGERS_ON_SYSTEM_SCHEMA`)

  Message: Triggers can not be created on system tables

- Error: `1466` SQLSTATE: `HY000` (`ER_REMOVED_SPACES`)

  Message: Leading spaces are removed from name '%s'

- Error: 1467 SQLSTATE: HY000 (ER_AUTOINC_READ_FAILED)

  Message: Failed to read auto-increment value from storage engine

- Error: 1468 SQLSTATE: HY000 (ER_USERNAME)

  Message: user name

- Error: 1469 SQLSTATE: HY000 (ER_HOSTNAME)

  Message: host name

- Error: 1470 SQLSTATE: HY000 (ER_WRONG_STRING_LENGTH)

  Message: String '%s' is too long for %s (should be no longer than %d)

- Error: 1471 SQLSTATE: HY000 (ER_NON_INSERTABLE_TABLE)

  Message: The target table %s of the %s is not insertable-into

- Error: 1472 SQLSTATE: HY000 (ER_ADMIN_WRONG_MRG_TABLE)

  Message: Table '%s' is differently defined or of non-MyISAM type or doesn't exist

- Error: 1473 SQLSTATE: HY000 (ER_TOO_HIGH_LEVEL_OF_NESTING_FOR_SELECT)

  Message: Too high level of nesting for select

- Error: 1474 SQLSTATE: HY000 (ER_NAME_BECOMES_EMPTY)

  Message: Name '%s' has become ''

- Error: 1475 SQLSTATE: HY000 (ER_AMBIGUOUS_FIELD_TERM)

  Message: First character of the FIELDS TERMINATED string is ambiguous; please use non-optional and non-empty FIELDS ENCLOSED BY

- Error: 1476 SQLSTATE: HY000 (ER_FOREIGN_SERVER_EXISTS)

  Message: The foreign server, %s, you are trying to create already exists.

- Error: 1477 SQLSTATE: HY000 (ER_FOREIGN_SERVER_DOESNT_EXIST)

  Message: The foreign server name you are trying to reference does not exist. Data source error: %s

- Error: 1478 SQLSTATE: HY000 (ER_ILLEGAL_HA_CREATE_OPTION)

  Message: Table storage engine '%s' does not support the create option '%s'

- Error: 1479 SQLSTATE: HY000 (ER_PARTITION_REQUIRES_VALUES_ERROR)

  Message: Syntax error: %s PARTITIONING requires definition of VALUES %s for each partition

- Error: 1480 SQLSTATE: HY000 (ER_PARTITION_WRONG_VALUES_ERROR)

  Message: Only %s PARTITIONING can use VALUES %s in partition definition

- Error: 1481 SQLSTATE: HY000 (ER_PARTITION_MAXVALUE_ERROR)

  Message: MAXVALUE can only be used in last partition definition

- Error: 1482 SQLSTATE: HY000 (ER_PARTITION_SUBPARTITION_ERROR)

  Message: Subpartitions can only be hash partitions and by key
- Error: 1483 SQLSTATE: HY000 (ER_PARTITION_SUBPART_MIX_ERROR)

  Message: Must define subpartitions on all partitions if on one partition
- Error: 1484 SQLSTATE: HY000 (ER_PARTITION_WRONG_NO_PART_ERROR)

  Message: Wrong number of partitions defined, mismatch with previous setting
- Error: 1485 SQLSTATE: HY000 (ER_PARTITION_WRONG_NO_SUBPART_ERROR)

  Message: Wrong number of subpartitions defined, mismatch with previous setting
- Error: 1486 SQLSTATE: HY000 (ER_WRONG_EXPR_IN_PARTITION_FUNC_ERROR)

  Message: Constant, random or timezone-dependent expressions in (sub)partitioning function are not allowed
- Error: 1487 SQLSTATE: HY000 (ER_NO_CONST_EXPR_IN_RANGE_OR_LIST_ERROR)

  Message: Expression in RANGE/LIST VALUES must be constant
- Error: 1488 SQLSTATE: HY000 (ER_FIELD_NOT_FOUND_PART_ERROR)

  Message: Field in list of fields for partition function not found in table
- Error: 1489 SQLSTATE: HY000 (ER_LIST_OF_FIELDS_ONLY_IN_HASH_ERROR)

  Message: List of fields is only allowed in KEY partitions
- Error: 1490 SQLSTATE: HY000 (ER_INCONSISTENT_PARTITION_INFO_ERROR)

  Message: The partition info in the frm file is not consistent with what can be written into the frm file
- Error: 1491 SQLSTATE: HY000 (ER_PARTITION_FUNC_NOT_ALLOWED_ERROR)

  Message: The %s function returns the wrong type
- Error: 1492 SQLSTATE: HY000 (ER_PARTITIONS_MUST_BE_DEFINED_ERROR)

  Message: For %s partitions each partition must be defined
- Error: 1493 SQLSTATE: HY000 (ER_RANGE_NOT_INCREASING_ERROR)

  Message: VALUES LESS THAN value must be strictly increasing for each partition
- Error: 1494 SQLSTATE: HY000 (ER_INCONSISTENT_TYPE_OF_FUNCTIONS_ERROR)

  Message: VALUES value must be of same type as partition function
- Error: 1495 SQLSTATE: HY000 (ER_MULTIPLE_DEF_CONST_IN_LIST_PART_ERROR)

  Message: Multiple definition of same constant in list partitioning
- Error: 1496 SQLSTATE: HY000 (ER_PARTITION_ENTRY_ERROR)

  Message: Partitioning can not be used stand-alone in query

- Error: 1497 SQLSTATE: HY000 (ER_MIX_HANDLER_ERROR)

  Message: The mix of handlers in the partitions is not allowed in this version of MySQL
- Error: 1498 SQLSTATE: HY000 (ER_PARTITION_NOT_DEFINED_ERROR)

  Message: For the partitioned engine it is necessary to define all %s
- Error: 1499 SQLSTATE: HY000 (ER_TOO_MANY_PARTITIONS_ERROR)

  Message: Too many partitions (including subpartitions) were defined
- Error: 1500 SQLSTATE: HY000 (ER_SUBPARTITION_ERROR)

  Message: It is only possible to mix RANGE/LIST partitioning with HASH/KEY partitioning for subpartitioning
- Error: 1501 SQLSTATE: HY000 (ER_CANT_CREATE_HANDLER_FILE)

  Message: Failed to create specific handler file
- Error: 1502 SQLSTATE: HY000 (ER_BLOB_FIELD_IN_PART_FUNC_ERROR)

  Message: A BLOB field is not allowed in partition function
- Error: 1503 SQLSTATE: HY000 (ER_UNIQUE_KEY_NEED_ALL_FIELDS_IN_PF)

  Message: A %s must include all columns in the table's partitioning function
- Error: 1504 SQLSTATE: HY000 (ER_NO_PARTS_ERROR)

  Message: Number of %s = 0 is not an allowed value
- Error: 1505 SQLSTATE: HY000 (ER_PARTITION_MGMT_ON_NONPARTITIONED)

  Message: Partition management on a not partitioned table is not possible
- Error: 1506 SQLSTATE: HY000 (ER_FOREIGN_KEY_ON_PARTITIONED)

  Message: Foreign key clause is not yet supported in conjunction with partitioning
- Error: 1507 SQLSTATE: HY000 (ER_DROP_PARTITION_NON_EXISTENT)

  Message: Error in list of partitions to %s
- Error: 1508 SQLSTATE: HY000 (ER_DROP_LAST_PARTITION)

  Message: Cannot remove all partitions, use DROP TABLE instead
- Error: 1509 SQLSTATE: HY000 (ER_COALESCE_ONLY_ON_HASH_PARTITION)

  Message: COALESCE PARTITION can only be used on HASH/KEY partitions
- Error: 1510 SQLSTATE: HY000 (ER_REORG_HASH_ONLY_ON_SAME_NO)

  Message: REORGANIZE PARTITION can only be used to reorganize partitions not to change their numbers
- Error: 1511 SQLSTATE: HY000 (ER_REORG_NO_PARAM_ERROR)

Message: REORGANIZE PARTITION without parameters can only be used on auto-partitioned tables using HASH PARTITIONs

- Error: 1512 SQLSTATE: HY000 (ER_ONLY_ON_RANGE_LIST_PARTITION)

  Message: %s PARTITION can only be used on RANGE/LIST partitions

- Error: 1513 SQLSTATE: HY000 (ER_ADD_PARTITION_SUBPART_ERROR)

  Message: Trying to Add partition(s) with wrong number of subpartitions

- Error: 1514 SQLSTATE: HY000 (ER_ADD_PARTITION_NO_NEW_PARTITION)

  Message: At least one partition must be added

- Error: 1515 SQLSTATE: HY000 (ER_COALESCE_PARTITION_NO_PARTITION)

  Message: At least one partition must be coalesced

- Error: 1516 SQLSTATE: HY000 (ER_REORG_PARTITION_NOT_EXIST)

  Message: More partitions to reorganize than there are partitions

- Error: 1517 SQLSTATE: HY000 (ER_SAME_NAME_PARTITION)

  Message: Duplicate partition name %s

- Error: 1518 SQLSTATE: HY000 (ER_NO_BINLOG_ERROR)

  Message: It is not allowed to shut off binlog on this command

- Error: 1519 SQLSTATE: HY000 (ER_CONSECUTIVE_REORG_PARTITIONS)

  Message: When reorganizing a set of partitions they must be in consecutive order

- Error: 1520 SQLSTATE: HY000 (ER_REORG_OUTSIDE_RANGE)

  Message: Reorganize of range partitions cannot change total ranges except for last partition where it can extend the range

- Error: 1521 SQLSTATE: HY000 (ER_PARTITION_FUNCTION_FAILURE)

  Message: Partition function not supported in this version for this handler

- Error: 1522 SQLSTATE: HY000 (ER_PART_STATE_ERROR)

  Message: Partition state cannot be defined from CREATE/ALTER TABLE

- Error: 1523 SQLSTATE: HY000 (ER_LIMITED_PART_RANGE)

  Message: The %s handler only supports 32 bit integers in VALUES

- Error: 1524 SQLSTATE: HY000 (ER_PLUGIN_IS_NOT_LOADED)

  Message: Plugin '%s' is not loaded

- Error: 1525 SQLSTATE: HY000 (ER_WRONG_VALUE)

  Message: Incorrect %s value: '%s'

- Error: 1526 SQLSTATE: HY000 (ER_NO_PARTITION_FOR_GIVEN_VALUE)

  Message: Table has no partition for value %s

- Error: 1527 SQLSTATE: HY000 (ER_FILEGROUP_OPTION_ONLY_ONCE)

  Message: It is not allowed to specify %s more than once

- Error: 1528 SQLSTATE: HY000 (ER_CREATE_FILEGROUP_FAILED)

  Message: Failed to create %s

- Error: 1529 SQLSTATE: HY000 (ER_DROP_FILEGROUP_FAILED)

  Message: Failed to drop %s

- Error: 1530 SQLSTATE: HY000 (ER_TABLESPACE_AUTO_EXTEND_ERROR)

  Message: The handler doesn't support autoextend of tablespaces

- Error: 1531 SQLSTATE: HY000 (ER_WRONG_SIZE_NUMBER)

  Message: A size parameter was incorrectly specified, either number or on the form 10M

- Error: 1532 SQLSTATE: HY000 (ER_SIZE_OVERFLOW_ERROR)

  Message: The size number was correct but we don't allow the digit part to be more than 2 billion

- Error: 1533 SQLSTATE: HY000 (ER_ALTER_FILEGROUP_FAILED)

  Message: Failed to alter: %s

- Error: 1534 SQLSTATE: HY000 (ER_BINLOG_ROW_LOGGING_FAILED)

  Message: Writing one row to the row-based binary log failed

- Error: 1535 SQLSTATE: HY000 (ER_BINLOG_ROW_WRONG_TABLE_DEF)

  Message: Table definition on master and slave does not match: %s

- Error: 1536 SQLSTATE: HY000 (ER_BINLOG_ROW_RBR_TO_SBR)

  Message: Slave running with --log-slave-updates must use row-based binary logging to be able to replicate row-based binary log events

- Error: 1537 SQLSTATE: HY000 (ER_EVENT_ALREADY_EXISTS)

  Message: Event '%s' already exists

- Error: 1538 SQLSTATE: HY000 (ER_EVENT_STORE_FAILED)

  Message: Failed to store event %s. Error code %d from storage engine.

- Error: 1539 SQLSTATE: HY000 (ER_EVENT_DOES_NOT_EXIST)

  Message: Unknown event '%s'

- Error: 1540 SQLSTATE: HY000 (ER_EVENT_CANT_ALTER)

  Message: Failed to alter event '%s'

- Error: 1541 SQLSTATE: HY000 (ER_EVENT_DROP_FAILED)

  Message: Failed to drop %s
- Error: 1542 SQLSTATE: HY000 (ER_EVENT_INTERVAL_NOT_POSITIVE_OR_TOO_BIG)

  Message: INTERVAL is either not positive or too big
- Error: 1543 SQLSTATE: HY000 (ER_EVENT_ENDS_BEFORE_STARTS)

  Message: ENDS is either invalid or before STARTS
- Error: 1544 SQLSTATE: HY000 (ER_EVENT_EXEC_TIME_IN_THE_PAST)

  Message: Event execution time is in the past. Event has been disabled
- Error: 1545 SQLSTATE: HY000 (ER_EVENT_OPEN_TABLE_FAILED)

  Message: Failed to open mysql.event
- Error: 1546 SQLSTATE: HY000 (ER_EVENT_NEITHER_M_EXPR_NOR_M_AT)

  Message: No datetime expression provided
- Error: 1547 SQLSTATE: HY000 (ER_OBSOLETE_COL_COUNT_DOESNT_MATCH_CORRUPTED)

  Message: Column count of mysql.%s is wrong. Expected %d, found %d. The table is probably corrupted
- Error: 1548 SQLSTATE: HY000 (ER_OBSOLETE_CANNOT_LOAD_FROM_TABLE)

  Message: Cannot load from mysql.%s. The table is probably corrupted
- Error: 1549 SQLSTATE: HY000 (ER_EVENT_CANNOT_DELETE)

  Message: Failed to delete the event from mysql.event
- Error: 1550 SQLSTATE: HY000 (ER_EVENT_COMPILE_ERROR)

  Message: Error during compilation of event's body
- Error: 1551 SQLSTATE: HY000 (ER_EVENT_SAME_NAME)

  Message: Same old and new event name
- Error: 1552 SQLSTATE: HY000 (ER_EVENT_DATA_TOO_LONG)

  Message: Data for column '%s' too long
- Error: 1553 SQLSTATE: HY000 (ER_DROP_INDEX_FK)

  Message: Cannot drop index '%s': needed in a foreign key constraint
- Error: 1554 SQLSTATE: HY000 (ER_WARN_DEPRECATED_SYNTAX_WITH_VER)

  Message: The syntax '%s' is deprecated and will be removed in MySQL %s. Please use %s instead
- Error: 1555 SQLSTATE: HY000 (ER_CANT_WRITE_LOCK_LOG_TABLE)

  Message: You can't write-lock a log table. Only read access is possible

- Error: 1556 SQLSTATE: HY000 (ER_CANT_LOCK_LOG_TABLE)

  Message: You can't use locks with log tables.

- Error: 1557 SQLSTATE: 23000 (ER_FOREIGN_DUPLICATE_KEY_OLD_UNUSED)

  Message: Upholding foreign key constraints for table '%s', entry '%s', key %d would lead to a duplicate entry

- Error: 1558 SQLSTATE: HY000 (ER_COL_COUNT_DOESNT_MATCH_PLEASE_UPDATE)

  Message: Column count of mysql.%s is wrong. Expected %d, found %d. Created with MySQL %d, now running %d. Please use mysql_upgrade to fix this error.

- Error: 1559 SQLSTATE: HY000 (ER_TEMP_TABLE_PREVENTS_SWITCH_OUT_OF_RBR)

  Message: Cannot switch out of the row-based binary log format when the session has open temporary tables

- Error: 1560 SQLSTATE: HY000 (ER_STORED_FUNCTION_PREVENTS_SWITCH_BINLOG_FORMAT)

  Message: Cannot change the binary logging format inside a stored function or trigger

- Error: 1561 SQLSTATE: HY000 (ER_NDB_CANT_SWITCH_BINLOG_FORMAT)

  Message: The NDB cluster engine does not support changing the binlog format on the fly yet

- Error: 1562 SQLSTATE: HY000 (ER_PARTITION_NO_TEMPORARY)

  Message: Cannot create temporary table with partitions

- Error: 1563 SQLSTATE: HY000 (ER_PARTITION_CONST_DOMAIN_ERROR)

  Message: Partition constant is out of partition function domain

- Error: 1564 SQLSTATE: HY000 (ER_PARTITION_FUNCTION_IS_NOT_ALLOWED)

  Message: This partition function is not allowed

- Error: 1565 SQLSTATE: HY000 (ER_DDL_LOG_ERROR)

  Message: Error in DDL log

- Error: 1566 SQLSTATE: HY000 (ER_NULL_IN_VALUES_LESS_THAN)

  Message: Not allowed to use NULL value in VALUES LESS THAN

- Error: 1567 SQLSTATE: HY000 (ER_WRONG_PARTITION_NAME)

  Message: Incorrect partition name

- Error: 1568 SQLSTATE: 25001 (ER_CANT_CHANGE_TX_CHARACTERISTICS)

  Message: Transaction characteristics can't be changed while a transaction is in progress

- Error: 1569 SQLSTATE: HY000 (ER_DUP_ENTRY_AUTOINCREMENT_CASE)

  Message: ALTER TABLE causes auto_increment resequencing, resulting in duplicate entry '%s' for key '%s'

- Error: 1570 SQLSTATE: HY000 (ER_EVENT_MODIFY_QUEUE_ERROR)

  Message: Internal scheduler error %d

- Error: 1571 SQLSTATE: HY000 (ER_EVENT_SET_VAR_ERROR)

  Message: Error during starting/stopping of the scheduler. Error code %u

- Error: 1572 SQLSTATE: HY000 (ER_PARTITION_MERGE_ERROR)

  Message: Engine cannot be used in partitioned tables

- Error: 1573 SQLSTATE: HY000 (ER_CANT_ACTIVATE_LOG)

  Message: Cannot activate '%s' log

- Error: 1574 SQLSTATE: HY000 (ER_RBR_NOT_AVAILABLE)

  Message: The server was not built with row-based replication

- Error: 1575 SQLSTATE: HY000 (ER_BASE64_DECODE_ERROR)

  Message: Decoding of base64 string failed

- Error: 1576 SQLSTATE: HY000 (ER_EVENT_RECURSION_FORBIDDEN)

  Message: Recursion of EVENT DDL statements is forbidden when body is present

- Error: 1577 SQLSTATE: HY000 (ER_EVENTS_DB_ERROR)

  Message: Cannot proceed because system tables used by Event Scheduler were found damaged at server start

- Error: 1578 SQLSTATE: HY000 (ER_ONLY_INTEGERS_ALLOWED)

  Message: Only integers allowed as number here

- Error: 1579 SQLSTATE: HY000 (ER_UNSUPORTED_LOG_ENGINE)

  Message: This storage engine cannot be used for log tables"

- Error: 1580 SQLSTATE: HY000 (ER_BAD_LOG_STATEMENT)

  Message: You cannot '%s' a log table if logging is enabled

- Error: 1581 SQLSTATE: HY000 (ER_CANT_RENAME_LOG_TABLE)

  Message: Cannot rename '%s'. When logging enabled, rename to/from log table must rename two tables: the log table to an archive table and another table back to '%s'

- Error: 1582 SQLSTATE: 42000 (ER_WRONG_PARAMCOUNT_TO_NATIVE_FCT)

  Message: Incorrect parameter count in the call to native function '%s'

- Error: 1583 SQLSTATE: 42000 (ER_WRONG_PARAMETERS_TO_NATIVE_FCT)

  Message: Incorrect parameters in the call to native function '%s'

- Error: 1584 SQLSTATE: 42000 (ER_WRONG_PARAMETERS_TO_STORED_FCT)

Message: Incorrect parameters in the call to stored function %s

- Error: 1585 SQLSTATE: HY000 (ER_NATIVE_FCT_NAME_COLLISION)

  Message: This function '%s' has the same name as a native function

- Error: 1586 SQLSTATE: 23000 (ER_DUP_ENTRY_WITH_KEY_NAME)

  Message: Duplicate entry '%s' for key '%s'

  The format string for this error is also used with ER_DUP_ENTRY.

- Error: 1587 SQLSTATE: HY000 (ER_BINLOG_PURGE_EMFILE)

  Message: Too many files opened, please execute the command again

- Error: 1588 SQLSTATE: HY000 (ER_EVENT_CANNOT_CREATE_IN_THE_PAST)

  Message: Event execution time is in the past and ON COMPLETION NOT PRESERVE is set. The event was dropped immediately after creation.

- Error: 1589 SQLSTATE: HY000 (ER_EVENT_CANNOT_ALTER_IN_THE_PAST)

  Message: Event execution time is in the past and ON COMPLETION NOT PRESERVE is set. The event was not changed. Specify a time in the future.

- Error: 1590 SQLSTATE: HY000 (ER_SLAVE_INCIDENT)

  Message: The incident %s occured on the master. Message: %s

- Error: 1591 SQLSTATE: HY000 (ER_NO_PARTITION_FOR_GIVEN_VALUE_SILENT)

  Message: Table has no partition for some existing values

- Error: 1592 SQLSTATE: HY000 (ER_BINLOG_UNSAFE_STATEMENT)

  Message: Unsafe statement written to the binary log using statement format since BINLOG_FORMAT = STATEMENT. %s

- Error: 1593 SQLSTATE: HY000 (ER_SLAVE_FATAL_ERROR)

  Message: Fatal error: %s

- Error: 1594 SQLSTATE: HY000 (ER_SLAVE_RELAY_LOG_READ_FAILURE)

  Message: Relay log read failure: %s

- Error: 1595 SQLSTATE: HY000 (ER_SLAVE_RELAY_LOG_WRITE_FAILURE)

  Message: Relay log write failure: %s

- Error: 1596 SQLSTATE: HY000 (ER_SLAVE_CREATE_EVENT_FAILURE)

  Message: Failed to create %s

- Error: 1597 SQLSTATE: HY000 (ER_SLAVE_MASTER_COM_FAILURE)

  Message: Master command %s failed: %s

- Error: 1598 SQLSTATE: HY000 (ER_BINLOG_LOGGING_IMPOSSIBLE)

  Message: Binary logging not possible. Message: %s

- Error: 1599 SQLSTATE: HY000 (ER_VIEW_NO_CREATION_CTX)

  Message: View `%s`.`%s` has no creation context

- Error: 1600 SQLSTATE: HY000 (ER_VIEW_INVALID_CREATION_CTX)

  Message: Creation context of view `%s`.`%s' is invalid

- Error: 1601 SQLSTATE: HY000 (ER_SR_INVALID_CREATION_CTX)

  Message: Creation context of stored routine `%s`.`%s` is invalid

- Error: 1602 SQLSTATE: HY000 (ER_TRG_CORRUPTED_FILE)

  Message: Corrupted TRG file for table `%s`.`%s`

- Error: 1603 SQLSTATE: HY000 (ER_TRG_NO_CREATION_CTX)

  Message: Triggers for table `%s`.`%s` have no creation context

- Error: 1604 SQLSTATE: HY000 (ER_TRG_INVALID_CREATION_CTX)

  Message: Trigger creation context of table `%s`.`%s` is invalid

- Error: 1605 SQLSTATE: HY000 (ER_EVENT_INVALID_CREATION_CTX)

  Message: Creation context of event `%s`.`%s` is invalid

- Error: 1606 SQLSTATE: HY000 (ER_TRG_CANT_OPEN_TABLE)

  Message: Cannot open table for trigger `%s`.`%s`

- Error: 1607 SQLSTATE: HY000 (ER_CANT_CREATE_SROUTINE)

  Message: Cannot create stored routine `%s`. Check warnings

- Error: 1608 SQLSTATE: HY000 (ER_NEVER_USED)

  Message: Ambiguous slave modes combination. %s

- Error: 1609 SQLSTATE: HY000 (ER_NO_FORMAT_DESCRIPTION_EVENT_BEFORE_BINLOG_STATEMENT)

  Message: The BINLOG statement of type `%s` was not preceded by a format description BINLOG statement.

- Error: 1610 SQLSTATE: HY000 (ER_SLAVE_CORRUPT_EVENT)

  Message: Corrupted replication event was detected

- Error: 1611 SQLSTATE: HY000 (ER_LOAD_DATA_INVALID_COLUMN)

  Message: Invalid column reference (%s) in LOAD DATA

- Error: 1612 SQLSTATE: HY000 (ER_LOG_PURGE_NO_FILE)

Message: Being purged log %s was not found

- Error: 1613 SQLSTATE: XA106 (ER_XA_RBTIMEOUT)

  Message: XA_RBTIMEOUT: Transaction branch was rolled back: took too long

- Error: 1614 SQLSTATE: XA102 (ER_XA_RBDEADLOCK)

  Message: XA_RBDEADLOCK: Transaction branch was rolled back: deadlock was detected

- Error: 1615 SQLSTATE: HY000 (ER_NEED_REPREPARE)

  Message: Prepared statement needs to be re-prepared

- Error: 1616 SQLSTATE: HY000 (ER_DELAYED_NOT_SUPPORTED)

  Message: DELAYED option not supported for table '%s'

- Error: 1617 SQLSTATE: HY000 (WARN_NO_MASTER_INFO)

  Message: The master info structure does not exist

- Error: 1618 SQLSTATE: HY000 (WARN_OPTION_IGNORED)

  Message: <%s> option ignored

- Error: 1619 SQLSTATE: HY000 (WARN_PLUGIN_DELETE_BUILTIN)

  Message: Built-in plugins cannot be deleted

  WARN_PLUGIN_DELETE_BUILTIN was removed after 5.7.4.

- Error: 1619 SQLSTATE: HY000 (ER_PLUGIN_DELETE_BUILTIN)

  Message: Built-in plugins cannot be deleted

  ER_PLUGIN_DELETE_BUILTIN was introduced in 5.7.5.

- Error: 1620 SQLSTATE: HY000 (WARN_PLUGIN_BUSY)

  Message: Plugin is busy and will be uninstalled on shutdown

- Error: 1621 SQLSTATE: HY000 (ER_VARIABLE_IS_READONLY)

  Message: %s variable '%s' is read-only. Use SET %s to assign the value

- Error: 1622 SQLSTATE: HY000 (ER_WARN_ENGINE_TRANSACTION_ROLLBACK)

  Message: Storage engine %s does not support rollback for this statement. Transaction rolled back and must be restarted

- Error: 1623 SQLSTATE: HY000 (ER_SLAVE_HEARTBEAT_FAILURE)

  Message: Unexpected master's heartbeat data: %s

- Error: 1624 SQLSTATE: HY000 (ER_SLAVE_HEARTBEAT_VALUE_OUT_OF_RANGE)

  Message: The requested value for the heartbeat period is either negative or exceeds the maximum allowed (%s seconds).

- Error: 1625 SQLSTATE: HY000 (ER_NDB_REPLICATION_SCHEMA_ERROR)

  Message: Bad schema for mysql.ndb_replication table. Message: %s

- Error: 1626 SQLSTATE: HY000 (ER_CONFLICT_FN_PARSE_ERROR)

  Message: Error in parsing conflict function. Message: %s

- Error: 1627 SQLSTATE: HY000 (ER_EXCEPTIONS_WRITE_ERROR)

  Message: Write to exceptions table failed. Message: %s"

- Error: 1628 SQLSTATE: HY000 (ER_TOO_LONG_TABLE_COMMENT)

  Message: Comment for table '%s' is too long (max = %lu)

- Error: 1629 SQLSTATE: HY000 (ER_TOO_LONG_FIELD_COMMENT)

  Message: Comment for field '%s' is too long (max = %lu)

- Error: 1630 SQLSTATE: 42000 (ER_FUNC_INEXISTENT_NAME_COLLISION)

  Message: FUNCTION %s does not exist. Check the 'Function Name Parsing and Resolution' section in the Reference Manual

- Error: 1631 SQLSTATE: HY000 (ER_DATABASE_NAME)

  Message: Database

- Error: 1632 SQLSTATE: HY000 (ER_TABLE_NAME)

  Message: Table

- Error: 1633 SQLSTATE: HY000 (ER_PARTITION_NAME)

  Message: Partition

- Error: 1634 SQLSTATE: HY000 (ER_SUBPARTITION_NAME)

  Message: Subpartition

- Error: 1635 SQLSTATE: HY000 (ER_TEMPORARY_NAME)

  Message: Temporary

- Error: 1636 SQLSTATE: HY000 (ER_RENAMED_NAME)

  Message: Renamed

- Error: 1637 SQLSTATE: HY000 (ER_TOO_MANY_CONCURRENT_TRXS)

  Message: Too many active concurrent transactions

- Error: 1638 SQLSTATE: HY000 (WARN_NON_ASCII_SEPARATOR_NOT_IMPLEMENTED)

  Message: Non-ASCII separator arguments are not fully supported

- Error: 1639 SQLSTATE: HY000 (ER_DEBUG_SYNC_TIMEOUT)

  Message: debug sync point wait timed out

- Error: `1640` SQLSTATE: `HY000` (`ER_DEBUG_SYNC_HIT_LIMIT`)

  Message: debug sync point hit limit reached

- Error: `1641` SQLSTATE: `42000` (`ER_DUP_SIGNAL_SET`)

  Message: Duplicate condition information item '%s'

- Error: `1642` SQLSTATE: `01000` (`ER_SIGNAL_WARN`)

  Message: Unhandled user-defined warning condition

- Error: `1643` SQLSTATE: `02000` (`ER_SIGNAL_NOT_FOUND`)

  Message: Unhandled user-defined not found condition

- Error: `1644` SQLSTATE: `HY000` (`ER_SIGNAL_EXCEPTION`)

  Message: Unhandled user-defined exception condition

- Error: `1645` SQLSTATE: `0K000` (`ER_RESIGNAL_WITHOUT_ACTIVE_HANDLER`)

  Message: RESIGNAL when handler not active

- Error: `1646` SQLSTATE: `HY000` (`ER_SIGNAL_BAD_CONDITION_TYPE`)

  Message: SIGNAL/RESIGNAL can only use a CONDITION defined with SQLSTATE

- Error: `1647` SQLSTATE: `HY000` (`WARN_COND_ITEM_TRUNCATED`)

  Message: Data truncated for condition item '%s'

- Error: `1648` SQLSTATE: `HY000` (`ER_COND_ITEM_TOO_LONG`)

  Message: Data too long for condition item '%s'

- Error: `1649` SQLSTATE: `HY000` (`ER_UNKNOWN_LOCALE`)

  Message: Unknown locale: '%s'

- Error: `1650` SQLSTATE: `HY000` (`ER_SLAVE_IGNORE_SERVER_IDS`)

  Message: The requested server id %d clashes with the slave startup option --replicate-same-server-id

- Error: `1651` SQLSTATE: `HY000` (`ER_QUERY_CACHE_DISABLED`)

  Message: Query cache is disabled; restart the server with query_cache_type=1 to enable it

- Error: `1652` SQLSTATE: `HY000` (`ER_SAME_NAME_PARTITION_FIELD`)

  Message: Duplicate partition field name '%s'

- Error: `1653` SQLSTATE: `HY000` (`ER_PARTITION_COLUMN_LIST_ERROR`)

  Message: Inconsistency in usage of column lists for partitioning

- Error: `1654` SQLSTATE: `HY000` (`ER_WRONG_TYPE_COLUMN_VALUE_ERROR`)

  Message: Partition column values of incorrect type

- Error: 1655 SQLSTATE: HY000 (ER_TOO_MANY_PARTITION_FUNC_FIELDS_ERROR)

  Message: Too many fields in '%s'

- Error: 1656 SQLSTATE: HY000 (ER_MAXVALUE_IN_VALUES_IN)

  Message: Cannot use MAXVALUE as value in VALUES IN

- Error: 1657 SQLSTATE: HY000 (ER_TOO_MANY_VALUES_ERROR)

  Message: Cannot have more than one value for this type of %s partitioning

- Error: 1658 SQLSTATE: HY000 (ER_ROW_SINGLE_PARTITION_FIELD_ERROR)

  Message: Row expressions in VALUES IN only allowed for multi-field column partitioning

- Error: 1659 SQLSTATE: HY000 (ER_FIELD_TYPE_NOT_ALLOWED_AS_PARTITION_FIELD)

  Message: Field '%s' is of a not allowed type for this type of partitioning

- Error: 1660 SQLSTATE: HY000 (ER_PARTITION_FIELDS_TOO_LONG)

  Message: The total length of the partitioning fields is too large

- Error: 1661 SQLSTATE: HY000 (ER_BINLOG_ROW_ENGINE_AND_STMT_ENGINE)

  Message: Cannot execute statement: impossible to write to binary log since both row-incapable engines and statement-incapable engines are involved.

- Error: 1662 SQLSTATE: HY000 (ER_BINLOG_ROW_MODE_AND_STMT_ENGINE)

  Message: Cannot execute statement: impossible to write to binary log since BINLOG_FORMAT = ROW and at least one table uses a storage engine limited to statement-based logging.

- Error: 1663 SQLSTATE: HY000 (ER_BINLOG_UNSAFE_AND_STMT_ENGINE)

  Message: Cannot execute statement: impossible to write to binary log since statement is unsafe, storage engine is limited to statement-based logging, and BINLOG_FORMAT = MIXED. %s

- Error: 1664 SQLSTATE: HY000 (ER_BINLOG_ROW_INJECTION_AND_STMT_ENGINE)

  Message: Cannot execute statement: impossible to write to binary log since statement is in row format and at least one table uses a storage engine limited to statement-based logging.

- Error: 1665 SQLSTATE: HY000 (ER_BINLOG_STMT_MODE_AND_ROW_ENGINE)

  Message: Cannot execute statement: impossible to write to binary log since BINLOG_FORMAT = STATEMENT and at least one table uses a storage engine limited to row-based logging.%s

- Error: 1666 SQLSTATE: HY000 (ER_BINLOG_ROW_INJECTION_AND_STMT_MODE)

  Message: Cannot execute statement: impossible to write to binary log since statement is in row format and BINLOG_FORMAT = STATEMENT.

- Error: 1667 SQLSTATE: HY000 (ER_BINLOG_MULTIPLE_ENGINES_AND_SELF_LOGGING_ENGINE)

  Message: Cannot execute statement: impossible to write to binary log since more than one engine is involved and at least one engine is self-logging.

- Error: 1668 SQLSTATE: HY000 (ER_BINLOG_UNSAFE_LIMIT)

Message: The statement is unsafe because it uses a LIMIT clause. This is unsafe because the set of rows included cannot be predicted.

- Error: `1669` SQLSTATE: `HY000` (`ER_UNUSED4`)

  Message: The statement is unsafe because it uses INSERT DELAYED. This is unsafe because the times when rows are inserted cannot be predicted.

- Error: `1670` SQLSTATE: `HY000` (`ER_BINLOG_UNSAFE_SYSTEM_TABLE`)

  Message: The statement is unsafe because it uses the general log, slow query log, or performance_schema table(s). This is unsafe because system tables may differ on slaves.

- Error: `1671` SQLSTATE: `HY000` (`ER_BINLOG_UNSAFE_AUTOINC_COLUMNS`)

  Message: Statement is unsafe because it invokes a trigger or a stored function that inserts into an AUTO_INCREMENT column. Inserted values cannot be logged correctly.

- Error: `1672` SQLSTATE: `HY000` (`ER_BINLOG_UNSAFE_UDF`)

  Message: Statement is unsafe because it uses a UDF which may not return the same value on the slave.

- Error: `1673` SQLSTATE: `HY000` (`ER_BINLOG_UNSAFE_SYSTEM_VARIABLE`)

  Message: Statement is unsafe because it uses a system variable that may have a different value on the slave.

- Error: `1674` SQLSTATE: `HY000` (`ER_BINLOG_UNSAFE_SYSTEM_FUNCTION`)

  Message: Statement is unsafe because it uses a system function that may return a different value on the slave.

- Error: `1675` SQLSTATE: `HY000` (`ER_BINLOG_UNSAFE_NONTRANS_AFTER_TRANS`)

  Message: Statement is unsafe because it accesses a non-transactional table after accessing a transactional table within the same transaction.

- Error: `1676` SQLSTATE: `HY000` (`ER_MESSAGE_AND_STATEMENT`)

  Message: %s Statement: %s

- Error: `1677` SQLSTATE: `HY000` (`ER_SLAVE_CONVERSION_FAILED`)

  Message: Column %d of table '%s.%s' cannot be converted from type '%s' to type '%s'

- Error: `1678` SQLSTATE: `HY000` (`ER_SLAVE_CANT_CREATE_CONVERSION`)

  Message: Can't create conversion table for table '%s.%s'

- Error: `1679` SQLSTATE: `HY000` (`ER_INSIDE_TRANSACTION_PREVENTS_SWITCH_BINLOG_FORMAT`)

  Message: Cannot modify @@session.binlog_format inside a transaction

- Error: `1680` SQLSTATE: `HY000` (`ER_PATH_LENGTH`)

  Message: The path specified for %s is too long.

- Error: `1681` SQLSTATE: `HY000` (`ER_WARN_DEPRECATED_SYNTAX_NO_REPLACEMENT`)

Message: '%s' is deprecated and will be removed in a future release.

- Error: 1682 SQLSTATE: HY000 (ER_WRONG_NATIVE_TABLE_STRUCTURE)

  Message: Native table '%s'.'%s' has the wrong structure

- Error: 1683 SQLSTATE: HY000 (ER_WRONG_PERFSCHEMA_USAGE)

  Message: Invalid performance_schema usage.

- Error: 1684 SQLSTATE: HY000 (ER_WARN_I_S_SKIPPED_TABLE)

  Message: Table '%s'.'%s' was skipped since its definition is being modified by concurrent DDL statement

- Error: 1685 SQLSTATE: HY000 (ER_INSIDE_TRANSACTION_PREVENTS_SWITCH_BINLOG_DIRECT)

  Message: Cannot modify @@session.binlog_direct_non_transactional_updates inside a transaction

- Error: 1686 SQLSTATE: HY000 (ER_STORED_FUNCTION_PREVENTS_SWITCH_BINLOG_DIRECT)

  Message: Cannot change the binlog direct flag inside a stored function or trigger

- Error: 1687 SQLSTATE: 42000 (ER_SPATIAL_MUST_HAVE_GEOM_COL)

  Message: A SPATIAL index may only contain a geometrical type column

- Error: 1688 SQLSTATE: HY000 (ER_TOO_LONG_INDEX_COMMENT)

  Message: Comment for index '%s' is too long (max = %lu)

- Error: 1689 SQLSTATE: HY000 (ER_LOCK_ABORTED)

  Message: Wait on a lock was aborted due to a pending exclusive lock

- Error: 1690 SQLSTATE: 22003 (ER_DATA_OUT_OF_RANGE)

  Message: %s value is out of range in '%s'

- Error: 1691 SQLSTATE: HY000 (ER_WRONG_SPVAR_TYPE_IN_LIMIT)

  Message: A variable of a non-integer based type in LIMIT clause

- Error: 1692 SQLSTATE: HY000 (ER_BINLOG_UNSAFE_MULTIPLE_ENGINES_AND_SELF_LOGGING_ENGINE)

  Message: Mixing self-logging and non-self-logging engines in a statement is unsafe.

- Error: 1693 SQLSTATE: HY000 (ER_BINLOG_UNSAFE_MIXED_STATEMENT)

  Message: Statement accesses nontransactional table as well as transactional or temporary table, and writes to any of them.

- Error: 1694 SQLSTATE: HY000 (ER_INSIDE_TRANSACTION_PREVENTS_SWITCH_SQL_LOG_BIN)

  Message: Cannot modify @@session.sql_log_bin inside a transaction

- Error: 1695 SQLSTATE: HY000 (ER_STORED_FUNCTION_PREVENTS_SWITCH_SQL_LOG_BIN)

  Message: Cannot change the sql_log_bin inside a stored function or trigger

- Error: 1696 SQLSTATE: HY000 (ER_FAILED_READ_FROM_PAR_FILE)

  Message: Failed to read from the .par file

- Error: 1697 SQLSTATE: HY000 (ER_VALUES_IS_NOT_INT_TYPE_ERROR)

  Message: VALUES value for partition '%s' must have type INT

- Error: 1698 SQLSTATE: 28000 (ER_ACCESS_DENIED_NO_PASSWORD_ERROR)

  Message: Access denied for user '%s'@'%s'

- Error: 1699 SQLSTATE: HY000 (ER_SET_PASSWORD_AUTH_PLUGIN)

  Message: SET PASSWORD has no significance for users authenticating via plugins

- Error: 1700 SQLSTATE: HY000 (ER_GRANT_PLUGIN_USER_EXISTS)

  Message: GRANT with IDENTIFIED WITH is illegal because the user %-.*s already exists

- Error: 1701 SQLSTATE: 42000 (ER_TRUNCATE_ILLEGAL_FK)

  Message: Cannot truncate a table referenced in a foreign key constraint (%s)

- Error: 1702 SQLSTATE: HY000 (ER_PLUGIN_IS_PERMANENT)

  Message: Plugin '%s' is force_plus_permanent and can not be unloaded

- Error: 1703 SQLSTATE: HY000 (ER_SLAVE_HEARTBEAT_VALUE_OUT_OF_RANGE_MIN)

  Message: The requested value for the heartbeat period is less than 1 millisecond. The value is reset to 0, meaning that heartbeating will effectively be disabled.

- Error: 1704 SQLSTATE: HY000 (ER_SLAVE_HEARTBEAT_VALUE_OUT_OF_RANGE_MAX)

  Message: The requested value for the heartbeat period exceeds the value of `slave_net_timeout' seconds. A sensible value for the period should be less than the timeout.

- Error: 1705 SQLSTATE: HY000 (ER_STMT_CACHE_FULL)

  Message: Multi-row statements required more than 'max_binlog_stmt_cache_size' bytes of storage; increase this mysqld variable and try again

- Error: 1706 SQLSTATE: HY000 (ER_MULTI_UPDATE_KEY_CONFLICT)

  Message: Primary key/partition key update is not allowed since the table is updated both as '%s' and '%s'.

- Error: 1707 SQLSTATE: HY000 (ER_TABLE_NEEDS_REBUILD)

  Message: Table rebuild required. Please do "ALTER TABLE `%s` FORCE" or dump/reload to fix it!

- Error: 1708 SQLSTATE: HY000 (WARN_OPTION_BELOW_LIMIT)

  Message: The value of '%s' should be no less than the value of '%s'

- Error: 1709 SQLSTATE: HY000 (ER_INDEX_COLUMN_TOO_LONG)

  Message: Index column size too large. The maximum column size is %lu bytes.

- Error: 1710 SQLSTATE: HY000 (ER_ERROR_IN_TRIGGER_BODY)

  Message: Trigger '%s' has an error in its body: '%s'

- Error: 1711 SQLSTATE: HY000 (ER_ERROR_IN_UNKNOWN_TRIGGER_BODY)

  Message: Unknown trigger has an error in its body: '%s'

- Error: 1712 SQLSTATE: HY000 (ER_INDEX_CORRUPT)

  Message: Index %s is corrupted

- Error: 1713 SQLSTATE: HY000 (ER_UNDO_RECORD_TOO_BIG)

  Message: Undo log record is too big.

- Error: 1714 SQLSTATE: HY000 (ER_BINLOG_UNSAFE_INSERT_IGNORE_SELECT)

  Message: INSERT IGNORE... SELECT is unsafe because the order in which rows are retrieved by the SELECT determines which (if any) rows are ignored. This order cannot be predicted and may differ on master and the slave.

- Error: 1715 SQLSTATE: HY000 (ER_BINLOG_UNSAFE_INSERT_SELECT_UPDATE)

  Message: INSERT... SELECT... ON DUPLICATE KEY UPDATE is unsafe because the order in which rows are retrieved by the SELECT determines which (if any) rows are updated. This order cannot be predicted and may differ on master and the slave.

- Error: 1716 SQLSTATE: HY000 (ER_BINLOG_UNSAFE_REPLACE_SELECT)

  Message: REPLACE... SELECT is unsafe because the order in which rows are retrieved by the SELECT determines which (if any) rows are replaced. This order cannot be predicted and may differ on master and the slave.

- Error: 1717 SQLSTATE: HY000 (ER_BINLOG_UNSAFE_CREATE_IGNORE_SELECT)

  Message: CREATE... IGNORE SELECT is unsafe because the order in which rows are retrieved by the SELECT determines which (if any) rows are ignored. This order cannot be predicted and may differ on master and the slave.

- Error: 1718 SQLSTATE: HY000 (ER_BINLOG_UNSAFE_CREATE_REPLACE_SELECT)

  Message: CREATE... REPLACE SELECT is unsafe because the order in which rows are retrieved by the SELECT determines which (if any) rows are replaced. This order cannot be predicted and may differ on master and the slave.

- Error: 1719 SQLSTATE: HY000 (ER_BINLOG_UNSAFE_UPDATE_IGNORE)

  Message: UPDATE IGNORE is unsafe because the order in which rows are updated determines which (if any) rows are ignored. This order cannot be predicted and may differ on master and the slave.

- Error: 1720 SQLSTATE: HY000 (ER_PLUGIN_NO_UNINSTALL)

  Message: Plugin '%s' is marked as not dynamically uninstallable. You have to stop the server to uninstall it.

- Error: 1721 SQLSTATE: HY000 (ER_PLUGIN_NO_INSTALL)

  Message: Plugin '%s' is marked as not dynamically installable. You have to stop the server to install it.

- Error: 1722 SQLSTATE: HY000 (ER_BINLOG_UNSAFE_WRITE_AUTOINC_SELECT)

  Message: Statements writing to a table with an auto-increment column after selecting from another table are unsafe because the order in which rows are retrieved determines what (if any) rows will be written. This order cannot be predicted and may differ on master and the slave.

- Error: 1723 SQLSTATE: HY000 (ER_BINLOG_UNSAFE_CREATE_SELECT_AUTOINC)

  Message: CREATE TABLE... SELECT... on a table with an auto-increment column is unsafe because the order in which rows are retrieved by the SELECT determines which (if any) rows are inserted. This order cannot be predicted and may differ on master and the slave.

- Error: 1724 SQLSTATE: HY000 (ER_BINLOG_UNSAFE_INSERT_TWO_KEYS)

  Message: INSERT... ON DUPLICATE KEY UPDATE on a table with more than one UNIQUE KEY is unsafe

- Error: 1725 SQLSTATE: HY000 (ER_TABLE_IN_FK_CHECK)

  Message: Table is being used in foreign key check.

- Error: 1726 SQLSTATE: HY000 (ER_UNSUPPORTED_ENGINE)

  Message: Storage engine '%s' does not support system tables. [%s.%s]

- Error: 1727 SQLSTATE: HY000 (ER_BINLOG_UNSAFE_AUTOINC_NOT_FIRST)

  Message: INSERT into autoincrement field which is not the first part in the composed primary key is unsafe.

- Error: 1728 SQLSTATE: HY000 (ER_CANNOT_LOAD_FROM_TABLE_V2)

  Message: Cannot load from %s.%s. The table is probably corrupted

- Error: 1729 SQLSTATE: HY000 (ER_MASTER_DELAY_VALUE_OUT_OF_RANGE)

  Message: The requested value %s for the master delay exceeds the maximum %u

- Error: 1730 SQLSTATE: HY000 (ER_ONLY_FD_AND_RBR_EVENTS_ALLOWED_IN_BINLOG_STATEMENT)

  Message: Only Format_description_log_event and row events are allowed in BINLOG statements (but %s was provided)

- Error: 1731 SQLSTATE: HY000 (ER_PARTITION_EXCHANGE_DIFFERENT_OPTION)

  Message: Non matching attribute '%s' between partition and table

- Error: 1732 SQLSTATE: HY000 (ER_PARTITION_EXCHANGE_PART_TABLE)

  Message: Table to exchange with partition is partitioned: '%s'

- Error: 1733 SQLSTATE: HY000 (ER_PARTITION_EXCHANGE_TEMP_TABLE)

  Message: Table to exchange with partition is temporary: '%s'

- Error: 1734 SQLSTATE: HY000 (ER_PARTITION_INSTEAD_OF_SUBPARTITION)

  Message: Subpartitioned table, use subpartition instead of partition

- Error: 1735 SQLSTATE: HY000 (ER_UNKNOWN_PARTITION)

  Message: Unknown partition '%s' in table '%s'

- Error: 1736 SQLSTATE: HY000 (ER_TABLES_DIFFERENT_METADATA)

  Message: Tables have different definitions

- Error: 1737 SQLSTATE: HY000 (ER_ROW_DOES_NOT_MATCH_PARTITION)

  Message: Found a row that does not match the partition

- Error: 1738 SQLSTATE: HY000 (ER_BINLOG_CACHE_SIZE_GREATER_THAN_MAX)

  Message: Option binlog_cache_size (%lu) is greater than max_binlog_cache_size (%lu); setting binlog_cache_size equal to max_binlog_cache_size.

- Error: 1739 SQLSTATE: HY000 (ER_WARN_INDEX_NOT_APPLICABLE)

  Message: Cannot use %s access on index '%s' due to type or collation conversion on field '%s'

- Error: 1740 SQLSTATE: HY000 (ER_PARTITION_EXCHANGE_FOREIGN_KEY)

  Message: Table to exchange with partition has foreign key references: '%s'

- Error: 1741 SQLSTATE: HY000 (ER_NO_SUCH_KEY_VALUE)

  Message: Key value '%s' was not found in table '%s.%s'

- Error: 1742 SQLSTATE: HY000 (ER_RPL_INFO_DATA_TOO_LONG)

  Message: Data for column '%s' too long

- Error: 1743 SQLSTATE: HY000 (ER_NETWORK_READ_EVENT_CHECKSUM_FAILURE)

  Message: Replication event checksum verification failed while reading from network.

- Error: 1744 SQLSTATE: HY000 (ER_BINLOG_READ_EVENT_CHECKSUM_FAILURE)

  Message: Replication event checksum verification failed while reading from a log file.

- Error: 1745 SQLSTATE: HY000 (ER_BINLOG_STMT_CACHE_SIZE_GREATER_THAN_MAX)

  Message: Option binlog_stmt_cache_size (%lu) is greater than max_binlog_stmt_cache_size (%lu); setting binlog_stmt_cache_size equal to max_binlog_stmt_cache_size.

- Error: 1746 SQLSTATE: HY000 (ER_CANT_UPDATE_TABLE_IN_CREATE_TABLE_SELECT)

  Message: Can't update table '%s' while '%s' is being created.

- Error: 1747 SQLSTATE: HY000 (ER_PARTITION_CLAUSE_ON_NONPARTITIONED)

  Message: PARTITION () clause on non partitioned table

- Error: 1748 SQLSTATE: HY000 (ER_ROW_DOES_NOT_MATCH_GIVEN_PARTITION_SET)

  Message: Found a row not matching the given partition set

- Error: 1749 SQLSTATE: HY000 (ER_NO_SUCH_PARTITION__UNUSED)

Message: partition '%s' doesn't exist

- Error: 1750 SQLSTATE: HY000 (ER_CHANGE_RPL_INFO_REPOSITORY_FAILURE)

  Message: Failure while changing the type of replication repository: %s.

- Error: 1751 SQLSTATE: HY000 (ER_WARNING_NOT_COMPLETE_ROLLBACK_WITH_CREATED_TEMP_TABLE)

  Message: The creation of some temporary tables could not be rolled back.

- Error: 1752 SQLSTATE: HY000 (ER_WARNING_NOT_COMPLETE_ROLLBACK_WITH_DROPPED_TEMP_TABLE)

  Message: Some temporary tables were dropped, but these operations could not be rolled back.

- Error: 1753 SQLSTATE: HY000 (ER_MTS_FEATURE_IS_NOT_SUPPORTED)

  Message: %s is not supported in multi-threaded slave mode. %s

- Error: 1754 SQLSTATE: HY000 (ER_MTS_UPDATED_DBS_GREATER_MAX)

  Message: The number of modified databases exceeds the maximum %d; the database names will not be included in the replication event metadata.

- Error: 1755 SQLSTATE: HY000 (ER_MTS_CANT_PARALLEL)

  Message: Cannot execute the current event group in the parallel mode. Encountered event %s, relay-log name %s, position %s which prevents execution of this event group in parallel mode. Reason: %s.

- Error: 1756 SQLSTATE: HY000 (ER_MTS_INCONSISTENT_DATA)

  Message: %s

- Error: 1757 SQLSTATE: HY000 (ER_FULLTEXT_NOT_SUPPORTED_WITH_PARTITIONING)

  Message: FULLTEXT index is not supported for partitioned tables.

- Error: 1758 SQLSTATE: 35000 (ER_DA_INVALID_CONDITION_NUMBER)

  Message: Invalid condition number

- Error: 1759 SQLSTATE: HY000 (ER_INSECURE_PLAIN_TEXT)

  Message: Sending passwords in plain text without SSL/TLS is extremely insecure.

- Error: 1760 SQLSTATE: HY000 (ER_INSECURE_CHANGE_MASTER)

  Message: Storing MySQL user name or password information in the master info repository is not secure and is therefore not recommended. Please consider using the USER and PASSWORD connection options for START SLAVE; see the 'START SLAVE Syntax' in the MySQL Manual for more information.

- Error: 1761 SQLSTATE: 23000 (ER_FOREIGN_DUPLICATE_KEY_WITH_CHILD_INFO)

  Message: Foreign key constraint for table '%s', record '%s' would lead to a duplicate entry in table '%s', key '%s'

- Error: 1762 SQLSTATE: 23000 (ER_FOREIGN_DUPLICATE_KEY_WITHOUT_CHILD_INFO)

Message: Foreign key constraint for table '%s', record '%s' would lead to a duplicate entry in a child table

- Error: 1763 SQLSTATE: HY000 (ER_SQLTHREAD_WITH_SECURE_SLAVE)

  Message: Setting authentication options is not possible when only the Slave SQL Thread is being started.

- Error: 1764 SQLSTATE: HY000 (ER_TABLE_HAS_NO_FT)

  Message: The table does not have FULLTEXT index to support this query

- Error: 1765 SQLSTATE: HY000 (ER_VARIABLE_NOT_SETTABLE_IN_SF_OR_TRIGGER)

  Message: The system variable %s cannot be set in stored functions or triggers.

- Error: 1766 SQLSTATE: HY000 (ER_VARIABLE_NOT_SETTABLE_IN_TRANSACTION)

  Message: The system variable %s cannot be set when there is an ongoing transaction.

- Error: 1767 SQLSTATE: HY000 (ER_GTID_NEXT_IS_NOT_IN_GTID_NEXT_LIST)

  Message: The system variable @@SESSION.GTID_NEXT has the value %s, which is not listed in @@SESSION.GTID_NEXT_LIST.

- Error: 1768 SQLSTATE: HY000 (ER_CANT_CHANGE_GTID_NEXT_IN_TRANSACTION_WHEN_GTID_NEXT_LIST_IS_NULL)

  Message: The system variable @@SESSION.GTID_NEXT cannot change inside a transaction.

- Error: 1769 SQLSTATE: HY000 (ER_SET_STATEMENT_CANNOT_INVOKE_FUNCTION)

  Message: The statement 'SET %s' cannot invoke a stored function.

- Error: 1770 SQLSTATE: HY000 (ER_GTID_NEXT_CANT_BE_AUTOMATIC_IF_GTID_NEXT_LIST_IS_NON_NULL)

  Message: The system variable @@SESSION.GTID_NEXT cannot be 'AUTOMATIC' when @@SESSION.GTID_NEXT_LIST is non-NULL.

- Error: 1771 SQLSTATE: HY000 (ER_SKIPPING_LOGGED_TRANSACTION)

  Message: Skipping transaction %s because it has already been executed and logged.

- Error: 1772 SQLSTATE: HY000 (ER_MALFORMED_GTID_SET_SPECIFICATION)

  Message: Malformed GTID set specification '%s'.

- Error: 1773 SQLSTATE: HY000 (ER_MALFORMED_GTID_SET_ENCODING)

  Message: Malformed GTID set encoding.

- Error: 1774 SQLSTATE: HY000 (ER_MALFORMED_GTID_SPECIFICATION)

  Message: Malformed GTID specification '%s'.

- Error: 1775 SQLSTATE: HY000 (ER_GNO_EXHAUSTED)

  Message: Impossible to generate Global Transaction Identifier: the integer component reached the maximal value. Restart the server with a new server_uuid.

- Error: 1776 SQLSTATE: HY000 (ER_BAD_SLAVE_AUTO_POSITION)

  Message: Parameters MASTER_LOG_FILE, MASTER_LOG_POS, RELAY_LOG_FILE and RELAY_LOG_POS cannot be set when MASTER_AUTO_POSITION is active.

- Error: 1777 SQLSTATE: HY000 (ER_AUTO_POSITION_REQUIRES_GTID_MODE_ON)

  Message: CHANGE MASTER TO MASTER_AUTO_POSITION = 1 can only be executed when @@GLOBAL.GTID_MODE = ON.

- Error: 1778 SQLSTATE: HY000 (ER_CANT_DO_IMPLICIT_COMMIT_IN_TRX_WHEN_GTID_NEXT_IS_SET)

  Message: Cannot execute statements with implicit commit inside a transaction when @@SESSION.GTID_NEXT == 'UUID:NUMBER'.

- Error: 1779 SQLSTATE: HY000 (ER_GTID_MODE_2_OR_3_REQUIRES_ENFORCE_GTID_CONSISTENCY_ON)

  Message: @@GLOBAL.GTID_MODE = ON or UPGRADE_STEP_2 requires @@GLOBAL.ENFORCE_GTID_CONSISTENCY = 1.

- Error: 1780 SQLSTATE: HY000 (ER_GTID_MODE_REQUIRES_BINLOG)

  Message: @@GLOBAL.GTID_MODE = ON or UPGRADE_STEP_1 or UPGRADE_STEP_2 requires --log-bin and --log-slave-updates.

- Error: 1781 SQLSTATE: HY000 (ER_CANT_SET_GTID_NEXT_TO_GTID_WHEN_GTID_MODE_IS_OFF)

  Message: @@SESSION.GTID_NEXT cannot be set to UUID:NUMBER when @@GLOBAL.GTID_MODE = OFF.

- Error: 1782 SQLSTATE: HY000 (ER_CANT_SET_GTID_NEXT_TO_ANONYMOUS_WHEN_GTID_MODE_IS_ON)

  Message: @@SESSION.GTID_NEXT cannot be set to ANONYMOUS when @@GLOBAL.GTID_MODE = ON.

- Error: 1783 SQLSTATE: HY000 (ER_CANT_SET_GTID_NEXT_LIST_TO_NON_NULL_WHEN_GTID_MODE_IS_OFF)

  Message: @@SESSION.GTID_NEXT_LIST cannot be set to a non-NULL value when @@GLOBAL.GTID_MODE = OFF.

- Error: 1784 SQLSTATE: HY000 (ER_FOUND_GTID_EVENT_WHEN_GTID_MODE_IS_OFF)

  Message: Found a Gtid_log_event or Previous_gtids_log_event when @@GLOBAL.GTID_MODE = OFF.

- Error: 1785 SQLSTATE: HY000 (ER_GTID_UNSAFE_NON_TRANSACTIONAL_TABLE)

  Message: When @@GLOBAL.ENFORCE_GTID_CONSISTENCY = 1, updates to non-transactional tables can only be done in either autocommitted statements or single-statement transactions, and never in the same statement as updates to transactional tables.

- Error: 1786 SQLSTATE: HY000 (ER_GTID_UNSAFE_CREATE_SELECT)

  Message: CREATE TABLE ... SELECT is forbidden when @@GLOBAL.ENFORCE_GTID_CONSISTENCY = 1.

- Error: 1787 SQLSTATE: HY000 (ER_GTID_UNSAFE_CREATE_DROP_TEMPORARY_TABLE_IN_TRANSACTION)

  Message: When @@GLOBAL.ENFORCE_GTID_CONSISTENCY = 1, the statements CREATE TEMPORARY TABLE and DROP TEMPORARY TABLE can be executed in a non-transactional context only, and require that AUTOCOMMIT = 1.

- Error: 1788 SQLSTATE: HY000 (ER_GTID_MODE_CAN_ONLY_CHANGE_ONE_STEP_AT_A_TIME)

  Message: The value of @@GLOBAL.GTID_MODE can only change one step at a time: OFF <-> UPGRADE_STEP_1 <-> UPGRADE_STEP_2 <-> ON. Also note that this value must be stepped up or down simultaneously on all servers; see the Manual for instructions.

- Error: 1789 SQLSTATE: HY000 (ER_MASTER_HAS_PURGED_REQUIRED_GTIDS)

  Message: The slave is connecting using CHANGE MASTER TO MASTER_AUTO_POSITION = 1, but the master has purged binary logs containing GTIDs that the slave requires.

- Error: 1790 SQLSTATE: HY000 (ER_CANT_SET_GTID_NEXT_WHEN_OWNING_GTID)

  Message: @@SESSION.GTID_NEXT cannot be changed by a client that owns a GTID. The client owns %s. Ownership is released on COMMIT or ROLLBACK.

- Error: 1791 SQLSTATE: HY000 (ER_UNKNOWN_EXPLAIN_FORMAT)

  Message: Unknown EXPLAIN format name: '%s'

- Error: 1792 SQLSTATE: 25006 (ER_CANT_EXECUTE_IN_READ_ONLY_TRANSACTION)

  Message: Cannot execute statement in a READ ONLY transaction.

- Error: 1793 SQLSTATE: HY000 (ER_TOO_LONG_TABLE_PARTITION_COMMENT)

  Message: Comment for table partition '%s' is too long (max = %lu)

- Error: 1794 SQLSTATE: HY000 (ER_SLAVE_CONFIGURATION)

  Message: Slave is not configured or failed to initialize properly. You must at least set --server-id to enable either a master or a slave. Additional error messages can be found in the MySQL error log.

- Error: 1795 SQLSTATE: HY000 (ER_INNODB_FT_LIMIT)

  Message: InnoDB presently supports one FULLTEXT index creation at a time

- Error: 1796 SQLSTATE: HY000 (ER_INNODB_NO_FT_TEMP_TABLE)

  Message: Cannot create FULLTEXT index on temporary InnoDB table

- Error: 1797 SQLSTATE: HY000 (ER_INNODB_FT_WRONG_DOCID_COLUMN)

  Message: Column '%s' is of wrong type for an InnoDB FULLTEXT index

- Error: 1798 SQLSTATE: HY000 (ER_INNODB_FT_WRONG_DOCID_INDEX)

  Message: Index '%s' is of wrong type for an InnoDB FULLTEXT index

- Error: 1799 SQLSTATE: HY000 (ER_INNODB_ONLINE_LOG_TOO_BIG)

  Message: Creating index '%s' required more than 'innodb_online_alter_log_max_size' bytes of modification log. Please try again.

- Error: 1800 SQLSTATE: HY000 (ER_UNKNOWN_ALTER_ALGORITHM)

  Message: Unknown ALGORITHM '%s'

- Error: 1801 SQLSTATE: HY000 (ER_UNKNOWN_ALTER_LOCK)

  Message: Unknown LOCK type '%s'

- Error: 1802 SQLSTATE: HY000 (ER_MTS_CHANGE_MASTER_CANT_RUN_WITH_GAPS)

  Message: CHANGE MASTER cannot be executed when the slave was stopped with an error or killed in MTS mode. Consider using RESET SLAVE or START SLAVE UNTIL.

- Error: 1803 SQLSTATE: HY000 (ER_MTS_RECOVERY_FAILURE)

  Message: Cannot recover after SLAVE errored out in parallel execution mode. Additional error messages can be found in the MySQL error log.

- Error: 1804 SQLSTATE: HY000 (ER_MTS_RESET_WORKERS)

  Message: Cannot clean up worker info tables. Additional error messages can be found in the MySQL error log.

- Error: 1805 SQLSTATE: HY000 (ER_COL_COUNT_DOESNT_MATCH_CORRUPTED_V2)

  Message: Column count of %s.%s is wrong. Expected %d, found %d. The table is probably corrupted

- Error: 1806 SQLSTATE: HY000 (ER_SLAVE_SILENT_RETRY_TRANSACTION)

  Message: Slave must silently retry current transaction

- Error: 1807 SQLSTATE: HY000 (ER_DISCARD_FK_CHECKS_RUNNING)

  Message: There is a foreign key check running on table '%s'. Cannot discard the table.

- Error: 1808 SQLSTATE: HY000 (ER_TABLE_SCHEMA_MISMATCH)

  Message: Schema mismatch (%s)

- Error: 1809 SQLSTATE: HY000 (ER_TABLE_IN_SYSTEM_TABLESPACE)

  Message: Table '%s' in system tablespace

- Error: 1810 SQLSTATE: HY000 (ER_IO_READ_ERROR)

  Message: IO Read error: (%lu, %s) %s

- Error: 1811 SQLSTATE: HY000 (ER_IO_WRITE_ERROR)

  Message: IO Write error: (%lu, %s) %s

- Error: 1812 SQLSTATE: HY000 (ER_TABLESPACE_MISSING)

  Message: Tablespace is missing for table '%s'

- Error: 1813 SQLSTATE: HY000 (ER_TABLESPACE_EXISTS)

  Message: Tablespace for table '%s' exists. Please DISCARD the tablespace before IMPORT.

- Error: 1814 SQLSTATE: HY000 (ER_TABLESPACE_DISCARDED)

Message: Tablespace has been discarded for table '%s'

- Error: 1815 SQLSTATE: HY000 (ER_INTERNAL_ERROR)

  Message: Internal error: %s

- Error: 1816 SQLSTATE: HY000 (ER_INNODB_IMPORT_ERROR)

  Message: ALTER TABLE '%s' IMPORT TABLESPACE failed with error %lu : '%s'

- Error: 1817 SQLSTATE: HY000 (ER_INNODB_INDEX_CORRUPT)

  Message: Index corrupt: %s

- Error: 1818 SQLSTATE: HY000 (ER_INVALID_YEAR_COLUMN_LENGTH)

  Message: YEAR(%lu) column type is deprecated. Creating YEAR(4) column instead.

- Error: 1819 SQLSTATE: HY000 (ER_NOT_VALID_PASSWORD)

  Message: Your password does not satisfy the current policy requirements

- Error: 1820 SQLSTATE: HY000 (ER_MUST_CHANGE_PASSWORD)

  Message: You must SET PASSWORD before executing this statement

- Error: 1821 SQLSTATE: HY000 (ER_FK_NO_INDEX_CHILD)

  Message: Failed to add the foreign key constaint. Missing index for constraint '%s' in the foreign table '%s'

- Error: 1822 SQLSTATE: HY000 (ER_FK_NO_INDEX_PARENT)

  Message: Failed to add the foreign key constaint. Missing index for constraint '%s' in the referenced table '%s'

- Error: 1823 SQLSTATE: HY000 (ER_FK_FAIL_ADD_SYSTEM)

  Message: Failed to add the foreign key constraint '%s' to system tables

- Error: 1824 SQLSTATE: HY000 (ER_FK_CANNOT_OPEN_PARENT)

  Message: Failed to open the referenced table '%s'

- Error: 1825 SQLSTATE: HY000 (ER_FK_INCORRECT_OPTION)

  Message: Failed to add the foreign key constraint on table '%s'. Incorrect options in FOREIGN KEY constraint '%s'

- Error: 1826 SQLSTATE: HY000 (ER_FK_DUP_NAME)

  Message: Duplicate foreign key constraint name '%s'

- Error: 1827 SQLSTATE: HY000 (ER_PASSWORD_FORMAT)

  Message: The password hash doesn't have the expected format. Check if the correct password algorithm is being used with the PASSWORD() function.

- Error: 1828 SQLSTATE: HY000 (ER_FK_COLUMN_CANNOT_DROP)

Message: Cannot drop column '%s': needed in a foreign key constraint '%s'

- Error: 1829 SQLSTATE: HY000 (ER_FK_COLUMN_CANNOT_DROP_CHILD)

  Message: Cannot drop column '%s': needed in a foreign key constraint '%s' of table '%s'

- Error: 1830 SQLSTATE: HY000 (ER_FK_COLUMN_NOT_NULL)

  Message: Column '%s' cannot be NOT NULL: needed in a foreign key constraint '%s' SET NULL

- Error: 1831 SQLSTATE: HY000 (ER_DUP_INDEX)

  Message: Duplicate index '%s' defined on the table '%s.%s'. This is deprecated and will be disallowed in a future release.

- Error: 1832 SQLSTATE: HY000 (ER_FK_COLUMN_CANNOT_CHANGE)

  Message: Cannot change column '%s': used in a foreign key constraint '%s'

- Error: 1833 SQLSTATE: HY000 (ER_FK_COLUMN_CANNOT_CHANGE_CHILD)

  Message: Cannot change column '%s': used in a foreign key constraint '%s' of table '%s'

- Error: 1834 SQLSTATE: HY000 (ER_FK_CANNOT_DELETE_PARENT)

  Message: Cannot delete rows from table which is parent in a foreign key constraint '%s' of table '%s'

  ER_FK_CANNOT_DELETE_PARENT was removed after 5.7.3.

- Error: 1834 SQLSTATE: HY000 (ER_UNUSED5)

  Message: Cannot delete rows from table which is parent in a foreign key constraint '%s' of table '%s'

  ER_UNUSED5 was introduced in 5.7.4.

- Error: 1835 SQLSTATE: HY000 (ER_MALFORMED_PACKET)

  Message: Malformed communication packet.

- Error: 1836 SQLSTATE: HY000 (ER_READ_ONLY_MODE)

  Message: Running in read-only mode

- Error: 1837 SQLSTATE: HY000 (ER_GTID_NEXT_TYPE_UNDEFINED_GROUP)

  Message: When @@SESSION.GTID_NEXT is set to a GTID, you must explicitly set it to a different value after a COMMIT or ROLLBACK. Please check GTID_NEXT variable manual page for detailed explanation. Current @@SESSION.GTID_NEXT is '%s'.

- Error: 1838 SQLSTATE: HY000 (ER_VARIABLE_NOT_SETTABLE_IN_SP)

  Message: The system variable %s cannot be set in stored procedures.

- Error: 1839 SQLSTATE: HY000 (ER_CANT_SET_GTID_PURGED_WHEN_GTID_MODE_IS_OFF)

  Message: @@GLOBAL.GTID_PURGED can only be set when @@GLOBAL.GTID_MODE = ON.

- Error: 1840 SQLSTATE: HY000 (ER_CANT_SET_GTID_PURGED_WHEN_GTID_EXECUTED_IS_NOT_EMPTY)

Message: @@GLOBAL.GTID_PURGED can only be set when @@GLOBAL.GTID_EXECUTED is empty.

- Error: 1841 SQLSTATE: HY000 (ER_CANT_SET_GTID_PURGED_WHEN_OWNED_GTIDS_IS_NOT_EMPTY)

  Message: @@GLOBAL.GTID_PURGED can only be set when there are no ongoing transactions (not even in other clients).

- Error: 1842 SQLSTATE: HY000 (ER_GTID_PURGED_WAS_CHANGED)

  Message: @@GLOBAL.GTID_PURGED was changed from '%s' to '%s'.

- Error: 1843 SQLSTATE: HY000 (ER_GTID_EXECUTED_WAS_CHANGED)

  Message: @@GLOBAL.GTID_EXECUTED was changed from '%s' to '%s'.

- Error: 1844 SQLSTATE: HY000 (ER_BINLOG_STMT_MODE_AND_NO_REPL_TABLES)

  Message: Cannot execute statement: impossible to write to binary log since BINLOG_FORMAT = STATEMENT, and both replicated and non replicated tables are written to.

- Error: 1845 SQLSTATE: 0A000 (ER_ALTER_OPERATION_NOT_SUPPORTED)

  Message: %s is not supported for this operation. Try %s.

  ER_ALTER_OPERATION_NOT_SUPPORTED was introduced in 5.7.1.

- Error: 1846 SQLSTATE: 0A000 (ER_ALTER_OPERATION_NOT_SUPPORTED_REASON)

  Message: %s is not supported. Reason: %s. Try %s.

  ER_ALTER_OPERATION_NOT_SUPPORTED_REASON was introduced in 5.7.1.

- Error: 1847 SQLSTATE: HY000 (ER_ALTER_OPERATION_NOT_SUPPORTED_REASON_COPY)

  Message: COPY algorithm requires a lock

  ER_ALTER_OPERATION_NOT_SUPPORTED_REASON_COPY was introduced in 5.7.1.

- Error: 1848 SQLSTATE: HY000 (ER_ALTER_OPERATION_NOT_SUPPORTED_REASON_PARTITION)

  Message: Partition specific operations do not yet support LOCK/ALGORITHM

  ER_ALTER_OPERATION_NOT_SUPPORTED_REASON_PARTITION was introduced in 5.7.1.

- Error: 1849 SQLSTATE: HY000 (ER_ALTER_OPERATION_NOT_SUPPORTED_REASON_FK_RENAME)

  Message: Columns participating in a foreign key are renamed

  ER_ALTER_OPERATION_NOT_SUPPORTED_REASON_FK_RENAME was introduced in 5.7.1.

- Error: 1850 SQLSTATE: HY000 (ER_ALTER_OPERATION_NOT_SUPPORTED_REASON_COLUMN_TYPE)

  Message: Cannot change column type INPLACE

  ER_ALTER_OPERATION_NOT_SUPPORTED_REASON_COLUMN_TYPE was introduced in 5.7.1.

- Error: 1851 SQLSTATE: HY000 (ER_ALTER_OPERATION_NOT_SUPPORTED_REASON_FK_CHECK)

Message: Adding foreign keys needs foreign_key_checks=OFF

ER_ALTER_OPERATION_NOT_SUPPORTED_REASON_FK_CHECK was introduced in 5.7.1.

- Error: 1852 SQLSTATE: HY000 (ER_ALTER_OPERATION_NOT_SUPPORTED_REASON_IGNORE)

  Message: Creating unique indexes with IGNORE requires COPY algorithm to remove duplicate rows

  ER_ALTER_OPERATION_NOT_SUPPORTED_REASON_IGNORE was introduced in 5.7.1, removed after 5.7.3.

- Error: 1852 SQLSTATE: HY000 (ER_UNUSED6)

  Message: Creating unique indexes with IGNORE requires COPY algorithm to remove duplicate rows

  ER_UNUSED6 was introduced in 5.7.4.

- Error: 1853 SQLSTATE: HY000 (ER_ALTER_OPERATION_NOT_SUPPORTED_REASON_NOPK)

  Message: Dropping a primary key is not allowed without also adding a new primary key

  ER_ALTER_OPERATION_NOT_SUPPORTED_REASON_NOPK was introduced in 5.7.1.

- Error: 1854 SQLSTATE: HY000 (ER_ALTER_OPERATION_NOT_SUPPORTED_REASON_AUTOINC)

  Message: Adding an auto-increment column requires a lock

  ER_ALTER_OPERATION_NOT_SUPPORTED_REASON_AUTOINC was introduced in 5.7.1.

- Error: 1855 SQLSTATE: HY000 (ER_ALTER_OPERATION_NOT_SUPPORTED_REASON_HIDDEN_FTS)

  Message: Cannot replace hidden FTS_DOC_ID with a user-visible one

  ER_ALTER_OPERATION_NOT_SUPPORTED_REASON_HIDDEN_FTS was introduced in 5.7.1.

- Error: 1856 SQLSTATE: HY000 (ER_ALTER_OPERATION_NOT_SUPPORTED_REASON_CHANGE_FTS)

  Message: Cannot drop or rename FTS_DOC_ID

  ER_ALTER_OPERATION_NOT_SUPPORTED_REASON_CHANGE_FTS was introduced in 5.7.1.

- Error: 1857 SQLSTATE: HY000 (ER_ALTER_OPERATION_NOT_SUPPORTED_REASON_FTS)

  Message: Fulltext index creation requires a lock

  ER_ALTER_OPERATION_NOT_SUPPORTED_REASON_FTS was introduced in 5.7.1.

- Error: 1858 SQLSTATE: HY000 (ER_SQL_SLAVE_SKIP_COUNTER_NOT_SETTABLE_IN_GTID_MODE)

  Message: sql_slave_skip_counter can not be set when the server is running with @@GLOBAL.GTID_MODE = ON. Instead, for each transaction that you want to skip, generate an empty transaction with the same GTID as the transaction

  ER_SQL_SLAVE_SKIP_COUNTER_NOT_SETTABLE_IN_GTID_MODE was introduced in 5.7.1.

- Error: 1859 SQLSTATE: 23000 (ER_DUP_UNKNOWN_IN_INDEX)

  Message: Duplicate entry for key '%s'

ER_DUP_UNKNOWN_IN_INDEX was introduced in 5.7.1.

- Error: 1860 SQLSTATE: HY000 (ER_IDENT_CAUSES_TOO_LONG_PATH)

  Message: Long database name and identifier for object resulted in path length exceeding %d characters. Path: '%s'.

  ER_IDENT_CAUSES_TOO_LONG_PATH was introduced in 5.7.1.

- Error: 1861 SQLSTATE: HY000 (ER_ALTER_OPERATION_NOT_SUPPORTED_REASON_NOT_NULL)

  Message: cannot silently convert NULL values, as required in this SQL_MODE

  ER_ALTER_OPERATION_NOT_SUPPORTED_REASON_NOT_NULL was introduced in 5.7.1.

- Error: 1862 SQLSTATE: HY000 (ER_MUST_CHANGE_PASSWORD_LOGIN)

  Message: Your password has expired. To log in you must change it using a client that supports expired passwords.

  ER_MUST_CHANGE_PASSWORD_LOGIN was introduced in 5.7.1.

- Error: 1863 SQLSTATE: HY000 (ER_ROW_IN_WRONG_PARTITION)

  Message: Found a row in wrong partition %s

  ER_ROW_IN_WRONG_PARTITION was introduced in 5.7.1.

- Error: 1864 SQLSTATE: HY000 (ER_MTS_EVENT_BIGGER_PENDING_JOBS_SIZE_MAX)

  Message: Cannot schedule event %s, relay-log name %s, position %s to Worker thread because its size %lu exceeds %lu of slave_pending_jobs_size_max.

  ER_MTS_EVENT_BIGGER_PENDING_JOBS_SIZE_MAX was introduced in 5.7.2.

- Error: 1865 SQLSTATE: HY000 (ER_INNODB_NO_FT_USES_PARSER)

  Message: Cannot CREATE FULLTEXT INDEX WITH PARSER on InnoDB table

  ER_INNODB_NO_FT_USES_PARSER was introduced in 5.7.2.

- Error: 1866 SQLSTATE: HY000 (ER_BINLOG_LOGICAL_CORRUPTION)

  Message: The binary log file '%s' is logically corrupted: %s

  ER_BINLOG_LOGICAL_CORRUPTION was introduced in 5.7.2.

- Error: 1867 SQLSTATE: HY000 (ER_WARN_PURGE_LOG_IN_USE)

  Message: file %s was not purged because it was being read by %d thread(s), purged only %d out of %d files.

  ER_WARN_PURGE_LOG_IN_USE was introduced in 5.7.2.

- Error: 1868 SQLSTATE: HY000 (ER_WARN_PURGE_LOG_IS_ACTIVE)

  Message: file %s was not purged because it is the active log file.

  ER_WARN_PURGE_LOG_IS_ACTIVE was introduced in 5.7.2.

- Error: 1869 SQLSTATE: HY000 (ER_AUTO_INCREMENT_CONFLICT)

  Message: Auto-increment value in UPDATE conflicts with internally generated values

  ER_AUTO_INCREMENT_CONFLICT was introduced in 5.7.2.

- Error: 1870 SQLSTATE: HY000 (WARN_ON_BLOCKHOLE_IN_RBR)

  Message: Row events are not logged for %s statements that modify BLACKHOLE tables in row format. Table(s): '%s'

  WARN_ON_BLOCKHOLE_IN_RBR was introduced in 5.7.2.

- Error: 1871 SQLSTATE: HY000 (ER_SLAVE_MI_INIT_REPOSITORY)

  Message: Slave failed to initialize master info structure from the repository

  ER_SLAVE_MI_INIT_REPOSITORY was introduced in 5.7.2.

- Error: 1872 SQLSTATE: HY000 (ER_SLAVE_RLI_INIT_REPOSITORY)

  Message: Slave failed to initialize relay log info structure from the repository

  ER_SLAVE_RLI_INIT_REPOSITORY was introduced in 5.7.2.

- Error: 1873 SQLSTATE: 28000 (ER_ACCESS_DENIED_CHANGE_USER_ERROR)

  Message: Access denied trying to change to user '%s'@'%s' (using password: %s). Disconnecting.

  ER_ACCESS_DENIED_CHANGE_USER_ERROR was introduced in 5.7.2.

- Error: 1874 SQLSTATE: HY000 (ER_INNODB_READ_ONLY)

  Message: InnoDB is in read only mode.

  ER_INNODB_READ_ONLY was introduced in 5.7.2.

- Error: 1875 SQLSTATE: HY000 (ER_STOP_SLAVE_SQL_THREAD_TIMEOUT)

  Message: STOP SLAVE command execution is incomplete: Slave SQL thread got the stop signal, thread is busy, SQL thread will stop once the current task is complete.

  ER_STOP_SLAVE_SQL_THREAD_TIMEOUT was introduced in 5.7.2.

- Error: 1876 SQLSTATE: HY000 (ER_STOP_SLAVE_IO_THREAD_TIMEOUT)

  Message: STOP SLAVE command execution is incomplete: Slave IO thread got the stop signal, thread is busy, IO thread will stop once the current task is complete.

  ER_STOP_SLAVE_IO_THREAD_TIMEOUT was introduced in 5.7.2.

- Error: 1877 SQLSTATE: HY000 (ER_TABLE_CORRUPT)

  Message: Operation cannot be performed. The table '%s.%s' is missing, corrupt or contains bad data.

  ER_TABLE_CORRUPT was introduced in 5.7.2.

- Error: 1878 SQLSTATE: HY000 (ER_TEMP_FILE_WRITE_FAILURE)

  Message: Temporary file write failure.

ER_TEMP_FILE_WRITE_FAILURE was introduced in 5.7.3.

- Error: 1879 SQLSTATE: HY000 (ER_INNODB_FT_AUX_NOT_HEX_ID)

  Message: Upgrade index name failed, please use create index(alter table) algorithm copy to rebuild index.

  ER_INNODB_FT_AUX_NOT_HEX_ID was introduced in 5.7.4.

- Error: 1880 SQLSTATE: HY000 (ER_OLD_TEMPORALS_UPGRADED)

  Message: TIME/TIMESTAMP/DATETIME columns of old format have been upgraded to the new format.

  ER_OLD_TEMPORALS_UPGRADED was introduced in 5.7.4.

- Error: 1881 SQLSTATE: HY000 (ER_INNODB_FORCED_RECOVERY)

  Message: Operation not allowed when innodb_forced_recovery > 0.

  ER_INNODB_FORCED_RECOVERY was introduced in 5.7.4.

- Error: 1882 SQLSTATE: HY000 (ER_AES_INVALID_IV)

  Message: The initialization vector supplied to %s is too short. Must be at least %d bytes long

  ER_AES_INVALID_IV was introduced in 5.7.4.

- Error: 1883 SQLSTATE: HY000 (ER_FILE_CORRUPT)

  Message: File %s is corrupted

- Error: 1884 SQLSTATE: HY000 (ER_ERROR_ON_MASTER)

  Message: Query partially completed on the master (error on master: %d) and was aborted. There is a chance that your master is inconsistent at this point. If you are sure that your master is ok, run this query manually on the slave and then restart the slave with SET GLOBAL SQL_SLAVE_SKIP_COUNTER=1; START SLAVE;. Query:'%s'

- Error: 1885 SQLSTATE: HY000 (ER_INCONSISTENT_ERROR)

  Message: Query caused different errors on master and slave. Error on master: message (format)='%s' error code=%d; Error on slave:actual message='%s', error code=%d. Default database:'%s'. Query:'%s'

- Error: 1886 SQLSTATE: HY000 (ER_STORAGE_ENGINE_NOT_LOADED)

  Message: Storage engine for table '%s'.'%s' is not loaded.

- Error: 1887 SQLSTATE: 0Z002 (ER_GET_STACKED_DA_WITHOUT_ACTIVE_HANDLER)

  Message: GET STACKED DIAGNOSTICS when handler not active

- Error: 1888 SQLSTATE: HY000 (ER_WARN_LEGACY_SYNTAX_CONVERTED)

  Message: %s is no longer supported. The statement was converted to %s.

- Error: 1889 SQLSTATE: HY000 (ER_BINLOG_UNSAFE_FULLTEXT_PLUGIN)

  Message: Statement is unsafe because it uses a fulltext parser plugin which may not return the same value on the slave.

ER_BINLOG_UNSAFE_FULLTEXT_PLUGIN was introduced in 5.7.1.

- Error: 1890 SQLSTATE: HY000 (ER_CANNOT_DISCARD_TEMPORARY_TABLE)

  Message: Cannot DISCARD/IMPORT tablespace associated with temporary table

  ER_CANNOT_DISCARD_TEMPORARY_TABLE was introduced in 5.7.1.

- Error: 1891 SQLSTATE: HY000 (ER_FK_DEPTH_EXCEEDED)

  Message: Foreign key cascade delete/update exceeds max depth of %d.

  ER_FK_DEPTH_EXCEEDED was introduced in 5.7.2.

- Error: 1892 SQLSTATE: HY000 (ER_COL_COUNT_DOESNT_MATCH_PLEASE_UPDATE_V2)

  Message: Column count of %s.%s is wrong. Expected %d, found %d. Created with MySQL %d, now running %d. Please use mysql_upgrade to fix this error.

  ER_COL_COUNT_DOESNT_MATCH_PLEASE_UPDATE_V2 was introduced in 5.7.2.

- Error: 1893 SQLSTATE: HY000 (ER_WARN_TRIGGER_DOESNT_HAVE_CREATED)

  Message: Trigger %s.%s.%s does not have CREATED attribute.

  ER_WARN_TRIGGER_DOESNT_HAVE_CREATED was introduced in 5.7.2.

- Error: 1894 SQLSTATE: HY000 (ER_REFERENCED_TRG_DOES_NOT_EXIST)

  Message: Referenced trigger '%s' for the given action time and event type does not exist.

  ER_REFERENCED_TRG_DOES_NOT_EXIST was introduced in 5.7.2.

- Error: 1895 SQLSTATE: HY000 (ER_EXPLAIN_NOT_SUPPORTED)

  Message: EXPLAIN FOR CONNECTION command is supported only for SELECT/UPDATE/INSERT/DELETE/REPLACE

  ER_EXPLAIN_NOT_SUPPORTED was introduced in 5.7.2.

- Error: 1896 SQLSTATE: HY000 (ER_INVALID_FIELD_SIZE)

  Message: Invalid size for column '%s'.

  ER_INVALID_FIELD_SIZE was introduced in 5.7.2.

- Error: 1897 SQLSTATE: HY000 (ER_MISSING_HA_CREATE_OPTION)

  Message: Table storage engine '%s' found required create option missing

  ER_MISSING_HA_CREATE_OPTION was introduced in 5.7.2.

- Error: 1898 SQLSTATE: HY000 (ER_ENGINE_OUT_OF_MEMORY)

  Message: Out of memory in storage engine '%s'.

  ER_ENGINE_OUT_OF_MEMORY was introduced in 5.7.3.

- Error: 1899 SQLSTATE: HY000 (ER_PASSWORD_EXPIRE_ANONYMOUS_USER)

Message: The password for anonymous user cannot be expired.

ER_PASSWORD_EXPIRE_ANONYMOUS_USER was introduced in 5.7.3.

- Error: 1900 SQLSTATE: HY000 (ER_SLAVE_SQL_THREAD_MUST_STOP)

  Message: This operation cannot be performed with a running slave sql thread; run STOP SLAVE SQL_THREAD first

  ER_SLAVE_SQL_THREAD_MUST_STOP was introduced in 5.7.3.

- Error: 1901 SQLSTATE: HY000 (ER_NO_FT_MATERIALIZED_SUBQUERY)

  Message: Cannot create FULLTEXT index on materialized subquery

  ER_NO_FT_MATERIALIZED_SUBQUERY was introduced in 5.7.4.

- Error: 1902 SQLSTATE: HY000 (ER_INNODB_UNDO_LOG_FULL)

  Message: Undo Log error: %s

  ER_INNODB_UNDO_LOG_FULL was introduced in 5.7.4.

- Error: 1903 SQLSTATE: 2201E (ER_INVALID_ARGUMENT_FOR_LOGARITHM)

  Message: Invalid argument for logarithm

  ER_INVALID_ARGUMENT_FOR_LOGARITHM was introduced in 5.7.4.

- Error: 1904 SQLSTATE: HY000 (ER_SLAVE_IO_THREAD_MUST_STOP)

  Message: This operation cannot be performed with a running slave io thread; run STOP SLAVE IO_THREAD first.

  ER_SLAVE_IO_THREAD_MUST_STOP was introduced in 5.7.4.

- Error: 1905 SQLSTATE: HY000 (ER_WARN_OPEN_TEMP_TABLES_MUST_BE_ZERO)

  Message: This operation may not be safe when the slave has temporary tables. The tables will be kept open until the server restarts or until the tables are deleted by any replicated DROP statement. Suggest to wait until slave_open_temp_tables = 0.

  ER_WARN_OPEN_TEMP_TABLES_MUST_BE_ZERO was introduced in 5.7.4.

- Error: 1906 SQLSTATE: HY000 (ER_WARN_ONLY_MASTER_LOG_FILE_NO_POS)

  Message: CHANGE MASTER TO with a MASTER_LOG_FILE clause but no MASTER_LOG_POS clause may not be safe. The old position value may not be valid for the new binary log file.

  ER_WARN_ONLY_MASTER_LOG_FILE_NO_POS was introduced in 5.7.4.

- Error: 1907 SQLSTATE: HY000 (ER_QUERY_TIMEOUT)

  Message: Query execution was interrupted, max_statement_time exceeded

  ER_QUERY_TIMEOUT was introduced in 5.7.4.

- Error: 1908 SQLSTATE: HY000 (ER_NON_RO_SELECT_DISABLE_TIMER)

Message: Select is not a read only statement, disabling timer

ER_NON_RO_SELECT_DISABLE_TIMER was introduced in 5.7.4.

- Error: 1909 SQLSTATE: HY000 (ER_DUP_LIST_ENTRY)

  Message: Duplicate entry '%s'.

  ER_DUP_LIST_ENTRY was introduced in 5.7.4.

- Error: 1910 SQLSTATE: HY000 (ER_SQL_MODE_NO_EFFECT)

  Message: '%s' mode no longer has any effect. Use STRICT_ALL_TABLES or STRICT_TRANS_TABLES instead.

  ER_SQL_MODE_NO_EFFECT was introduced in 5.7.4.

C.4 Client Error Codes and Messages

Client error information comes from the following source files:

- The Error values and the symbols in parentheses correspond to definitions in the include/errmsg.h MySQL source file.
- The Message values correspond to the error messages that are listed in the libmysql/errmsg.c file. %d and %s represent numbers and strings, respectively, that are substituted into the messages when they are displayed.

Because updates are frequent, it is possible that those files will contain additional error information not listed here.

- Error: 2000 (CR_UNKNOWN_ERROR)

  Message: Unknown MySQL error

- Error: 2001 (CR_SOCKET_CREATE_ERROR)

  Message: Can't create UNIX socket (%d)

- Error: 2002 (CR_CONNECTION_ERROR)

  Message: Can't connect to local MySQL server through socket '%s' (%d)

- Error: 2003 (CR_CONN_HOST_ERROR)

  Message: Can't connect to MySQL server on '%s' (%d)

- Error: 2004 (CR_IPSOCK_ERROR)

  Message: Can't create TCP/IP socket (%d)

- Error: 2005 (CR_UNKNOWN_HOST)

  Message: Unknown MySQL server host '%s' (%d)

- Error: 2006 (CR_SERVER_GONE_ERROR)

  Message: MySQL server has gone away

- Error: `2007` (`CR_VERSION_ERROR`)

  Message: Protocol mismatch; server version = %d, client version = %d

- Error: `2008` (`CR_OUT_OF_MEMORY`)

  Message: MySQL client ran out of memory

- Error: `2009` (`CR_WRONG_HOST_INFO`)

  Message: Wrong host info

- Error: `2010` (`CR_LOCALHOST_CONNECTION`)

  Message: Localhost via UNIX socket

- Error: `2011` (`CR_TCP_CONNECTION`)

  Message: %s via TCP/IP

- Error: `2012` (`CR_SERVER_HANDSHAKE_ERR`)

  Message: Error in server handshake

- Error: `2013` (`CR_SERVER_LOST`)

  Message: Lost connection to MySQL server during query

- Error: `2014` (`CR_COMMANDS_OUT_OF_SYNC`)

  Message: Commands out of sync; you can't run this command now

- Error: `2015` (`CR_NAMEDPIPE_CONNECTION`)

  Message: Named pipe: %s

- Error: `2016` (`CR_NAMEDPIPEWAIT_ERROR`)

  Message: Can't wait for named pipe to host: %s pipe: %s (%lu)

- Error: `2017` (`CR_NAMEDPIPEOPEN_ERROR`)

  Message: Can't open named pipe to host: %s pipe: %s (%lu)

- Error: `2018` (`CR_NAMEDPIPESETSTATE_ERROR`)

  Message: Can't set state of named pipe to host: %s pipe: %s (%lu)

- Error: `2019` (`CR_CANT_READ_CHARSET`)

  Message: Can't initialize character set %s (path: %s)

- Error: `2020` (`CR_NET_PACKET_TOO_LARGE`)

  Message: Got packet bigger than 'max_allowed_packet' bytes

- Error: `2021` (`CR_EMBEDDED_CONNECTION`)

  Message: Embedded server

- Error: `2022` (`CR_PROBE_SLAVE_STATUS`)

  Message: Error on SHOW SLAVE STATUS:

- Error: `2023` (`CR_PROBE_SLAVE_HOSTS`)

  Message: Error on SHOW SLAVE HOSTS:

- Error: `2024` (`CR_PROBE_SLAVE_CONNECT`)

  Message: Error connecting to slave:

- Error: `2025` (`CR_PROBE_MASTER_CONNECT`)

  Message: Error connecting to master:

- Error: `2026` (`CR_SSL_CONNECTION_ERROR`)

  Message: SSL connection error: %s

- Error: `2027` (`CR_MALFORMED_PACKET`)

  Message: Malformed packet

- Error: `2028` (`CR_WRONG_LICENSE`)

  Message: This client library is licensed only for use with MySQL servers having '%s' license

- Error: `2029` (`CR_NULL_POINTER`)

  Message: Invalid use of null pointer

- Error: `2030` (`CR_NO_PREPARE_STMT`)

  Message: Statement not prepared

- Error: `2031` (`CR_PARAMS_NOT_BOUND`)

  Message: No data supplied for parameters in prepared statement

- Error: `2032` (`CR_DATA_TRUNCATED`)

  Message: Data truncated

- Error: `2033` (`CR_NO_PARAMETERS_EXISTS`)

  Message: No parameters exist in the statement

- Error: `2034` (`CR_INVALID_PARAMETER_NO`)

  Message: Invalid parameter number

- Error: `2035` (`CR_INVALID_BUFFER_USE`)

  Message: Can't send long data for non-string/non-binary data types (parameter: %d)

- Error: `2036` (`CR_UNSUPPORTED_PARAM_TYPE`)

  Message: Using unsupported buffer type: %d (parameter: %d)

- Error: `2037` (`CR_SHARED_MEMORY_CONNECTION`)

  Message: Shared memory: %s

- Error: `2038` (`CR_SHARED_MEMORY_CONNECT_REQUEST_ERROR`)

  Message: Can't open shared memory; client could not create request event (%lu)

- Error: `2039` (`CR_SHARED_MEMORY_CONNECT_ANSWER_ERROR`)

  Message: Can't open shared memory; no answer event received from server (%lu)

- Error: `2040` (`CR_SHARED_MEMORY_CONNECT_FILE_MAP_ERROR`)

  Message: Can't open shared memory; server could not allocate file mapping (%lu)

- Error: `2041` (`CR_SHARED_MEMORY_CONNECT_MAP_ERROR`)

  Message: Can't open shared memory; server could not get pointer to file mapping (%lu)

- Error: `2042` (`CR_SHARED_MEMORY_FILE_MAP_ERROR`)

  Message: Can't open shared memory; client could not allocate file mapping (%lu)

- Error: `2043` (`CR_SHARED_MEMORY_MAP_ERROR`)

  Message: Can't open shared memory; client could not get pointer to file mapping (%lu)

- Error: `2044` (`CR_SHARED_MEMORY_EVENT_ERROR`)

  Message: Can't open shared memory; client could not create %s event (%lu)

- Error: `2045` (`CR_SHARED_MEMORY_CONNECT_ABANDONED_ERROR`)

  Message: Can't open shared memory; no answer from server (%lu)

- Error: `2046` (`CR_SHARED_MEMORY_CONNECT_SET_ERROR`)

  Message: Can't open shared memory; cannot send request event to server (%lu)

- Error: `2047` (`CR_CONN_UNKNOW_PROTOCOL`)

  Message: Wrong or unknown protocol

- Error: `2048` (`CR_INVALID_CONN_HANDLE`)

  Message: Invalid connection handle

- Error: `2049` (`CR_SECURE_AUTH`)

  Message: Connection using old (pre-4.1.1) authentication protocol refused (client option 'secure_auth' enabled)

- Error: `2050` (`CR_FETCH_CANCELED`)

  Message: Row retrieval was canceled by mysql_stmt_close() call

- Error: `2051` (`CR_NO_DATA`)

  Message: Attempt to read column without prior row fetch

- Error: `2052` (`CR_NO_STMT_METADATA`)

  Message: Prepared statement contains no metadata

- Error: `2053` (`CR_NO_RESULT_SET`)

  Message: Attempt to read a row while there is no result set associated with the statement

- Error: `2054` (`CR_NOT_IMPLEMENTED`)

  Message: This feature is not implemented yet

- Error: `2055` (`CR_SERVER_LOST_EXTENDED`)

  Message: Lost connection to MySQL server at '%s', system error: %d

- Error: `2056` (`CR_STMT_CLOSED`)

  Message: Statement closed indirectly because of a preceeding %s() call

- Error: `2057` (`CR_NEW_STMT_METADATA`)

  Message: The number of columns in the result set differs from the number of bound buffers. You must reset the statement, rebind the result set columns, and execute the statement again

- Error: `2058` (`CR_ALREADY_CONNECTED`)

  Message: This handle is already connected. Use a separate handle for each connection.

- Error: `2059` (`CR_AUTH_PLUGIN_CANNOT_LOAD`)

  Message: Authentication plugin '%s' cannot be loaded: %s

- Error: `2060` (`CR_DUPLICATE_CONNECTION_ATTR`)

  Message: There is an attribute with the same name already

- Error: `2061` (`CR_AUTH_PLUGIN_ERR`)

  Message: Authentication plugin '%s' reported error: %s

  `CR_AUTH_PLUGIN_ERR` was introduced in 5.7.1.

C.5 Problems and Common Errors

This section lists some common problems and error messages that you may encounter. It describes how to determine the causes of the problems and what to do to solve them.

C.5.1 How to Determine What Is Causing a Problem

When you run into a problem, the first thing you should do is to find out which program or piece of equipment is causing it:

- If you have one of the following symptoms, then it is probably a hardware problems (such as memory, motherboard, CPU, or hard disk) or kernel problem:

  - The keyboard does not work. This can normally be checked by pressing the Caps Lock key. If the Caps Lock light does not change, you have to replace your keyboard. (Before doing this, you should try to restart your computer and check all cables to the keyboard.)

  - The mouse pointer does not move.
  - The machine does not answer to a remote machine's pings.
  - Other programs that are not related to MySQL do not behave correctly.
  - Your system restarted unexpectedly. (A faulty user-level program should never be able to take down your system.)

  In this case, you should start by checking all your cables and run some diagnostic tool to check your hardware! You should also check whether there are any patches, updates, or service packs for your operating system that could likely solve your problem. Check also that all your libraries (such as `glibc`) are up to date.

  It is always good to use a machine with ECC memory to discover memory problems early.

- If your keyboard is locked up, you may be able to recover by logging in to your machine from another machine and executing `kbd_mode -a`.
- Please examine your system log file (`/var/log/messages` or similar) for reasons for your problem. If you think the problem is in MySQL, you should also examine MySQL's log files. See Section 5.2, “MySQL Server Logs”.
- If you do not think you have hardware problems, you should try to find out which program is causing problems. Try using `top`, `ps`, Task Manager, or some similar program, to check which program is taking all CPU or is locking the machine.
- Use `top`, `df`, or a similar program to check whether you are out of memory, disk space, file descriptors, or some other critical resource.
- If the problem is some runaway process, you can always try to kill it. If it does not want to die, there is probably a bug in the operating system.

If after you have examined all other possibilities and you have concluded that the MySQL server or a MySQL client is causing the problem, it is time to create a bug report for our mailing list or our support team. In the bug report, try to give a very detailed description of how the system is behaving and what you think is happening. You should also state why you think that MySQL is causing the problem. Take into consideration all the situations in this chapter. State any problems exactly how they appear when you examine your system. Use the “copy and paste” method for any output and error messages from programs and log files.

Try to describe in detail which program is not working and all symptoms you see. We have in the past received many bug reports that state only “the system does not work.” This provides us with no information about what could be the problem.

If a program fails, it is always useful to know the following information:

- Has the program in question made a segmentation fault (did it dump core)?
- Is the program taking up all available CPU time? Check with `top`. Let the program run for a while, it may simply be evaluating something computationally intensive.
- If the `mysqld` server is causing problems, can you get any response from it with `mysqladmin -u root ping` or `mysqladmin -u root processlist`?
- What does a client program say when you try to connect to the MySQL server? (Try with `mysql`, for example.) Does the client jam? Do you get any output from the program?

When sending a bug report, you should follow the outline described in Section 1.7, “How to Report Bugs or Problems”.

C.5.2 Common Errors When Using MySQL Programs

This section lists some errors that users frequently encounter when running MySQL programs. Although the problems show up when you try to run client programs, the solutions to many of the problems involves changing the configuration of the MySQL server.

C.5.2.1 Access denied

An `Access denied` error can have many causes. Often the problem is related to the MySQL accounts that the server permits client programs to use when connecting. See Section 6.2, “The MySQL Access Privilege System”, and Section 6.2.7, “Causes of Access-Denied Errors”.

C.5.2.2 Can't connect to [local] MySQL server

A MySQL client on Unix can connect to the `mysqld` server in two different ways: By using a Unix socket file to connect through a file in the file system (default `/tmp/mysql.sock`), or by using TCP/IP, which connects through a port number. A Unix socket file connection is faster than TCP/IP, but can be used only when connecting to a server on the same computer. A Unix socket file is used if you do not specify a host name or if you specify the special host name `localhost`.

If the MySQL server is running on Windows, you can connect using TCP/IP. If the server is started with the `--enable-named-pipe` option, you can also connect with named pipes if you run the client on the host where the server is running. The name of the named pipe is `MySQL` by default. If you do not give a host name when connecting to `mysqld`, a MySQL client first tries to connect to the named pipe. If that does not work, it connects to the TCP/IP port. You can force the use of named pipes on Windows by using `.` as the host name.

The error (2002) `Can't connect to ...` normally means that there is no MySQL server running on the system or that you are using an incorrect Unix socket file name or TCP/IP port number when trying to connect to the server. You should also check that the TCP/IP port you are using has not been blocked by a firewall or port blocking service.

The error (2003) `Can't connect to MySQL server on 'server' (10061)` indicates that the network connection has been refused. You should check that there is a MySQL server running, that it has network connections enabled, and that the network port you specified is the one configured on the server.

Start by checking whether there is a process named `mysqld` running on your server host. (Use `ps xa | grep mysqld` on Unix or the Task Manager on Windows.) If there is no such process, you should start the server. See Section 2.9.1.3, “Starting and Troubleshooting the MySQL Server”.

If a `mysqld` process is running, you can check it by trying the following commands. The port number or Unix socket file name might be different in your setup. `host_ip` represents the IP address of the machine where the server is running.

```
shell> mysqladmin version
shell> mysqladmin variables
shell> mysqladmin -h `hostname` version variables
shell> mysqladmin -h `hostname` --port=3306 version
shell> mysqladmin -h host_ip version
shell> mysqladmin --protocol=SOCKET --socket=/tmp/mysql.sock version
```

Note the use of backticks rather than forward quotation marks with the `hostname` command; these cause the output of `hostname` (that is, the current host name) to be substituted into the `mysqladmin` command.

If you have no hostname command or are running on Windows, you can manually type the host name of your machine (without backticks) following the -h option. You can also try -h 127.0.0.1 to connect with TCP/IP to the local host.

Make sure that the server has not been configured to ignore network connections or (if you are attempting to connect remotely) that it has not been configured to listen only locally on its network interfaces. If the server was started with --skip-networking, it will not accept TCP/IP connections at all. If the server was started with --bind-address=127.0.0.1, it will listen for TCP/IP connections only locally on the loopback interface and will not accept remote connections.

Check to make sure that there is no firewall blocking access to MySQL. Your firewall may be configured on the basis of the application being executed, or the port number used by MySQL for communication (3306 by default). Under Linux or Unix, check your IP tables (or similar) configuration to ensure that the port has not been blocked. Under Windows, applications such as ZoneAlarm or the Windows XP personal firewall may need to be configured not to block the MySQL port.

Here are some reasons the Can't connect to local MySQL server error might occur:

- mysqld is not running on the local host. Check your operating system's process list to ensure the mysqld process is present.

- You're running a MySQL server on Windows with many TCP/IP connections to it. If you're experiencing that quite often your clients get that error, you can find a workaround here: Connection to MySQL Server Failing on Windows.

- Someone has removed the Unix socket file that mysqld uses (/tmp/mysql.sock by default). For example, you might have a cron job that removes old files from the /tmp directory. You can always run mysqladmin version to check whether the Unix socket file that mysqladmin is trying to use really exists. The fix in this case is to change the cron job to not remove mysql.sock or to place the socket file somewhere else. See Section C.5.4.5, “How to Protect or Change the MySQL Unix Socket File”.

- You have started the mysqld server with the --socket=/path/to/socket option, but forgotten to tell client programs the new name of the socket file. If you change the socket path name for the server, you must also notify the MySQL clients. You can do this by providing the same --socket option when you run client programs. You also need to ensure that clients have permission to access the mysql.sock file. To find out where the socket file is, you can do:

  ```
  shell> netstat -ln | grep mysql
  ```

  See Section C.5.4.5, “How to Protect or Change the MySQL Unix Socket File”.

- You are using Linux and one server thread has died (dumped core). In this case, you must kill the other mysqld threads (for example, with kill or with the mysql_zap script) before you can restart the MySQL server. See Section C.5.4.2, “What to Do If MySQL Keeps Crashing”.

- The server or client program might not have the proper access privileges for the directory that holds the Unix socket file or the socket file itself. In this case, you must either change the access privileges for the directory or socket file so that the server and clients can access them, or restart mysqld with a --socket option that specifies a socket file name in a directory where the server can create it and where client programs can access it.

If you get the error message Can't connect to MySQL server on some_host, you can try the following things to find out what the problem is:

- Check whether the server is running on that host by executing telnet some_host 3306 and pressing the Enter key a couple of times. (3306 is the default MySQL port number. Change the value if your server is listening to a different port.) If there is a MySQL server running and listening to the port, you should get a response that includes the server's version number. If you get an error such as `telnet: Unable to connect to remote host: Connection refused`, then there is no server running on the given port.

- If the server is running on the local host, try using `mysqladmin -h localhost variables` to connect using the Unix socket file. Verify the TCP/IP port number that the server is configured to listen to (it is the value of the `port` variable.)

- If you are running under Linux and Security-Enhanced Linux (SELinux) is enabled, make sure you have disabled SELinux protection for the `mysqld` process.

Connection to MySQL Server Failing on Windows

When you're running a MySQL server on Windows with many TCP/IP connections to it, and you're experiencing that quite often your clients get a `Can't connect to MySQL server` error, the reason might be that Windows does not allow for enough ephemeral (short-lived) ports to serve those connections.

The purpose of `TIME_WAIT` is to keep a connection accepting packets even after the connection has been closed. This is because Internet routing can cause a packet to take a slow route to its destination and it may arrive after both sides have agreed to close. If the port is in use for a new connection, that packet from the old connection could break the protocol or compromise personal information from the original connection. The `TIME_WAIT` delay prevents this by ensuring that the port cannot be reused until after some time has been permitted for those delayed packets to arrive.

It is safe to reduce `TIME_WAIT` greatly on LAN connections because there is little chance of packets arriving at very long delays, as they could through the Internet with its comparatively large distances and latencies.

Windows permits ephemeral (short-lived) TCP ports to the user. After any port is closed it will remain in a `TIME_WAIT` status for 120 seconds. The port will not be available again until this time expires. The default range of port numbers depends on the version of Windows, with a more limited number of ports in older versions:

- Windows through Server 2003: Ports in range 1025–5000

- Windows Vista, Server 2008, and newer: Ports in range 49152–65535

With a small stack of available TCP ports (5000) and a high number of TCP ports being open and closed over a short period of time along with the `TIME_WAIT` status you have a good chance for running out of ports. There are two ways to address this problem:

- Reduce the number of TCP ports consumed quickly by investigating connection pooling or persistent connections where possible

- Tune some settings in the Windows registry (see below)

IMPORTANT: The following procedure involves modifying the Windows registry. Before you modify the registry, make sure to back it up and make sure that you understand how to restore the registry if a problem occurs. For information about how to back up, restore, and edit the registry, view the following article in the Microsoft Knowledge Base: http://support.microsoft.com/kb/256986/EN-US/.

1. Start Registry Editor (`Regedt32.exe`).

2. Locate the following key in the registry:

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters
```

3. On the `Edit` menu, click `Add Value`, and then add the following registry value:

   ```
   Value Name: MaxUserPort
   Data Type: REG_DWORD
   Value: 65534
   ```

   This sets the number of ephemeral ports available to any user. The valid range is between 5000 and 65534 (decimal). The default value is 0x1388 (5000 decimal).

4. On the `Edit` menu, click `Add Value`, and then add the following registry value:

   ```
   Value Name: TcpTimedWaitDelay
   Data Type: REG_DWORD
   Value: 30
   ```

   This sets the number of seconds to hold a TCP port connection in `TIME_WAIT` state before closing. The valid range is between 0 (zero) and 300 (decimal). The default value is 0x78 (120 decimal).

5. Quit Registry Editor.

6. Reboot the machine.

Note: Undoing the above should be as simple as deleting the registry entries you've created.

C.5.2.3 `Lost connection to MySQL server`

There are three likely causes for this error message.

Usually it indicates network connectivity trouble and you should check the condition of your network if this error occurs frequently. If the error message includes “during query,” this is probably the case you are experiencing.

Sometimes the “during query” form happens when millions of rows are being sent as part of one or more queries. If you know that this is happening, you should try increasing `net_read_timeout` from its default of 30 seconds to 60 seconds or longer, sufficient for the data transfer to complete.

More rarely, it can happen when the client is attempting the initial connection to the server. In this case, if your `connect_timeout` value is set to only a few seconds, you may be able to resolve the problem by increasing it to ten seconds, perhaps more if you have a very long distance or slow connection. You can determine whether you are experiencing this more uncommon cause by using `SHOW GLOBAL STATUS LIKE 'Aborted_connects'`. It will increase by one for each initial connection attempt that the server aborts. You may see “reading authorization packet” as part of the error message; if so, that also suggests that this is the solution that you need.

If the cause is none of those just described, you may be experiencing a problem with `BLOB` values that are larger than `max_allowed_packet`, which can cause this error with some clients. Sometime you may see an `ER_NET_PACKET_TOO_LARGE` error, and that confirms that you need to increase `max_allowed_packet`.

C.5.2.4 `Client does not support authentication protocol`

The current implementation of the authentication protocol uses a password hashing algorithm that is incompatible with that used by older (pre-4.1) clients. Attempts to connect to a 4.1 or newer server with an older client may fail with the following message:

```
shell> mysql
Client does not support authentication protocol requested
```

```
by server; consider upgrading MySQL client
```

To deal with this problem, the preferred solution is to upgrade all client programs to use a 4.1.1 or newer client library. If that is not possible, use one of the following approaches:

- To connect to the server with a pre-4.1 client program, use an account that still has a pre-4.1-style password.
- Reset the password to pre-4.1 style for each user that needs to use a pre-4.1 client program. This can be done using the SET PASSWORD statement and the OLD_PASSWORD() function. It is also necessary to first ensure that the authentication plugin for the account is mysql_old_password:

  ```
  mysql> UPDATE mysql.user SET plugin = 'mysql_old_password'
  mysql> WHERE User = 'some_user' AND Host = 'some_host';
  mysql> FLUSH PRIVILEGES;
  mysql> SET PASSWORD FOR
      -> 'some_user'@'some_host' = OLD_PASSWORD('newpwd');
  ```

  Substitute the password you want to use for “*newpwd*” in the preceding example. MySQL cannot tell you what the original password was, so you'll need to pick a new one.
- Tell the server to use the older password hashing algorithm by default:

  1. Start mysqld with the old_passwords system variable set to 1.
  2. Assign an old-format password to each account that has had its password updated to the longer 4.1 format. You can identify these accounts with the following query:

     ```
     mysql> SELECT Host, User, Password FROM mysql.user
         -> WHERE LENGTH(Password) > 16;
     ```

     For each account record displayed by the query, use the Host and User values and assign a password using one of the methods described previously.

The Client does not support authentication protocol error also can occur if multiple versions of MySQL are installed but client programs are dynamically linked and link to an older library. Make sure that clients use the most recent library version with which they are compatible. The procedure to do this will depend on your system.

Note

The mysql extension does not support the authentication protocol in MySQL 4.1.1 and higher. This is true regardless of the PHP version being used. If you wish to use the mysql extension with MySQL 4.1 or newer, you may need to follow one of the options discussed above for configuring MySQL to work with old clients. The mysqli extension (stands for "MySQL, Improved"; added in PHP 5) is compatible with the improved password hashing employed in MySQL 4.1 and higher, and no special configuration of MySQL need be done to use this MySQL client library. For more information about the mysqli extension, see http://php.net/mysqli.

For additional background on password hashing and authentication, see Section 6.1.2.4, “Password Hashing in MySQL”.

C.5.2.5 Password Fails When Entered Interactively

MySQL client programs prompt for a password when invoked with a --password or -p option that has no following password value:

```
shell> mysql -u user_name -p
Enter password:
```

On some systems, you may find that your password works when specified in an option file or on the command line, but not when you enter it interactively at the `Enter password:` prompt. This occurs when the library provided by the system to read passwords limits password values to a small number of characters (typically eight). That is a problem with the system library, not with MySQL. To work around it, change your MySQL password to a value that is eight or fewer characters long, or put your password in an option file.

C.5.2.6 `Host 'host_name' is blocked`

If the following error occurs, it means that `mysqld` has received many connection requests from the given host that were interrupted in the middle:

```
Host 'host_name' is blocked because of many connection errors.
Unblock with 'mysqladmin flush-hosts'
```

The value of the `max_connect_errors` system variable determines how many successive interrupted connection requests are permitted. (See Section 5.1.4, “Server System Variables”.) After `max_connect_errors` failed requests without a successful connection, `mysqld` assumes that something is wrong (for example, that someone is trying to break in), and blocks the host from further connections until you issue a `FLUSH HOSTS` statement or execute a `mysqladmin flush-hosts` command.

By default, `mysqld` blocks a host after 100 connection errors. You can adjust the value by setting `max_connect_errors` at server startup:

```
shell> mysqld_safe --max_connect_errors=10000 &
```

The value can also be set at runtime:

```
mysql> SET GLOBAL max_connect_errors=10000;
```

If you get the `Host 'host_name' is blocked` error message for a given host, you should first verify that there is nothing wrong with TCP/IP connections from that host. If you are having network problems, it does you no good to increase the value of the `max_connect_errors` variable.

C.5.2.7 `Too many connections`

If you get a `Too many connections` error when you try to connect to the `mysqld` server, this means that all available connections are in use by other clients.

The number of connections permitted is controlled by the `max_connections` system variable. The default value is 151 to improve performance when MySQL is used with the Apache Web server. (Previously, the default was 100.) If you need to support more connections, you should set a larger value for this variable.

`mysqld` actually permits `max_connections+1` clients to connect. The extra connection is reserved for use by accounts that have the `SUPER` privilege. By granting the `SUPER` privilege to administrators and not to normal users (who should not need it), an administrator can connect to the server and use `SHOW PROCESSLIST` to diagnose problems even if the maximum number of unprivileged clients are connected. See Section 13.7.5.28, “`SHOW PROCESSLIST` Syntax”.

The maximum number of connections MySQL can support depends on the quality of the thread library on a given platform, the amount of RAM available, how much RAM is used for each connection, the workload from each connection, and the desired response time. Linux or Solaris should be able to support at 500 to

1000 simultaneous connections routinely and as many as 10,000 connections if you have many gigabytes of RAM available and the workload from each is low or the response time target undemanding. Windows is limited to (open tables × 2 + open connections) < 2048 due to the Posix compatibility layer used on that platform.

Increasing open-files-limit may be necessary. Also see Section 2.5, “Installing MySQL on Linux”, for how to raise the operating system limit on how many handles can be used by MySQL.

C.5.2.8 Out of memory

If you issue a query using the mysql client program and receive an error like the following one, it means that mysql does not have enough memory to store the entire query result:

```
mysql: Out of memory at line 42, 'malloc.c'
mysql: needed 8136 byte (8k), memory in use: 12481367 bytes (12189k)
ERROR 2008: MySQL client ran out of memory
```

To remedy the problem, first check whether your query is correct. Is it reasonable that it should return so many rows? If not, correct the query and try again. Otherwise, you can invoke mysql with the --quick option. This causes it to use the mysql_use_result() C API function to retrieve the result set, which places less of a load on the client (but more on the server).

C.5.2.9 MySQL server has gone away

This section also covers the related Lost connection to server during query error.

The most common reason for the MySQL server has gone away error is that the server timed out and closed the connection. In this case, you normally get one of the following error codes (which one you get is operating system-dependent).

Error Code	Description
CR_SERVER_GONE_ERROR	The client couldn't send a question to the server.
CR_SERVER_LOST	The client didn't get an error when writing to the server, but it didn't get a full answer (or any answer) to the question.

By default, the server closes the connection after eight hours if nothing has happened. You can change the time limit by setting the wait_timeout variable when you start mysqld. See Section 5.1.4, “Server System Variables”.

If you have a script, you just have to issue the query again for the client to do an automatic reconnection. This assumes that you have automatic reconnection in the client enabled (which is the default for the mysql command-line client).

Some other common reasons for the MySQL server has gone away error are:

- You (or the db administrator) has killed the running thread with a KILL statement or a mysqladmin kill command.
- You tried to run a query after closing the connection to the server. This indicates a logic error in the application that should be corrected.
- A client application running on a different host does not have the necessary privileges to connect to the MySQL server from that host.
- You got a timeout from the TCP/IP connection on the client side. This may happen if you have been using the commands: mysql_options(..., MYSQL_OPT_READ_TIMEOUT,...) or

mysql_options(..., MYSQL_OPT_WRITE_TIMEOUT,...). In this case increasing the timeout may help solve the problem.

- You have encountered a timeout on the server side and the automatic reconnection in the client is disabled (the reconnect flag in the MYSQL structure is equal to 0).
- You are using a Windows client and the server had dropped the connection (probably because wait_timeout expired) before the command was issued.

  The problem on Windows is that in some cases MySQL does not get an error from the OS when writing to the TCP/IP connection to the server, but instead gets the error when trying to read the answer from the connection.

  The solution to this is to either do a mysql_ping() on the connection if there has been a long time since the last query (this is what Connector/ODBC does) or set wait_timeout on the mysqld server so high that it in practice never times out.
- You can also get these errors if you send a query to the server that is incorrect or too large. If mysqld receives a packet that is too large or out of order, it assumes that something has gone wrong with the client and closes the connection. If you need big queries (for example, if you are working with big BLOB columns), you can increase the query limit by setting the server's max_allowed_packet variable, which has a default value of 1MB. You may also need to increase the maximum packet size on the client end. More information on setting the packet size is given in Section C.5.2.10, “Packet Too Large”.

  An INSERT or REPLACE statement that inserts a great many rows can also cause these sorts of errors. Either one of these statements sends a single request to the server irrespective of the number of rows to be inserted; thus, you can often avoid the error by reducing the number of rows sent per INSERT or REPLACE.
- You also get a lost connection if you are sending a packet 16MB or larger if your client is older than 4.0.8 and your server is 4.0.8 and above, or the other way around.
- It is also possible to see this error if host name lookups fail (for example, if the DNS server on which your server or network relies goes down). This is because MySQL is dependent on the host system for name resolution, but has no way of knowing whether it is working—from MySQL's point of view the problem is indistinguishable from any other network timeout.

  You may also see the MySQL server has gone away error if MySQL is started with the --skip-networking option.

  Another networking issue that can cause this error occurs if the MySQL port (default 3306) is blocked by your firewall, thus preventing any connections at all to the MySQL server.
- You can also encounter this error with applications that fork child processes, all of which try to use the same connection to the MySQL server. This can be avoided by using a separate connection for each child process.
- You have encountered a bug where the server died while executing the query.

You can check whether the MySQL server died and restarted by executing mysqladmin version and examining the server's uptime. If the client connection was broken because mysqld crashed and restarted, you should concentrate on finding the reason for the crash. Start by checking whether issuing the query again kills the server again. See Section C.5.4.2, “What to Do If MySQL Keeps Crashing”.

You can get more information about the lost connections by starting mysqld with the log_error_verbosity system variable set to 3. This logs some of the disconnection messages in the hostname.err file. See Section 5.2.2, “The Error Log”.

If you want to create a bug report regarding this problem, be sure that you include the following information:

- Indicate whether the MySQL server died. You can find information about this in the server error log. See Section C.5.4.2, "What to Do If MySQL Keeps Crashing".
- If a specific query kills mysqld and the tables involved were checked with CHECK TABLE before you ran the query, can you provide a reproducible test case? See Section 22.4, "Debugging and Porting MySQL".
- What is the value of the wait_timeout system variable in the MySQL server? (mysqladmin variables gives you the value of this variable.)
- Have you tried to run mysqld with the general query log enabled to determine whether the problem query appears in the log? (See Section 5.2.3, "The General Query Log".)

See also Section C.5.2.11, "Communication Errors and Aborted Connections", and Section 1.7, "How to Report Bugs or Problems".

C.5.2.10 Packet Too Large

A communication packet is a single SQL statement sent to the MySQL server, a single row that is sent to the client, or a binary log event sent from a master replication server to a slave.

The largest possible packet that can be transmitted to or from a MySQL 5.7 server or client is 1GB.

When a MySQL client or the mysqld server receives a packet bigger than max_allowed_packet bytes, it issues an ER_NET_PACKET_TOO_LARGE error and closes the connection. With some clients, you may also get a Lost connection to MySQL server during query error if the communication packet is too large.

Both the client and the server have their own max_allowed_packet variable, so if you want to handle big packets, you must increase this variable both in the client and in the server.

If you are using the mysql client program, its default max_allowed_packet variable is 16MB. To set a larger value, start mysql like this:

```
shell> mysql --max_allowed_packet=32M
```

That sets the packet size to 32MB.

The server's default max_allowed_packet value is 1MB. You can increase this if the server needs to handle big queries (for example, if you are working with big BLOB columns). For example, to set the variable to 16MB, start the server like this:

```
shell> mysqld --max_allowed_packet=16M
```

You can also use an option file to set max_allowed_packet. For example, to set the size for the server to 16MB, add the following lines in an option file:

```
[mysqld]
max_allowed_packet=16M
```

It is safe to increase the value of this variable because the extra memory is allocated only when needed. For example, mysqld allocates more memory only when you issue a long query or when mysqld must return a large result row. The small default value of the variable is a precaution to catch incorrect packets between the client and server and also to ensure that you do not run out of memory by using large packets accidentally.

You can also get strange problems with large packets if you are using large `BLOB` values but have not given `mysqld` access to enough memory to handle the query. If you suspect this is the case, try adding `ulimit -d 256000` to the beginning of the `mysqld_safe` script and restarting `mysqld`.

C.5.2.11 Communication Errors and Aborted Connections

If connection problems occur such as communication errors or aborted connections, use these sources of information to diagnose problems:

- The error log. See Section 5.2.2, “The Error Log”.
- The general query log. See Section 5.2.3, “The General Query Log”.
- The `Aborted_xxx` and `Connection_errors_xxx` status variables. See Section 5.1.6, “Server Status Variables”.
- The host cache, which is accessible using the `host_cache` Performance Schema table. See Section 8.11.5.2, “DNS Lookup Optimization and the Host Cache”, and Section 20.9.13.1, “The `host_cache` Table”.

If you start the server with the `log_error_verbosity` system variable set to 3, you might find messages like this in your error log:

```
2013-09-24T12:12:37.839018Z 854 [Note] Aborted connection 854 to db:
'users' user: 'josh'
```

If a client successfully connects but later disconnects improperly or is terminated, the server increments the `Aborted_clients` status variable, and logs an `Aborted connection` message to the error log. The cause can be any of the following:

- The client program did not call `mysql_close()` before exiting.
- The client had been sleeping more than `wait_timeout` or `interactive_timeout` seconds without issuing any requests to the server. See Section 5.1.4, “Server System Variables”.
- The client program ended abruptly in the middle of a data transfer.

If a client is unable even to connect, the server increments the `Aborted_connects` status variable. Unsuccessful connection attempts can occur for the following reasons:

- A client does not have privileges to connect to a database.
- A client uses an incorrect password.
- A connection packet does not contain the right information.
- It takes more than `connect_timeout` seconds to get a connect packet. See Section 5.1.4, “Server System Variables”.

If these kinds of things happen, it might indicate that someone is trying to break into your server! Messages for these types of problems are logged to the general query log if it is enabled.

Other reasons for problems with aborted clients or aborted connections:

- The `max_allowed_packet` variable value is too small or queries require more memory than you have allocated for `mysqld`. See Section C.5.2.10, “Packet Too Large”.

- Use of Ethernet protocol with Linux, both half and full duplex. Many Linux Ethernet drivers have this bug. You should test for this bug by transferring a huge file using FTP between the client and server machines. If a transfer goes in burst-pause-burst-pause mode, you are experiencing a Linux duplex syndrome. Switch the duplex mode for both your network card and hub/switch to either full duplex or to half duplex and test the results to determine the best setting.
- A problem with the thread library that causes interrupts on reads.
- Badly configured TCP/IP.
- Faulty Ethernets, hubs, switches, cables, and so forth. This can be diagnosed properly only by replacing hardware.

See also Section C.5.2.9, “`MySQL server has gone away`”.

C.5.2.12 `The table is full`

If a table-full error occurs, it may be that the disk is full or that the table has reached its maximum size. The effective maximum table size for MySQL databases is usually determined by operating system constraints on file sizes, not by MySQL internal limits. See Section E.10.3, “Limits on Table Size”.

C.5.2.13 `Can't create/write to file`

If you get an error of the following type for some queries, it means that MySQL cannot create a temporary file for the result set in the temporary directory:

```
Can't create/write to file '\\sqla3fe_0.ism'.
```

The preceding error is a typical message for Windows; the Unix message is similar.

One fix is to start `mysqld` with the `--tmpdir` option or to add the option to the `[mysqld]` section of your option file. For example, to specify a directory of `C:\temp`, use these lines:

```
[mysqld]
tmpdir=C:/temp
```

The `C:\temp` directory must exist and have sufficient space for the MySQL server to write to. See Section 4.2.3.3, “Using Option Files”.

Another cause of this error can be permissions issues. Make sure that the MySQL server can write to the `tmpdir` directory.

Check also the error code that you get with `perror`. One reason the server cannot write to a table is that the file system is full:

```
shell> perror 28
OS error code  28:  No space left on device
```

If you get an error of the following type during startup, it indicates that the file system or directory used for storing data files is write protected. Provided that the write error is to a test file, the error is not serious and can be safely ignored.

```
Can't create test file /usr/local/mysql/data/master.lower-test
```

C.5.2.14 `Commands out of sync`

If you get Commands out of sync; you can't run this command now in your client code, you are calling client functions in the wrong order.

This can happen, for example, if you are using mysql_use_result() and try to execute a new query before you have called mysql_free_result(). It can also happen if you try to execute two queries that return data without calling mysql_use_result() or mysql_store_result() in between.

C.5.2.15 Ignoring user

If you get the following error, it means that when mysqld was started or when it reloaded the grant tables, it found an account in the user table that had an invalid password.

Found wrong password for user '*some_user*'@'*some_host*'; ignoring user

As a result, the account is simply ignored by the permission system.

The following list indicates possible causes of and fixes for this problem:

- You may be running a new version of mysqld with an old user table. You can check this by executing mysqlshow mysql user to see whether the Password column is shorter than 16 characters. If so, you can correct this condition by running the scripts/add_long_password script.
- The account has an old password (eight characters long). Update the account in the user table to have a new password.
- You have specified a password in the user table without using the PASSWORD() function. Use mysql to update the account in the user table with a new password, making sure to use the PASSWORD() function:

```
mysql> UPDATE user SET Password=PASSWORD('newpwd')
    -> WHERE User='some_user' AND Host='some_host';
```

C.5.2.16 Table '*tbl_name*' doesn't exist

If you get either of the following errors, it usually means that no table exists in the default database with the given name:

```
Table 'tbl_name' doesn't exist
Can't find file: 'tbl_name' (errno: 2)
```

In some cases, it may be that the table does exist but that you are referring to it incorrectly:

- Because MySQL uses directories and files to store databases and tables, database and table names are case sensitive if they are located on a file system that has case-sensitive file names.
- Even for file systems that are not case sensitive, such as on Windows, all references to a given table within a query must use the same lettercase.

You can check which tables are in the default database with SHOW TABLES. See Section 13.7.5, “SHOW Syntax”.

C.5.2.17 Can't initialize character set

You might see an error like this if you have character set problems:

```
MySQL Connection Failed: Can't initialize character set charset_name
```

This error can have any of the following causes:

- The character set is a multi-byte character set and you have no support for the character set in the client. In this case, you need to recompile the client by running `CMake` with the `-DDEFAULT_CHARSET=charset_name` or `-DWITH_EXTRA_CHARSETS=charset_name` option. See Section 2.8.4, “MySQL Source-Configuration Options”.

  All standard MySQL binaries are compiled with `-DWITH_EXTRA_CHARSETS=complex`, which enables support for all multi-byte character sets. See Section 2.8.4, “MySQL Source-Configuration Options”.

- The character set is a simple character set that is not compiled into `mysqld`, and the character set definition files are not in the place where the client expects to find them.

  In this case, you need to use one of the following methods to solve the problem:

  - Recompile the client with support for the character set. See Section 2.8.4, “MySQL Source-Configuration Options”.
  - Specify to the client the directory where the character set definition files are located. For many clients, you can do this with the `--character-sets-dir` option.
  - Copy the character definition files to the path where the client expects them to be.

C.5.2.18 `'File' Not Found` and Similar Errors

If you get `ERROR '...' not found (errno: 23)`, `Can't open file: ... (errno: 24)`, or any other error with `errno 23` or `errno 24` from MySQL, it means that you haven't allocated enough file descriptors for the MySQL server. You can use the `perror` utility to get a description of what the error number means:

```
shell> perror 23
OS error code  23:  File table overflow
shell> perror 24
OS error code  24:  Too many open files
shell> perror 11
OS error code  11:  Resource temporarily unavailable
```

The problem here is that `mysqld` is trying to keep open too many files simultaneously. You can either tell `mysqld` not to open so many files at once or increase the number of file descriptors available to `mysqld`.

To tell `mysqld` to keep open fewer files at a time, you can make the table cache smaller by reducing the value of the `table_open_cache` system variable (the default value is 64). This may not entirely prevent running out of file descriptors because in some circumstances the server may attempt to extend the cache size temporarily, as described in Section 8.4.3.1, “How MySQL Opens and Closes Tables”. Reducing the value of `max_connections` also reduces the number of open files (the default value is 100).

To change the number of file descriptors available to `mysqld`, you can use the `--open-files-limit` option to `mysqld_safe` or set the `open_files_limit` system variable. See Section 5.1.4, “Server System Variables”. The easiest way to set these values is to add an option to your option file. See Section 4.2.3.3, “Using Option Files”. If you have an old version of `mysqld` that does not support setting the open files limit, you can edit the `mysqld_safe` script. There is a commented-out line `ulimit -n 256` in the script. You can remove the “`#`” character to uncomment this line, and change the number `256` to set the number of file descriptors to be made available to `mysqld`.

`--open-files-limit` and `ulimit` can increase the number of file descriptors, but only up to the limit imposed by the operating system. There is also a “hard” limit that can be overridden only if you start `mysqld_safe` or `mysqld` as `root` (just remember that you also need to start the server with the

--user option in this case so that it does not continue to run as root after it starts up). If you need to increase the operating system limit on the number of file descriptors available to each process, consult the documentation for your system.

Note

If you run the tcsh shell, ulimit does not work! tcsh also reports incorrect values when you ask for the current limits. In this case, you should start mysqld_safe using sh.

C.5.2.19 Table-Corruption Issues

If you have started mysqld with --myisam-recover-options, MySQL automatically checks and tries to repair MyISAM tables if they are marked as 'not closed properly' or 'crashed'. If this happens, MySQL writes an entry in the hostname.err file 'Warning: Checking table ...' which is followed by Warning: Repairing table if the table needs to be repaired. If you get a lot of these errors, without mysqld having died unexpectedly just before, then something is wrong and needs to be investigated further.

In MySQL 5.7, when the server detects MyISAM table corruption, it writes additional information to the error log, such as the name and line number of the source file, and the list of threads accessing the table. Example: Got an error from thread_id=1, mi_dynrec.c:368. This is useful information to include in bug reports.

See also Section 5.1.3, “Server Command Options”, and Section 22.4.1.7, “Making a Test Case If You Experience Table Corruption”.

C.5.3 Installation-Related Issues

C.5.3.1 Problems with File Permissions

If you have problems with file permissions, the UMASK environment variable might be set incorrectly when mysqld starts. For example, MySQL might issue the following error message when you create a table:

```
ERROR: Can't find file: 'path/with/filename.frm' (Errcode: 13)
```

The default UMASK value is 0660. You can change this behavior by starting mysqld_safe as follows:

```
shell> UMASK=384  # = 600 in octal
shell> export UMASK
shell> mysqld_safe &
```

By default, MySQL creates database directories with an access permission value of 0700. You can modify this behavior by setting the UMASK_DIR variable. If you set its value, new directories are created with the combined UMASK and UMASK_DIR values. For example, if you want to give group access to all new directories, you can do this:

```
shell> UMASK_DIR=504  # = 770 in octal
shell> export UMASK_DIR
shell> mysqld_safe &
```

MySQL assumes that the value for UMASK or UMASK_DIR is in octal if it starts with a zero.

See Section 2.11, “Environment Variables”.

C.5.4 Administration-Related Issues

C.5.4.1 How to Reset the Root Password

If you have never set a `root` password for MySQL, the server does not require a password at all for connecting as `root`. However, this is insecure. For instructions on assigning passwords, see Section 2.9.2, "Securing the Initial MySQL Accounts".

If you know the `root` password, but want to change it, see Section 13.7.1.7, "`SET PASSWORD` Syntax".

If you set a `root` password previously, but have forgotten it, you can set a new password. The following sections provide instructions for Windows and Unix systems, as well as generic instructions that apply to any system.

Resetting the Root Password: Windows Systems

On Windows, use the following procedure to reset the password for all MySQL `root` accounts:

1. Log on to your system as Administrator.

2. Stop the MySQL server if it is running. For a server that is running as a Windows service, go to the Services manager: From the Start menu, select Control Panel, then Administrative Tools, then Services. Find the MySQL service in the list and stop it.

   If your server is not running as a service, you may need to use the Task Manager to force it to stop.

3. Create a text file containing the following statements. Replace the password with the password that you want to use.

   ```
   UPDATE mysql.user SET Password=PASSWORD('MyNewPass') WHERE User='root';
   FLUSH PRIVILEGES;
   ```

   Write the `UPDATE` and `FLUSH` statements each on a single line. The `UPDATE` statement resets the password for all `root` accounts, and the `FLUSH` statement tells the server to reload the grant tables into memory so that it notices the password change.

4. Save the file. For this example, the file will be named `C:\mysql-init.txt`.

5. Open a console window to get to the command prompt: From the Start menu, select Run, then enter `cmd` as the command to be run.

6. Start the MySQL server with the special `--init-file` option (notice that the backslash in the option value is doubled):

   ```
   C:\> C:\mysql\bin\mysqld --init-file=C:\\mysql-init.txt
   ```

   If you installed MySQL to a location other than `C:\mysql`, adjust the command accordingly.

   The server executes the contents of the file named by the `--init-file` option at startup, changing each `root` account password.

   You can also add the `--console` option to the command if you want server output to appear in the console window rather than in a log file.

   If you installed MySQL using the MySQL Installation Wizard, you may need to specify a `--defaults-file` option:

```
C:\> "C:\Program Files\MySQL\MySQL Server 5.7\bin\mysqld.exe"
         --defaults-file="C:\\Program Files\\MySQL\\MySQL Server 5.7\\my.ini"
         --init-file=C:\\mysql-init.txt
```

The appropriate `--defaults-file` setting can be found using the Services Manager: From the Start menu, select Control Panel, then Administrative Tools, then Services. Find the MySQL service in the list, right-click it, and choose the `Properties` option. The `Path to executable` field contains the `--defaults-file` setting.

7. After the server has started successfully, delete `C:\mysql-init.txt`.

You should now be able to connect to the MySQL server as `root` using the new password. Stop the MySQL server, then restart it in normal mode again. If you run the server as a service, start it from the Windows Services window. If you start the server manually, use whatever command you normally use.

Resetting the Root Password: Unix Systems

On Unix, use the following procedure to reset the password for all MySQL `root` accounts. The instructions assume that you will start the server so that it runs using the Unix login account that you normally use for running the server. For example, if you run the server using the `mysql` login account, you should log in as `mysql` before using the instructions. Alternatively, you can log in as `root`, but in this case you *must* start `mysqld` with the `--user=mysql` option. If you start the server as `root` without using `--user=mysql`, the server may create `root`-owned files in the data directory, such as log files, and these may cause permission-related problems for future server startups. If that happens, you will need to either change the ownership of the files to `mysql` or remove them.

1. Log on to your system as the Unix user that the `mysqld` server runs as (for example, `mysql`).

2. Locate the `.pid` file that contains the server's process ID. The exact location and name of this file depend on your distribution, host name, and configuration. Common locations are `/var/lib/mysql/`, `/var/run/mysqld/`, and `/usr/local/mysql/data/`. Generally, the file name has an extension of `.pid` and begins with either `mysqld` or your system's host name.

   You can stop the MySQL server by sending a normal `kill` (not `kill -9`) to the `mysqld` process, using the path name of the `.pid` file in the following command:

   ```
   shell> kill `cat /mysql-data-directory/host_name.pid`
   ```

   Use backticks (not forward quotation marks) with the `cat` command. These cause the output of `cat` to be substituted into the `kill` command.

3. Create a text file containing the following statements. Replace the password with the password that you want to use.

   ```
   UPDATE mysql.user SET Password=PASSWORD('MyNewPass') WHERE User='root';
   FLUSH PRIVILEGES;
   ```

   Write the `UPDATE` and `FLUSH` statements each on a single line. The `UPDATE` statement resets the password for all `root` accounts, and the `FLUSH` statement tells the server to reload the grant tables into memory so that it notices the password change.

4. Save the file. For this example, the file will be named `/home/me/mysql-init`. The file contains the password, so it should not be saved where it can be read by other users. If you are not logged in as `mysql` (the user the server runs as), make sure that the file has permissions that permit `mysql` to read it.

5. Start the MySQL server with the special `--init-file` option:

   ```
   shell> mysqld_safe --init-file=/home/me/mysql-init &
   ```

   The server executes the contents of the file named by the `--init-file` option at startup, changing each `root` account password.

6. After the server has started successfully, delete `/home/me/mysql-init`.

You should now be able to connect to the MySQL server as `root` using the new password. Stop the server and restart it normally.

Resetting the Root Password: Generic Instructions

The preceding sections provide password-resetting instructions for Windows and Unix systems. Alternatively, on any platform, you can set the new password using the `mysql` client (but this approach is less secure):

1. Stop `mysqld` and restart it with the `--skip-grant-tables` option. This enables anyone to connect without a password and with all privileges. Because this is insecure, you might want to use `--skip-grant-tables` in conjunction with `--skip-networking` to prevent remote clients from connecting.

2. Connect to the `mysqld` server with this command:

   ```
   shell> mysql
   ```

3. Issue the following statements in the `mysql` client. Replace the password with the password that you want to use.

   ```
   mysql> UPDATE mysql.user SET Password=PASSWORD('MyNewPass')
       ->                   WHERE User='root';
   mysql> FLUSH PRIVILEGES;
   ```

   The `FLUSH` statement tells the server to reload the grant tables into memory so that it notices the password change.

You should now be able to connect to the MySQL server as `root` using the new password. Stop the server, then restart it normally (without the `--skip-grant-tables` and `--skip-networking` options).

C.5.4.2 What to Do If MySQL Keeps Crashing

Each MySQL version is tested on many platforms before it is released. This does not mean that there are no bugs in MySQL, but if there are bugs, they should be very few and can be hard to find. If you have a problem, it always helps if you try to find out exactly what crashes your system, because you have a much better chance of getting the problem fixed quickly.

First, you should try to find out whether the problem is that the `mysqld` server dies or whether your problem has to do with your client. You can check how long your `mysqld` server has been up by executing `mysqladmin version`. If `mysqld` has died and restarted, you may find the reason by looking in the server's error log. See Section 5.2.2, “The Error Log”.

On some systems, you can find in the error log a stack trace of where `mysqld` died that you can resolve with the `resolve_stack_dump` program. See Section 22.4, “Debugging and Porting MySQL”. Note that the variable values written in the error log may not always be 100% correct.

Many server crashes are caused by corrupted data files or index files. MySQL updates the files on disk with the `write()` system call after every SQL statement and before the client is notified about the result.

(This is not true if you are running with --delay-key-write, in which case data files are written but not index files.) This means that data file contents are safe even if mysqld crashes, because the operating system ensures that the unflushed data is written to disk. You can force MySQL to flush everything to disk after every SQL statement by starting mysqld with the --flush option.

The preceding means that normally you should not get corrupted tables unless one of the following happens:

- The MySQL server or the server host was killed in the middle of an update.
- You have found a bug in mysqld that caused it to die in the middle of an update.
- Some external program is manipulating data files or index files at the same time as mysqld without locking the table properly.
- You are running many mysqld servers using the same data directory on a system that does not support good file system locks (normally handled by the lockd lock manager), or you are running multiple servers with external locking disabled.
- You have a crashed data file or index file that contains very corrupt data that confused mysqld.
- You have found a bug in the data storage code. This isn't likely, but it is at least possible. In this case, you can try to change the storage engine to another engine by using ALTER TABLE on a repaired copy of the table.

Because it is very difficult to know why something is crashing, first try to check whether things that work for others crash for you. Try the following things:

- Stop the mysqld server with mysqladmin shutdown, run myisamchk --silent --force */*.MYI from the data directory to check all MyISAM tables, and restart mysqld. This ensures that you are running from a clean state. See Chapter 5, *MySQL Server Administration*.
- Start mysqld with the general query log enabled (see Section 5.2.3, “The General Query Log”). Then try to determine from the information written to the log whether some specific query kills the server. About 95% of all bugs are related to a particular query. Normally, this is one of the last queries in the log file just before the server restarts. See Section 5.2.3, “The General Query Log”. If you can repeatedly kill MySQL with a specific query, even when you have checked all tables just before issuing it, then you have isolated the bug and should submit a bug report for it. See Section 1.7, “How to Report Bugs or Problems”.
- Try to make a test case that we can use to repeat the problem. See Section 22.4, “Debugging and Porting MySQL”.
- Try running the tests in the mysql-test directory and the MySQL benchmarks. See Section 22.1.2, “The MySQL Test Suite”. They test MySQL rather well. You can also add code to the benchmarks that simulates your application. The benchmarks are in the sql-bench directory in a source distribution or, for a binary distribution, in the sql-bench directory under your MySQL installation directory.
- Try the fork_big.pl script. (It is located in the tests directory of source distributions.)
- Configuring MySQL for debugging makes it much easier to gather information about possible errors if something goes wrong. Reconfigure MySQL with the -DWITH_DEBUG=1 option to CMake and then recompile. See Section 22.4, “Debugging and Porting MySQL”.
- Make sure that you have applied the latest patches for your operating system.
- Use the --skip-external-locking option to mysqld. On some systems, the lockd lock manager does not work properly; the --skip-external-locking option tells mysqld not to use external

locking. (This means that you cannot run two mysqld servers on the same data directory and that you must be careful if you use myisamchk. Nevertheless, it may be instructive to try the option as a test.)

- If mysqld appears to be running but not responding, try mysqladmin -u root processlist. Sometimes mysqld is not hung even though it seems unresponsive. The problem may be that all connections are in use, or there may be some internal lock problem. mysqladmin -u root processlist usually is able to make a connection even in these cases, and can provide useful information about the current number of connections and their status.

- Run the command mysqladmin -i 5 status or mysqladmin -i 5 -r status in a separate window to produce statistics while running other queries.

- Try the following:

  1. Start mysqld from gdb (or another debugger). See Section 22.4, “Debugging and Porting MySQL”.

  2. Run your test scripts.

  3. Print the backtrace and the local variables at the three lowest levels. In gdb, you can do this with the following commands when mysqld has crashed inside gdb:

     ```
     backtrace
     info local
     up
     info local
     up
     info local
     ```

     With gdb, you can also examine which threads exist with info threads and switch to a specific thread with thread *N*, where *N* is the thread ID.

- Try to simulate your application with a Perl script to force MySQL to crash or misbehave.

- Send a normal bug report. See Section 1.7, “How to Report Bugs or Problems”. Be even more detailed than usual. Because MySQL works for many people, the crash might result from something that exists only on your computer (for example, an error that is related to your particular system libraries).

- If you have a problem with tables containing dynamic-length rows and you are using only VARCHAR columns (not BLOB or TEXT columns), you can try to change all VARCHAR to CHAR with ALTER TABLE. This forces MySQL to use fixed-size rows. Fixed-size rows take a little extra space, but are much more tolerant to corruption.

  The current dynamic row code has been in use for several years with very few problems, but dynamic-length rows are by nature more prone to errors, so it may be a good idea to try this strategy to see whether it helps.

- Consider the possibility of hardware faults when diagnosing problems. Defective hardware can be the cause of data corruption. Pay particular attention to your memory and disk subsystems when troubleshooting hardware.

C.5.4.3 How MySQL Handles a Full Disk

This section describes how MySQL responds to disk-full errors (such as “no space left on device”), and to quota-exceeded errors (such as “write failed” or “user block limit reached”).

This section is relevant for writes to MyISAM tables. It also applies for writes to binary log files and binary log index file, except that references to “row” and “record” should be understood to mean “event.”

When a disk-full condition occurs, MySQL does the following:

- It checks once every minute to see whether there is enough space to write the current row. If there is enough space, it continues as if nothing had happened.
- Every 10 minutes it writes an entry to the log file, warning about the disk-full condition.

To alleviate the problem, take the following actions:

- To continue, you only have to free enough disk space to insert all records.
- Alternatively, to abort the thread, use `mysqladmin kill`. The thread is aborted the next time it checks the disk (in one minute).
- Other threads might be waiting for the table that caused the disk-full condition. If you have several “locked” threads, killing the one thread that is waiting on the disk-full condition enables the other threads to continue.

Exceptions to the preceding behavior are when you use `REPAIR TABLE` or `OPTIMIZE TABLE` or when the indexes are created in a batch after `LOAD DATA INFILE` or after an `ALTER TABLE` statement. All of these statements may create large temporary files that, if left to themselves, would cause big problems for the rest of the system. If the disk becomes full while MySQL is doing any of these operations, it removes the big temporary files and mark the table as crashed. The exception is that for `ALTER TABLE`, the old table is left unchanged.

C.5.4.4 Where MySQL Stores Temporary Files

As of MySQL 5.7.1, non-compressed `InnoDB` temporary tables are, by default, stored in a temporary tablespace named `ibtmp1` that is located in the MySQL `data` directory (`datadir`). The `innodb_temp_data_file_path` option can be used to specify a different file name and location. Compressed `InnoDB` temporary tables are stored in their own independent tablespace files (`.ibd` files) in the path specified by t he `TMPDIR` environment variable.

On Unix, MySQL uses the value of the `TMPDIR` environment variable as the path name of the directory in which to store temporary files (with the exception of non-compressed `InnoDB` temporary tables, as described above). If `TMPDIR` is not set, MySQL uses the system default, which is usually `/tmp`, `/var/tmp`, or `/usr/tmp`.

On Windows, MySQL checks in order the values of the `TMPDIR`, `TEMP`, and `TMP` environment variables. For the first one found to be set, MySQL uses it and does not check those remaining. If none of `TMPDIR`, `TEMP`, or `TMP` are set, MySQL uses the Windows system default, which is usually `C:\windows\temp\`.

If the file system containing your temporary file directory is too small, you can use the `--tmpdir` option to `mysqld` to specify a directory in a file system where you have enough space. On replication slaves, you can use `--slave-load-tmpdir` to specify a separate directory for holding temporary files when replicating `LOAD DATA INFILE` statements.

The `--tmpdir` option can be set to a list of several paths that are used in round-robin fashion. Paths should be separated by colon characters (“`:`”) on Unix and semicolon characters (“`;`”) on Windows.

Note

To spread the load effectively, these paths should be located on different *physical* disks, not different partitions of the same disk.

If the MySQL server is acting as a replication slave, you should be sure to set `--slave-load-tmpdir` not to point to a directory that is on a memory-based file system or to a directory that is cleared when the server host restarts. A replication slave needs some of its temporary files to survive a machine restart so that it can replicate temporary tables or LOAD DATA INFILE operations. If files in the slave temporary file directory are lost when the server restarts, replication fails.

MySQL arranges that temporary files are removed if mysqld is terminated. On platforms that support it (such as Unix), this is done by unlinking the file after opening it. The disadvantage of this is that the name does not appear in directory listings and you do not see a big temporary file that fills up the file system in which the temporary file directory is located. (In such cases, lsof +L1 may be helpful in identifying large files associated with mysqld.)

When sorting (ORDER BY or GROUP BY), MySQL normally uses one or two temporary files. The maximum disk space required is determined by the following expression:

```
(length of what is sorted + sizeof(row pointer))
* number of matched rows
* 2
```

The row pointer size is usually four bytes, but may grow in the future for really big tables.

For some SELECT queries, MySQL also creates temporary SQL tables. These are not hidden and have names of the form SQL_*.

In most cases, ALTER TABLE creates a temporary copy of the original table in the same directory as the original table. However, if ALTER TABLE uses the in-place technique (online DDL), InnoDB creates temporary files in the temporary file directory. If this directory is not large enough to hold such files, you may need to set the tmpdir system variable to a different directory. For more information about online DDL, Section 14.2.11, “InnoDB and Online DDL”.

C.5.4.5 How to Protect or Change the MySQL Unix Socket File

The default location for the Unix socket file that the server uses for communication with local clients is /tmp/mysql.sock. (For some distribution formats, the directory might be different, such as /var/lib/mysql for RPMs.)

On some versions of Unix, anyone can delete files in the /tmp directory or other similar directories used for temporary files. If the socket file is located in such a directory on your system, this might cause problems.

On most versions of Unix, you can protect your /tmp directory so that files can be deleted only by their owners or the superuser (root). To do this, set the sticky bit on the /tmp directory by logging in as root and using the following command:

```
shell> chmod +t /tmp
```

You can check whether the sticky bit is set by executing ls -ld /tmp. If the last permission character is t, the bit is set.

Another approach is to change the place where the server creates the Unix socket file. If you do this, you should also let client programs know the new location of the file. You can specify the file location in several ways:

- Specify the path in a global or local option file. For example, put the following lines in /etc/my.cnf:

```
[mysqld]
socket=/path/to/socket
```

```
[client]
socket=/path/to/socket
```

See Section 4.2.3.3, “Using Option Files”.

- Specify a `--socket` option on the command line to `mysqld_safe` and when you run client programs.
- Set the `MYSQL_UNIX_PORT` environment variable to the path of the Unix socket file.
- Recompile MySQL from source to use a different default Unix socket file location. Define the path to the file with the `MYSQL_UNIX_ADDR` option when you run `CMake`. See Section 2.8.4, “MySQL Source-Configuration Options”.

You can test whether the new socket location works by attempting to connect to the server with this command:

```
shell> mysqladmin --socket=/path/to/socket version
```

C.5.4.6 Time Zone Problems

If you have a problem with `SELECT NOW()` returning values in UTC and not your local time, you have to tell the server your current time zone. The same applies if `UNIX_TIMESTAMP()` returns the wrong value. This should be done for the environment in which the server runs; for example, in `mysqld_safe` or `mysql.server`. See Section 2.11, “Environment Variables”.

You can set the time zone for the server with the `--timezone=`*`timezone_name`* option to `mysqld_safe`. You can also set it by setting the `TZ` environment variable before you start `mysqld`.

The permissible values for `--timezone` or `TZ` are system dependent. Consult your operating system documentation to see what values are acceptable.

C.5.5 Query-Related Issues

C.5.5.1 Case Sensitivity in String Searches

For nonbinary strings (`CHAR`, `VARCHAR`, `TEXT`), string searches use the collation of the comparison operands. For binary strings (`BINARY`, `VARBINARY`, `BLOB`), comparisons use the numeric values of the bytes in the operands; this means that for alphabetic characters, comparisons will be case sensitive.

A comparison between a nonbinary string and binary string is treated as a comparison of binary strings.

Simple comparison operations (`>=, >, =, <, <=`, sorting, and grouping) are based on each character's “sort value.” Characters with the same sort value are treated as the same character. For example, if “`e`” and “`é`” have the same sort value in a given collation, they compare as equal.

The default character set and collation are `latin1` and `latin1_swedish_ci`, so nonbinary string comparisons are case insensitive by default. This means that if you search with *`col_name`* `LIKE 'a%'`, you get all column values that start with `A` or `a`. To make this search case sensitive, make sure that one of the operands has a case sensitive or binary collation. For example, if you are comparing a column and a string that both have the `latin1` character set, you can use the `COLLATE` operator to cause either operand to have the `latin1_general_cs` or `latin1_bin` collation:

```
col_name COLLATE latin1_general_cs LIKE 'a%'
col_name LIKE 'a%' COLLATE latin1_general_cs
col_name COLLATE latin1_bin LIKE 'a%'
col_name LIKE 'a%' COLLATE latin1_bin
```

If you want a column always to be treated in case-sensitive fashion, declare it with a case sensitive or binary collation. See Section 13.1.14, “CREATE TABLE Syntax”.

To cause a case-sensitive comparison of nonbinary strings to be case insensitive, use COLLATE to name a case-insensitive collation. The strings in the following example normally are case sensitive, but COLLATE changes the comparison to be case insensitive:

```
mysql> SET @s1 = 'MySQL' COLLATE latin1_bin,
    ->     @s2 = 'mysql' COLLATE latin1_bin;
mysql> SELECT @s1 = @s2;
+-----------+
| @s1 = @s2 |
+-----------+
|         0 |
+-----------+
mysql> SELECT @s1 COLLATE latin1_swedish_ci = @s2;
+-------------------------------------+
| @s1 COLLATE latin1_swedish_ci = @s2 |
+-------------------------------------+
|                                   1 |
+-------------------------------------+
```

A binary string is case sensitive in comparisons. To compare the string as case insensitive, convert it to a nonbinary string and use COLLATE to name a case-insensitive collation:

```
mysql> SET @s = BINARY 'MySQL';
mysql> SELECT @s = 'mysql';
+--------------+
| @s = 'mysql' |
+--------------+
|            0 |
+--------------+
mysql> SELECT CONVERT(@s USING latin1) COLLATE latin1_swedish_ci = 'mysql';
+--------------------------------------------------------------+
| CONVERT(@s USING latin1) COLLATE latin1_swedish_ci = 'mysql' |
+--------------------------------------------------------------+
|                                                            1 |
+--------------------------------------------------------------+
```

To determine whether a value will compare as a nonbinary or binary string, use the COLLATION() function. This example shows that VERSION() returns a string that has a case-insensitive collation, so comparisons are case insensitive:

```
mysql> SELECT COLLATION(VERSION());
+----------------------+
| COLLATION(VERSION()) |
+----------------------+
| utf8_general_ci      |
+----------------------+
```

For binary strings, the collation value is binary, so comparisons will be case sensitive. One context in which you will see binary is for compression and encryption functions, which return binary strings as a general rule: string:

```
mysql> SELECT COLLATION(ENCRYPT('x')), COLLATION(SHA1('x'));
+-------------------------+----------------------+
| COLLATION(ENCRYPT('x')) | COLLATION(SHA1('x')) |
+--------------------------+-----------------------+
| binary                   | binary                |
+--------------------------+-----------------------+
```

To check the sort value of a string, the WEIGHT_STRING() may be helpful. See Section 12.5, “String Functions”.

C.5.5.2 Problems Using DATE Columns

The format of a DATE value is 'YYYY-MM-DD'. According to standard SQL, no other format is permitted. You should use this format in UPDATE expressions and in the WHERE clause of SELECT statements. For example:

```
SELECT * FROM t1 WHERE date >= '2003-05-05';
```

As a convenience, MySQL automatically converts a date to a number if the date is used in a numeric context and vice versa. MySQL also permits a “relaxed” string format when updating and in a WHERE clause that compares a date to a DATE, DATETIME, or TIMESTAMP column. “Relaxed” format means that any punctuation character may be used as the separator between parts. For example, '2004-08-15' and '2004#08#15' are equivalent. MySQL can also convert a string containing no separators (such as '20040815'), provided it makes sense as a date.

When you compare a DATE, TIME, DATETIME, or TIMESTAMP to a constant string with the <, <=, =, >=, >, or BETWEEN operators, MySQL normally converts the string to an internal long integer for faster comparison (and also for a bit more “relaxed” string checking). However, this conversion is subject to the following exceptions:

- When you compare two columns
- When you compare a DATE, TIME, DATETIME, or TIMESTAMP column to an expression
- When you use any comparison method other than those just listed, such as IN or STRCMP().

For those exceptions, the comparison is done by converting the objects to strings and performing a string comparison.

To be on the safe side, assume that strings are compared as strings and use the appropriate string functions if you want to compare a temporal value to a string.

The special “zero” date '0000-00-00' can be stored and retrieved as '0000-00-00'. When a '0000-00-00' date is used through Connector/ODBC, it is automatically converted to NULL because ODBC cannot handle that kind of date.

Because MySQL performs the conversions just described, the following statements work (assume that idate is a DATE column):

```
INSERT INTO t1 (idate) VALUES (19970505);
INSERT INTO t1 (idate) VALUES ('19970505');
INSERT INTO t1 (idate) VALUES ('97-05-05');
INSERT INTO t1 (idate) VALUES ('1997.05.05');
INSERT INTO t1 (idate) VALUES ('1997 05 05');
INSERT INTO t1 (idate) VALUES ('0000-00-00');

SELECT idate FROM t1 WHERE idate >= '1997-05-05';
SELECT idate FROM t1 WHERE idate >= 19970505;
SELECT MOD(idate,100) FROM t1 WHERE idate >= 19970505;
SELECT idate FROM t1 WHERE idate >= '19970505';
```

However, the following statement does not work:

```
SELECT idate FROM t1 WHERE STRCMP(idate,'20030505')=0;
```

`STRCMP()` is a string function, so it converts `idate` to a string in `'YYYY-MM-DD'` format and performs a string comparison. It does not convert `'20030505'` to the date `'2003-05-05'` and perform a date comparison.

If you enable the `ALLOW_INVALID_DATES` SQL mode, MySQL permits you to store dates that are given only limited checking: MySQL requires only that the day is in the range from 1 to 31 and the month is in the range from 1 to 12. This makes MySQL very convenient for Web applications where you obtain year, month, and day in three different fields and you want to store exactly what the user inserted (without date validation).

MySQL permits you to store dates where the day or month and day are zero. This is convenient if you want to store a birthdate in a `DATE` column and you know only part of the date. To disallow zero month or day parts in dates, enable strict SQL mode (as of MySQL 5.7.4) or the `NO_ZERO_IN_DATE` mode (before MySQL 5.7.4).

MySQL permits you to store a “zero” value of `'0000-00-00'` as a “dummy date.” This is in some cases more convenient than using `NULL` values. If a date to be stored in a `DATE` column cannot be converted to any reasonable value, MySQL stores `'0000-00-00'`. To disallow `'0000-00-00'`, enable strict SQL mode (as of MySQL 5.7.4) or the `NO_ZERO_DATE` mode (before MySQL 5.7.4).

To have MySQL check all dates and accept only legal dates (unless overridden by `IGNORE`), enable strict SQL mode (as of MySQL 5.7.4) or the `NO_ZERO_IN_DATE` and `NO_ZERO_DATE` modes (before MySQL 5.7.4).

C.5.5.3 Problems with `NULL` Values

The concept of the `NULL` value is a common source of confusion for newcomers to SQL, who often think that `NULL` is the same thing as an empty string `''`. This is not the case. For example, the following statements are completely different:

```
mysql> INSERT INTO my_table (phone) VALUES (NULL);
mysql> INSERT INTO my_table (phone) VALUES ('');
```

Both statements insert a value into the `phone` column, but the first inserts a `NULL` value and the second inserts an empty string. The meaning of the first can be regarded as “phone number is not known” and the meaning of the second can be regarded as “the person is known to have no phone, and thus no phone number.”

To help with `NULL` handling, you can use the `IS NULL` and `IS NOT NULL` operators and the `IFNULL()` function.

In SQL, the `NULL` value is never true in comparison to any other value, even `NULL`. An expression that contains `NULL` always produces a `NULL` value unless otherwise indicated in the documentation for the operators and functions involved in the expression. All columns in the following example return `NULL`:

```
mysql> SELECT NULL, 1+NULL, CONCAT('Invisible',NULL);
```

To search for column values that are `NULL`, you cannot use an `expr = NULL` test. The following statement returns no rows, because `expr = NULL` is never true for any expression:

```
mysql> SELECT * FROM my_table WHERE phone = NULL;
```

To look for NULL values, you must use the IS NULL test. The following statements show how to find the NULL phone number and the empty phone number:

```
mysql> SELECT * FROM my_table WHERE phone IS NULL;
mysql> SELECT * FROM my_table WHERE phone = '';
```

See Section 3.3.4.6, "Working with NULL Values", for additional information and examples.

You can add an index on a column that can have NULL values if you are using the MyISAM, InnoDB, or MEMORY storage engine. Otherwise, you must declare an indexed column NOT NULL, and you cannot insert NULL into the column.

When reading data with LOAD DATA INFILE, empty or missing columns are updated with ''. To load a NULL value into a column, use \N in the data file. The literal word "NULL" may also be used under some circumstances. See Section 13.2.6, "LOAD DATA INFILE Syntax".

When using DISTINCT, GROUP BY, or ORDER BY, all NULL values are regarded as equal.

When using ORDER BY, NULL values are presented first, or last if you specify DESC to sort in descending order.

Aggregate (summary) functions such as COUNT(), MIN(), and SUM() ignore NULL values. The exception to this is COUNT(*), which counts rows and not individual column values. For example, the following statement produces two counts. The first is a count of the number of rows in the table, and the second is a count of the number of non-NULL values in the age column:

```
mysql> SELECT COUNT(*), COUNT(age) FROM person;
```

For some data types, MySQL handles NULL values specially. If you insert NULL into a TIMESTAMP column, the current date and time is inserted. If you insert NULL into an integer or floating-point column that has the AUTO_INCREMENT attribute, the next number in the sequence is inserted.

C.5.5.4 Problems with Column Aliases

An alias can be used in a query select list to give a column a different name. You can use the alias in GROUP BY, ORDER BY, or HAVING clauses to refer to the column:

```
SELECT SQRT(a*b) AS root FROM tbl_name
  GROUP BY root HAVING root > 0;
SELECT id, COUNT(*) AS cnt FROM tbl_name
  GROUP BY id HAVING cnt > 0;
SELECT id AS 'Customer identity' FROM tbl_name;
```

Standard SQL disallows references to column aliases in a WHERE clause. This restriction is imposed because when the WHERE clause is evaluated, the column value may not yet have been determined. For example, the following query is illegal:

```
SELECT id, COUNT(*) AS cnt FROM tbl_name
  WHERE cnt > 0 GROUP BY id;
```

The WHERE clause determines which rows should be included in the GROUP BY clause, but it refers to the alias of a column value that is not known until after the rows have been selected, and grouped by the GROUP BY.

In the select list of a query, a quoted column alias can be specified using identifier or string quoting characters:

```
SELECT 1 AS `one`, 2 AS 'two';
```

Elsewhere in the statement, quoted references to the alias must use identifier quoting or the reference is treated as a string literal. For example, this statement groups by the values in column `id`, referenced using the alias `` `a` ``:

```
SELECT id AS 'a', COUNT(*) AS cnt FROM tbl_name
  GROUP BY `a`;
```

But this statement groups by the literal string `'a'` and will not work as expected:

```
SELECT id AS 'a', COUNT(*) AS cnt FROM tbl_name
  GROUP BY 'a';
```

C.5.5.5 Rollback Failure for Nontransactional Tables

If you receive the following message when trying to perform a `ROLLBACK`, it means that one or more of the tables you used in the transaction do not support transactions:

```
Warning: Some non-transactional changed tables couldn't be rolled back
```

These nontransactional tables are not affected by the `ROLLBACK` statement.

If you were not deliberately mixing transactional and nontransactional tables within the transaction, the most likely cause for this message is that a table you thought was transactional actually is not. This can happen if you try to create a table using a transactional storage engine that is not supported by your `mysqld` server (or that was disabled with a startup option). If `mysqld` does not support a storage engine, it instead creates the table as a `MyISAM` table, which is nontransactional.

You can check the storage engine for a table by using either of these statements:

```
SHOW TABLE STATUS LIKE 'tbl_name';
SHOW CREATE TABLE tbl_name;
```

See Section 13.7.5.35, “`SHOW TABLE STATUS` Syntax”, and Section 13.7.5.10, “`SHOW CREATE TABLE` Syntax”.

To check which storage engines your `mysqld` server supports, use this statement:

```
SHOW ENGINES;
```

See Section 13.7.5.15, “`SHOW ENGINES` Syntax” for full details.

C.5.5.6 Deleting Rows from Related Tables

If the total length of the `DELETE` statement for `related_table` is more than 1MB (the default value of the `max_allowed_packet` system variable), you should split it into smaller parts and execute multiple `DELETE` statements. You probably get the fastest `DELETE` by specifying only 100 to 1,000 `related_column` values per statement if the `related_column` is indexed. If the `related_column` isn't indexed, the speed is independent of the number of arguments in the `IN` clause.

C.5.5.7 Solving Problems with No Matching Rows

If you have a complicated query that uses many tables but that returns no rows, you should use the following procedure to find out what is wrong:

1. Test the query with `EXPLAIN` to check whether you can find something that is obviously wrong. See Section 13.8.2, "`EXPLAIN` Syntax".

2. Select only those columns that are used in the `WHERE` clause.

3. Remove one table at a time from the query until it returns some rows. If the tables are large, it is a good idea to use `LIMIT 10` with the query.

4. Issue a `SELECT` for the column that should have matched a row against the table that was last removed from the query.

5. If you are comparing `FLOAT` or `DOUBLE` columns with numbers that have decimals, you cannot use equality (`=`) comparisons. This problem is common in most computer languages because not all floating-point values can be stored with exact precision. In some cases, changing the `FLOAT` to a `DOUBLE` fixes this. See Section C.5.5.8, "Problems with Floating-Point Values".

6. If you still cannot figure out what is wrong, create a minimal test that can be run with `mysql test < query.sql` that shows your problems. You can create a test file by dumping the tables with `mysqldump --quick db_name *tbl_name_1* ... *tbl_name_n* > query.sql`. Open the file in an editor, remove some insert lines (if there are more than needed to demonstrate the problem), and add your `SELECT` statement at the end of the file.

   Verify that the test file demonstrates the problem by executing these commands:

   ```
   shell> mysqladmin create test2
   shell> mysql test2 < query.sql
   ```

   Attach the test file to a bug report, which you can file using the instructions in Section 1.7, "How to Report Bugs or Problems".

C.5.5.8 Problems with Floating-Point Values

Floating-point numbers sometimes cause confusion because they are approximate and not stored as exact values. A floating-point value as written in an SQL statement may not be the same as the value represented internally. Attempts to treat floating-point values as exact in comparisons may lead to problems. They are also subject to platform or implementation dependencies. The `FLOAT` and `DOUBLE` data types are subject to these issues. For `DECIMAL` columns, MySQL performs operations with a precision of 65 decimal digits, which should solve most common inaccuracy problems.

The following example uses `DOUBLE` to demonstrate how calculations that are done using floating-point operations are subject to floating-point error.

```
mysql> CREATE TABLE t1 (i INT, d1 DOUBLE, d2 DOUBLE);
mysql> INSERT INTO t1 VALUES (1, 101.40, 21.40), (1, -80.00, 0.00),
    -> (2, 0.00, 0.00), (2, -13.20, 0.00), (2, 59.60, 46.40),
    -> (2, 30.40, 30.40), (3, 37.00, 7.40), (3, -29.60, 0.00),
    -> (4, 60.00, 15.40), (4, -10.60, 0.00), (4, -34.00, 0.00),
    -> (5, 33.00, 0.00), (5, -25.80, 0.00), (5, 0.00, 7.20),
    -> (6, 0.00, 0.00), (6, -51.40, 0.00);

mysql> SELECT i, SUM(d1) AS a, SUM(d2) AS b
    -> FROM t1 GROUP BY i HAVING a <> b;
```

```
+------+-------+------+
| i    | a     | b    |
+------+-------+------+
|    1 |  21.4 | 21.4 |
|    2 |  76.8 | 76.8 |
|    3 |   7.4 |  7.4 |
|    4 |  15.4 | 15.4 |
|    5 |   7.2 |  7.2 |
|    6 | -51.4 |    0 |
+------+-------+------+
```

The result is correct. Although the first five records look like they should not satisfy the comparison (the values of a and b do not appear to be different), they may do so because the difference between the numbers shows up around the tenth decimal or so, depending on factors such as computer architecture or the compiler version or optimization level. For example, different CPUs may evaluate floating-point numbers differently.

If columns d1 and d2 had been defined as DECIMAL rather than DOUBLE, the result of the SELECT query would have contained only one row—the last one shown above.

The correct way to do floating-point number comparison is to first decide on an acceptable tolerance for differences between the numbers and then do the comparison against the tolerance value. For example, if we agree that floating-point numbers should be regarded the same if they are same within a precision of one in ten thousand (0.0001), the comparison should be written to find differences larger than the tolerance value:

```
mysql> SELECT i, SUM(d1) AS a, SUM(d2) AS b FROM t1
    -> GROUP BY i HAVING ABS(a - b) > 0.0001;
+------+-------+------+
| i    | a     | b    |
+------+-------+------+
|    6 | -51.4 |    0 |
+------+-------+------+
1 row in set (0.00 sec)
```

Conversely, to get rows where the numbers are the same, the test should find differences within the tolerance value:

```
mysql> SELECT i, SUM(d1) AS a, SUM(d2) AS b FROM t1
    -> GROUP BY i HAVING ABS(a - b) <= 0.0001;
+------+------+------+
| i    | a    | b    |
+------+------+------+
|    1 | 21.4 | 21.4 |
|    2 | 76.8 | 76.8 |
|    3 |  7.4 |  7.4 |
|    4 | 15.4 | 15.4 |
|    5 |  7.2 |  7.2 |
+------+------+------+
5 rows in set (0.03 sec)
```

Floating-point values are subject to platform or implementation dependencies. Suppose that you execute the following statements:

```
CREATE TABLE t1(c1 FLOAT(53,0), c2 FLOAT(53,0));
INSERT INTO t1 VALUES('1e+52','-1e+52');
SELECT * FROM t1;
```

On some platforms, the SELECT statement returns inf and -inf. On others, it returns 0 and -0.

An implication of the preceding issues is that if you attempt to create a replication slave by dumping table contents with `mysqldump` on the master and reloading the dump file into the slave, tables containing floating-point columns might differ between the two hosts.

C.5.6 Optimizer-Related Issues

MySQL uses a cost-based optimizer to determine the best way to resolve a query. In many cases, MySQL can calculate the best possible query plan, but sometimes MySQL does not have enough information about the data at hand and has to make “educated” guesses about the data.

For the cases when MySQL does not do the "right" thing, tools that you have available to help MySQL are:

- Use the `EXPLAIN` statement to get information about how MySQL processes a query. To use it, just add the keyword `EXPLAIN` to the front of your `SELECT` statement:

  ```
  mysql> EXPLAIN SELECT * FROM t1, t2 WHERE t1.i = t2.i;
  ```

  `EXPLAIN` is discussed in more detail in Section 13.8.2, “`EXPLAIN` Syntax”.

- Use `ANALYZE TABLE` *`tbl_name`* to update the key distributions for the scanned table. See Section 13.7.2.1, “`ANALYZE TABLE` Syntax”.

- Use `FORCE INDEX` for the scanned table to tell MySQL that table scans are very expensive compared to using the given index:

  ```
  SELECT * FROM t1, t2 FORCE INDEX (index_for_column)
  WHERE t1.col_name=t2.col_name;
  ```

  `USE INDEX` and `IGNORE INDEX` may also be useful. See Section 13.2.9.3, “Index Hint Syntax”.

- Global and table-level `STRAIGHT_JOIN`. See Section 13.2.9, “`SELECT` Syntax”.

- You can tune global or thread-specific system variables. For example, start `mysqld` with the `--max-seeks-for-key=1000` option or use `SET max_seeks_for_key=1000` to tell the optimizer to assume that no key scan causes more than 1,000 key seeks. See Section 5.1.4, “Server System Variables”.

C.5.7 Table Definition-Related Issues

C.5.7.1 Problems with `ALTER TABLE`

If you get a duplicate-key error when using `ALTER TABLE` to change the character set or collation of a character column, the cause is either that the new column collation maps two keys to the same value or that the table is corrupted. In the latter case, you should run `REPAIR TABLE` on the table.

If `ALTER TABLE` dies with the following error, the problem may be that MySQL crashed during an earlier `ALTER TABLE` operation and there is an old table named `A-`*`xxx`* or `B-`*`xxx`* lying around:

```
Error on rename of './database/name.frm'
to './database/B-xxx.frm' (Errcode: 17)
```

In this case, go to the MySQL data directory and delete all files that have names starting with `A-` or `B-`. (You may want to move them elsewhere instead of deleting them.)

`ALTER TABLE` works in the following way:

- Create a new table named `A-`*`xxx`* with the requested structural changes.

- Copy all rows from the original table to `A-xxx`.
- Rename the original table to `B-xxx`.
- Rename `A-xxx` to your original table name.
- Delete `B-xxx`.

If something goes wrong with the renaming operation, MySQL tries to undo the changes. If something goes seriously wrong (although this shouldn't happen), MySQL may leave the old table as `B-xxx`. A simple rename of the table files at the system level should get your data back.

If you use `ALTER TABLE` on a transactional table or if you are using Windows, `ALTER TABLE` unlocks the table if you had done a `LOCK TABLE` on it. This is done because `InnoDB` and these operating systems cannot drop a table that is in use.

C.5.7.2 `TEMPORARY` Table Problems

The following list indicates limitations on the use of `TEMPORARY` tables:

- A `TEMPORARY` table can only be of type `MEMORY`, `MyISAM`, `MERGE`, or `InnoDB`.
- You cannot refer to a `TEMPORARY` table more than once in the same query. For example, the following does not work:

  ```
  mysql> SELECT * FROM temp_table, temp_table AS t2;
  ERROR 1137: Can't reopen table: 'temp_table'
  ```

  This error also occurs if you refer to a temporary table multiple times in a stored function under different aliases, even if the references occur in different statements within the function.
- The `SHOW TABLES` statement does not list `TEMPORARY` tables.
- You cannot use `RENAME` to rename a `TEMPORARY` table. However, you can use `ALTER TABLE` instead:

  ```
  mysql> ALTER TABLE orig_name RENAME new_name;
  ```
- There are known issues in using temporary tables with replication. See Section 16.4.1, “Replication Features and Issues”, for more information.

C.5.8 Known Issues in MySQL

This section lists known issues in recent versions of MySQL.

For information about platform-specific issues, see the installation and porting instructions in Section 2.1, “General Installation Guidance”, and Section 22.4, “Debugging and Porting MySQL”.

The following problems are known:

- Subquery optimization for `IN` is not as effective as for `=`.
- Even if you use `lower_case_table_names=2` (which enables MySQL to remember the case used for databases and table names), MySQL does not remember the case used for database names for the function `DATABASE()` or within the various logs (on case-insensitive systems).
- Dropping a `FOREIGN KEY` constraint does not work in replication because the constraint may have another name on the slave.

- REPLACE (and LOAD DATA with the REPLACE option) does not trigger ON DELETE CASCADE.
- DISTINCT with ORDER BY does not work inside GROUP_CONCAT() if you do not use all and only those columns that are in the DISTINCT list.
- When inserting a big integer value (between 2^{63} and 2^{64}–1) into a decimal or string column, it is inserted as a negative value because the number is evaluated in a signed integer context.
- With statement-based binary logging, the master writes the executed queries to the binary log. This is a very fast, compact, and efficient logging method that works perfectly in most cases. However, it is possible for the data on the master and slave to become different if a query is designed in such a way that the data modification is nondeterministic (generally not a recommended practice, even outside of replication).

  For example:

  - CREATE TABLE ... SELECT or INSERT ... SELECT statements that insert zero or NULL values into an AUTO_INCREMENT column.
  - DELETE if you are deleting rows from a table that has foreign keys with ON DELETE CASCADE properties.
  - REPLACE ... SELECT, INSERT IGNORE ... SELECT if you have duplicate key values in the inserted data.

  If and only if the preceding queries have no ORDER BY clause guaranteeing a deterministic order.

  For example, for INSERT ... SELECT with no ORDER BY, the SELECT may return rows in a different order (which results in a row having different ranks, hence getting a different number in the AUTO_INCREMENT column), depending on the choices made by the optimizers on the master and slave.

  A query is optimized differently on the master and slave only if:

  - The table is stored using a different storage engine on the master than on the slave. (It is possible to use different storage engines on the master and slave. For example, you can use InnoDB on the master, but MyISAM on the slave if the slave has less available disk space.)
  - MySQL buffer sizes (key_buffer_size, and so on) are different on the master and slave.
  - The master and slave run different MySQL versions, and the optimizer code differs between these versions.

  This problem may also affect database restoration using mysqlbinlog|mysql.

  The easiest way to avoid this problem is to add an ORDER BY clause to the aforementioned nondeterministic queries to ensure that the rows are always stored or modified in the same order. Using row-based or mixed logging format also avoids the problem.

- Log file names are based on the server host name if you do not specify a file name with the startup option. To retain the same log file names if you change your host name to something else, you must explicitly use options such as --log-bin=*old_host_name*-bin. See Section 5.1.3, “Server Command Options”. Alternatively, rename the old files to reflect your host name change. If these are binary logs, you must edit the binary log index file and fix the binary log file names there as well. (The same is true for the relay logs on a slave server.)
- mysqlbinlog does not delete temporary files left after a LOAD DATA INFILE statement. See Section 4.6.7, “mysqlbinlog — Utility for Processing Binary Log Files”.

- `RENAME` does not work with `TEMPORARY` tables or tables used in a `MERGE` table.
- When using `SET CHARACTER SET`, you cannot use translated characters in database, table, and column names.
- You cannot use “`_`” or “`%`” with `ESCAPE` in `LIKE ... ESCAPE`.
- Only the first `max_sort_length` bytes are used when comparing data values. This means that values cannot reliably be used in `GROUP BY`, `ORDER BY` or `DISTINCT` if they are not distinct in the first `max_sort_length` bytes. To work around this, increase the variable value. The default value of `max_sort_length` is 1024 and can be changed at server startup time or at runtime.
- Numeric calculations are done with `BIGINT` or `DOUBLE` (both are normally 64 bits long). Which precision you get depends on the function. The general rule is that bit functions are performed with `BIGINT` precision, `IF()` and `ELT()` with `BIGINT` or `DOUBLE` precision, and the rest with `DOUBLE` precision. You should try to avoid using unsigned long long values if they resolve to be larger than 63 bits (9223372036854775807) for anything other than bit fields.
- You can have up to 255 `ENUM` and `SET` columns in one table.
- In `MIN()`, `MAX()`, and other aggregate functions, MySQL currently compares `ENUM` and `SET` columns by their string value rather than by the string's relative position in the set.
- In an `UPDATE` statement, columns are updated from left to right. If you refer to an updated column, you get the updated value instead of the original value. For example, the following statement increments `KEY` by `2`, **not** `1`:

  ```
  mysql> UPDATE tbl_name SET KEY=KEY+1,KEY=KEY+1;
  ```

- You can refer to multiple temporary tables in the same query, but you cannot refer to any given temporary table more than once. For example, the following does not work:

  ```
  mysql> SELECT * FROM temp_table, temp_table AS t2;
  ERROR 1137: Can't reopen table: 'temp_table'
  ```

- The optimizer may handle `DISTINCT` differently when you are using “hidden” columns in a join than when you are not. In a join, hidden columns are counted as part of the result (even if they are not shown), whereas in normal queries, hidden columns do not participate in the `DISTINCT` comparison.

  An example of this is:

  ```
  SELECT DISTINCT mp3id FROM band_downloads
         WHERE userid = 9 ORDER BY id DESC;
  ```

  and

  ```
  SELECT DISTINCT band_downloads.mp3id
         FROM band_downloads,band_mp3
         WHERE band_downloads.userid = 9
         AND band_mp3.id = band_downloads.mp3id
         ORDER BY band_downloads.id DESC;
  ```

  In the second case, using MySQL Server 3.23.x, you may get two identical rows in the result set (because the values in the hidden `id` column may differ).

  Note that this happens only for queries that do not have the `ORDER BY` columns in the result.

- If you execute a `PROCEDURE` on a query that returns an empty set, in some cases the `PROCEDURE` does not transform the columns.
- Creation of a table of type `MERGE` does not check whether the underlying tables are compatible types.
- If you use `ALTER TABLE` to add a `UNIQUE` index to a table used in a `MERGE` table and then add a normal index on the `MERGE` table, the key order is different for the tables if there was an old, non-`UNIQUE` key in the table. This is because `ALTER TABLE` puts `UNIQUE` indexes before normal indexes to be able to detect duplicate keys as early as possible.

Appendix D MySQL Release Notes

Release notes for MySQL Server and associated products are published in standalone form, not as part of the MySQL Reference Manual. For release notes, see these documents:

- MySQL 5.7 Release Notes
- MySQL Connector/ODBC Release Notes
- MySQL Connector/Net Release Notes (includes MySQL Visual Studio Plugin release notes)
- MySQL Connector/J Release Notes
- MySQL Connector/C++ Release Notes
- MySQL Proxy Release Notes
- MySQL for Excel Release Notes
- MySQL Installer Release Notes

Appendix E Restrictions and Limits

Table of Contents

The discussion here describes restrictions that apply to the use of MySQL features such as subqueries or views.

E.1 Restrictions on Stored Programs

These restrictions apply to the features described in Chapter 18, *Stored Programs and Views*.

Some of the restrictions noted here apply to all stored routines; that is, both to stored procedures and stored functions. There are also some restrictions specific to stored functions but not to stored procedures.

The restrictions for stored functions also apply to triggers. There are also some restrictions specific to triggers.

The restrictions for stored procedures also apply to the `DO` clause of Event Scheduler event definitions. There are also some restrictions specific to events.

SQL Statements Not Permitted in Stored Routines

Stored routines cannot contain arbitrary SQL statements. The following statements are not permitted:

- The locking statements `LOCK TABLES` and `UNLOCK TABLES`.
- `ALTER VIEW`.
- `LOAD DATA` and `LOAD TABLE`.
- SQL prepared statements (`PREPARE`, `EXECUTE`, `DEALLOCATE PREPARE`) can be used in stored procedures, but not stored functions or triggers. Thus, stored functions and triggers cannot use dynamic SQL (where you construct statements as strings and then execute them).
- Generally, statements not permitted in SQL prepared statements are also not permitted in stored programs. For a list of statements supported as prepared statements, see Section 13.5, “SQL Syntax for Prepared Statements”. Exceptions are `SIGNAL`, `RESIGNAL`, and `GET DIAGNOSTICS`, which are not permissible as prepared statements but are permitted in stored programs.

- Because local variables are in scope only during stored program execution, references to them are not permitted in prepared statements created within a stored program. Prepared statement scope is the current session, not the stored program, so the statement could be executed after the program ends, at which point the variables would no longer be in scope. For example, `SELECT ... INTO local_var` cannot be used as a prepared statement. This restriction also applies to stored procedure and function parameters. See Section 13.5.1, "PREPARE Syntax".

- Within all stored programs (stored procedures and functions, triggers, and events), the parser treats `BEGIN [WORK]` as the beginning of a `BEGIN ... END` block. To begin a transaction in this context, use `START TRANSACTION` instead.

Restrictions for Stored Functions

The following additional statements or operations are not permitted within stored functions. They are permitted within stored procedures, except stored procedures that are invoked from within a stored function or trigger. For example, if you use `FLUSH` in a stored procedure, that stored procedure cannot be called from a stored function or trigger.

- Statements that perform explicit or implicit commit or rollback. Support for these statements is not required by the SQL standard, which states that each DBMS vendor may decide whether to permit them.

- Statements that return a result set. This includes `SELECT` statements that do not have an `INTO var_list` clause and other statements such as `SHOW`, `EXPLAIN`, and `CHECK TABLE`. A function can process a result set either with `SELECT ... INTO var_list` or by using a cursor and `FETCH` statements. See Section 13.2.9.1, "`SELECT ... INTO` Syntax", and Section 13.6.6, "Cursors".

- `FLUSH` statements.

- Stored functions cannot be used recursively.

- A stored function or trigger cannot modify a table that is already being used (for reading or writing) by the statement that invoked the function or trigger.

- If you refer to a temporary table multiple times in a stored function under different aliases, a `Can't reopen table: 'tbl_name'` error occurs, even if the references occur in different statements within the function.

- `HANDLER ... READ` statements that invoke stored functions can cause replication errors and are disallowed.

Restrictions for Triggers

For triggers, the following additional restrictions apply:

- Triggers are not activated by foreign key actions.

- When using row-based replication, triggers on the slave are not activated by statements originating on the master. The triggers on the slave are activated when using statement-based replication. For more information, see Section 16.4.1.32, "Replication and Triggers".

- The `RETURN` statement is not permitted in triggers, which cannot return a value. To exit a trigger immediately, use the `LEAVE` statement.

- Triggers are not permitted on tables in the `mysql` database.

- The trigger cache does not detect when metadata of the underlying objects has changed. If a trigger uses a table and the table has changed since the trigger was loaded into the cache, the trigger operates using the outdated metadata.

Name Conflicts within Stored Routines

The same identifier might be used for a routine parameter, a local variable, and a table column. Also, the same local variable name can be used in nested blocks. For example:

```
CREATE PROCEDURE p (i INT)
BEGIN
  DECLARE i INT DEFAULT 0;
  SELECT i FROM t;
  BEGIN
    DECLARE i INT DEFAULT 1;
    SELECT i FROM t;
  END;
END;
```

In such cases, the identifier is ambiguous and the following precedence rules apply:

- A local variable takes precedence over a routine parameter or table column.
- A routine parameter takes precedence over a table column.
- A local variable in an inner block takes precedence over a local variable in an outer block.

The behavior that variables take precedence over table columns is nonstandard.

Replication Considerations

Use of stored routines can cause replication problems. This issue is discussed further in Section 18.7, “Binary Logging of Stored Programs”.

The `--replicate-wild-do-table=`*`db_name.tbl_name`* option applies to tables, views, and triggers. It does not apply to stored procedures and functions, or events. To filter statements operating on the latter objects, use one or more of the `--replicate-*-db` options.

Debugging Considerations

There are no stored routine debugging facilities.

Unsupported Syntax from the SQL:2003 Standard

The MySQL stored routine syntax is based on the SQL:2003 standard. The following items from that standard are not currently supported:

- `UNDO` handlers
- `FOR` loops

Concurrency Considerations

To prevent problems of interaction between sessions, when a client issues a statement, the server uses a snapshot of routines and triggers available for execution of the statement. That is, the server calculates a list of procedures, functions, and triggers that may be used during execution of the statement, loads them, and then proceeds to execute the statement. While the statement executes, it does not see changes to routines performed by other sessions.

For maximum concurrency, stored functions should minimize their side-effects; in particular, updating a table within a stored function can reduce concurrent operations on that table. A stored function acquires table locks before executing, to avoid inconsistency in the binary log due to mismatch of the order in which statements execute and when they appear in the log. When statement-based binary logging is used, statements that invoke a function are recorded rather than the statements executed within the function. Consequently, stored functions that update the same underlying tables do not execute in parallel. In contrast, stored procedures do not acquire table-level locks. All statements executed within stored procedures are written to the binary log, even for statement-based binary logging. See Section 18.7, “Binary Logging of Stored Programs”.

Event Scheduler Restrictions

The following limitations are specific to the Event Scheduler:

- Event names are handled in case-insensitive fashion. For example, you cannot have two events in the same database with the names `anEvent` and `AnEvent`.

- An event may not be created, altered, or dropped by a stored routine, trigger, or another event. An event also may not create, alter, or drop stored routines or triggers. (Bug #16409, Bug #18896)

- DDL statements on events are prohibited while a `LOCK TABLES` statement is in effect.

- Event timings using the intervals `YEAR`, `QUARTER`, `MONTH`, and `YEAR_MONTH` are resolved in months; those using any other interval are resolved in seconds. There is no way to cause events scheduled to occur at the same second to execute in a given order. In addition—due to rounding, the nature of threaded applications, and the fact that a nonzero length of time is required to create events and to signal their execution—events may be delayed by as much as 1 or 2 seconds. However, the time shown in the `INFORMATION_SCHEMA.EVENTS` table's `LAST_EXECUTED` column or the `mysql.event` table's `last_executed` column is always accurate to within one second of the actual event execution time. (See also Bug #16522.)

- Each execution of the statements contained in the body of an event takes place in a new connection; thus, these statements has no effect in a given user session on the server's statement counts such as `Com_select` and `Com_insert` that are displayed by `SHOW STATUS`. However, such counts *are* updated in the global scope. (Bug #16422)

- Events do not support times later than the end of the Unix Epoch; this is approximately the beginning of the year 2038. Such dates are specifically not permitted by the Event Scheduler. (Bug #16396)

- References to stored functions, user-defined functions, and tables in the `ON SCHEDULE` clauses of `CREATE EVENT` and `ALTER EVENT` statements are not supported. These sorts of references are not permitted. (See Bug #22830 for more information.)

E.2 Restrictions on Condition Handling

`SIGNAL`, `RESIGNAL`, and `GET DIAGNOSTICS` are not permissible as prepared statements. For example, this statement is invalid:

```
PREPARE stmt1 FROM 'SIGNAL SQLSTATE "02000"';
```

`SQLSTATE` values in class `'04'` are not treated specially. They are handled the same as other exceptions.

Standard SQL has a diagnostics area stack, containing a diagnostics area for each nested execution context. Standard SQL syntax includes `GET STACKED DIAGNOSTICS` for referring to stacked areas. MySQL does not support the `STACKED` keyword because there is a single diagnostics area containing information from the most recent statement that wrote to it. See also Section 13.6.7.7, “The MySQL Diagnostics Area”.

In standard SQL, the first condition relates to the `SQLSTATE` value returned for the previous SQL statement. In MySQL, this is not guaranteed, so to get the main error, you cannot do this:

```
GET DIAGNOSTICS CONDITION 1 @errno = MYSQL_ERRNO;
```

Instead, do this:

```
GET DIAGNOSTICS @cno = NUMBER;
GET DIAGNOSTICS CONDITION @cno @errno = MYSQL_ERRNO;
```

E.3 Restrictions on Server-Side Cursors

Server-side cursors are implemented in the C API using the `mysql_stmt_attr_set()` function. The same implementation is used for cursors in stored routines. A server-side cursor enables a result set to be generated on the server side, but not transferred to the client except for those rows that the client requests. For example, if a client executes a query but is only interested in the first row, the remaining rows are not transferred.

In MySQL, a server-side cursor is materialized into an internal temporary table. Initially, this is a `MEMORY` table, but is converted to a `MyISAM` table when its size exceeds the minimum value of the `max_heap_table_size` and `tmp_table_size` system variables. Note that the same restrictions apply to internal temporary tables created to hold the result set for a cursor as for other uses of internal temporary tables. See Section 8.4.4, “How MySQL Uses Internal Temporary Tables”. One limitation of the implementation is that for a large result set, retrieving its rows through a cursor might be slow.

Cursors are read only; you cannot use a cursor to update rows.

`UPDATE WHERE CURRENT OF` and `DELETE WHERE CURRENT OF` are not implemented, because updatable cursors are not supported.

Cursors are nonholdable (not held open after a commit).

Cursors are asensitive.

Cursors are nonscrollable.

Cursors are not named. The statement handler acts as the cursor ID.

You can have open only a single cursor per prepared statement. If you need several cursors, you must prepare several statements.

You cannot use a cursor for a statement that generates a result set if the statement is not supported in prepared mode. This includes statements such as `CHECK TABLE`, `HANDLER READ`, and `SHOW BINLOG EVENTS`.

E.4 Restrictions on Subqueries

- Subquery optimization for `IN` is not as effective as for the `=` operator or for the `IN(`*`value_list`*`)` operator.

  A typical case for poor `IN` subquery performance is when the subquery returns a small number of rows but the outer query returns a large number of rows to be compared to the subquery result.

  The problem is that, for a statement that uses an `IN` subquery, the optimizer rewrites it as a correlated subquery. Consider the following statement that uses an uncorrelated subquery:

  ```
  SELECT ... FROM t1 WHERE t1.a IN (SELECT b FROM t2);
  ```

  The optimizer rewrites the statement to a correlated subquery:

```
SELECT ... FROM t1 WHERE EXISTS (SELECT 1 FROM t2 WHERE t2.b = t1.a);
```

If the inner and outer queries return *M* and *N* rows, respectively, the execution time becomes on the order of O(*M*×*N*), rather than O(*M*+*N*) as it would be for an uncorrelated subquery.

An implication is that an `IN` subquery can be much slower than a query written using an `IN(`*`value_list`*`)` operator that lists the same values that the subquery would return.

- In general, you cannot modify a table and select from the same table in a subquery. For example, this limitation applies to statements of the following forms:

  ```
  DELETE FROM t WHERE ... (SELECT ... FROM t ...);
  UPDATE t ... WHERE col = (SELECT ... FROM t ...);
  {INSERT|REPLACE} INTO t (SELECT ... FROM t ...);
  ```

  Exception: The preceding prohibition does not apply if you are using a subquery for the modified table in the `FROM` clause. Example:

  ```
  UPDATE t ... WHERE col = (SELECT * FROM (SELECT ... FROM t...) AS _t ...);
  ```

  Here the result from the subquery in the `FROM` clause is stored as a temporary table, so the relevant rows in `t` have already been selected by the time the update to `t` takes place.

- Row comparison operations are only partially supported:

  - For *`expr`* `[NOT] IN` *`subquery`*, *`expr`* can be an *n*-tuple (specified using row constructor syntax) and the subquery can return rows of *n*-tuples. The permitted syntax is therefore more specifically expressed as *`row_constructor`* `[NOT] IN` *`table_subquery`*

  - For *`expr`* *`op`* `{ALL|ANY|SOME}` *`subquery`*, *`expr`* must be a scalar value and the subquery must be a column subquery; it cannot return multiple-column rows.

  In other words, for a subquery that returns rows of *n*-tuples, this is supported:

  ```
  (expr_1, ..., expr_n) [NOT] IN table_subquery
  ```

  But this is not supported:

  ```
  (expr_1, ..., expr_n) op {ALL|ANY|SOME} subquery
  ```

  The reason for supporting row comparisons for `IN` but not for the others is that `IN` is implemented by rewriting it as a sequence of `=` comparisons and `AND` operations. This approach cannot be used for `ALL`, `ANY`, or `SOME`.

- Subqueries in the `FROM` clause cannot be correlated subqueries. They are materialized in whole (evaluated to produce a result set) during query execution, so they cannot be evaluated per row of the outer query. In MySQL 5.7, the optimizer delays materialization until the result is needed, which may permit materialization to be avoided. See Optimizing Subqueries in the `FROM` Clause (Derived Tables).

- MySQL does not support `LIMIT` in subqueries for certain subquery operators:

  ```
  mysql> SELECT * FROM t1
      ->   WHERE s1 IN (SELECT s2 FROM t2 ORDER BY s1 LIMIT 1);
  ERROR 1235 (42000): This version of MySQL doesn't yet support
  ```

```
 'LIMIT & IN/ALL/ANY/SOME subquery'
```

- The optimizer is more mature for joins than for subqueries, so in many cases a statement that uses a subquery can be executed more efficiently if you rewrite it as a join.

  An exception occurs for the case where an `IN` subquery can be rewritten as a `SELECT DISTINCT` join. Example:

  ```
  SELECT col FROM t1 WHERE id_col IN (SELECT id_col2 FROM t2 WHERE condition);
  ```

  That statement can be rewritten as follows:

  ```
  SELECT DISTINCT col FROM t1, t2 WHERE t1.id_col = t2.id_col AND condition;
  ```

- MySQL permits a subquery to refer to a stored function that has data-modifying side effects such as inserting rows into a table. For example, if `f()` inserts rows, the following query can modify data:

  ```
  SELECT ... WHERE x IN (SELECT f() ...);
  ```

  This behavior is an extension to the SQL standard. In MySQL, it can produce indeterminate results because `f()` might be executed a different number of times for different executions of a given query depending on how the optimizer chooses to handle it.

  For statement-based or mixed-format replication, one implication of this indeterminism is that such a query can produce different results on the master and its slaves.

- In MySQL 5.7, the optimizer creates an index on the materialized table if this will result in faster query execution. See Optimizing Subqueries in the `FROM` Clause (Derived Tables).

E.5 Restrictions on Views

View processing is not optimized:

- It is not possible to create an index on a view.

- Indexes can be used for views processed using the merge algorithm. However, a view that is processed with the temptable algorithm is unable to take advantage of indexes on its underlying tables (although indexes can be used during generation of the temporary tables).

Subqueries cannot be used in the `FROM` clause of a view.

There is a general principle that you cannot modify a table and select from the same table in a subquery. See Section E.4, “Restrictions on Subqueries”.

The same principle also applies if you select from a view that selects from the table, if the view selects from the table in a subquery and the view is evaluated using the merge algorithm. Example:

```
CREATE VIEW v1 AS
SELECT * FROM t2 WHERE EXISTS (SELECT 1 FROM t1 WHERE t1.a = t2.a);

UPDATE t1, v2 SET t1.a = 1 WHERE t1.b = v2.b;
```

If the view is evaluated using a temporary table, you *can* select from the table in the view subquery and still modify that table in the outer query. In this case the view will be stored in a temporary table and thus you are not really selecting from the table in a subquery and modifying it “at the same time.” (This is another reason you might wish to force MySQL to use the temptable algorithm by specifying ALGORITHM = TEMPTABLE in the view definition.)

You can use DROP TABLE or ALTER TABLE to drop or alter a table that is used in a view definition. No warning results from the DROP or ALTER operation, even though this invalidates the view. Instead, an error occurs later, when the view is used. CHECK TABLE can be used to check for views that have been invalidated by DROP or ALTER operations.

With regard to view updatability, the overall goal for views is that if any view is theoretically updatable, it should be updatable in practice. This includes views that have UNION in their definition. Currently, not all views that are theoretically updatable can be updated. The initial view implementation was deliberately written this way to get usable, updatable views into MySQL as quickly as possible. Many theoretically updatable views can be updated now, but limitations still exist:

- Updatable views with subqueries anywhere other than in the WHERE clause. Some views that have subqueries in the SELECT list may be updatable.
- You cannot use UPDATE to update more than one underlying table of a view that is defined as a join.
- You cannot use DELETE to update a view that is defined as a join.

There exists a shortcoming with the current implementation of views. If a user is granted the basic privileges necessary to create a view (the CREATE VIEW and SELECT privileges), that user will be unable to call SHOW CREATE VIEW on that object unless the user is also granted the SHOW VIEW privilege.

That shortcoming can lead to problems backing up a database with mysqldump, which may fail due to insufficient privileges. This problem is described in Bug #22062.

The workaround to the problem is for the administrator to manually grant the SHOW VIEW privilege to users who are granted CREATE VIEW, since MySQL doesn't grant it implicitly when views are created.

Views do not have indexes, so index hints do not apply. Use of index hints when selecting from a view is not permitted.

SHOW CREATE VIEW displays view definitions using an AS *alias_name* clause for each column. If a column is created from an expression, the default alias is the expression text, which can be quite long. Aliases for column names in CREATE VIEW statements are checked against the maximum column length of 64 characters (not the maximum alias length of 256 characters). As a result, views created from the output of SHOW CREATE VIEW fail if any column alias exceeds 64 characters. This can cause problems in the following circumstances for views with too-long aliases:

- View definitions fail to replicate to newer slaves that enforce the column-length restriction.
- Dump files created with mysqldump cannot be loaded into servers that enforce the column-length restriction.

A workaround for either problem is to modify each problematic view definition to use aliases that provide shorter column names. Then the view will replicate properly, and can be dumped and reloaded without causing an error. To modify the definition, drop and create the view again with DROP VIEW and CREATE VIEW, or replace the definition with CREATE OR REPLACE VIEW.

For problems that occur when reloading view definitions in dump files, another workaround is to edit the dump file to modify its CREATE VIEW statements. However, this does not change the original view definitions, which may cause problems for subsequent dump operations.

E.6 Restrictions on XA Transactions

XA transaction support is limited to the InnoDB storage engine.

For “external XA,” a MySQL server acts as a Resource Manager and client programs act as Transaction Managers. For “Internal XA”, storage engines within a MySQL server act as RMs, and the server itself acts as a TM. Internal XA support is limited by the capabilities of individual storage engines. Internal XA is required for handling XA transactions that involve more than one storage engine. The implementation of internal XA requires that a storage engine support two-phase commit at the table handler level, and currently this is true only for InnoDB.

For XA START, the JOIN and RESUME clauses are not supported.

For XA END, the SUSPEND [FOR MIGRATE] clause is not supported.

The requirement that the *bqual* part of the *xid* value be different for each XA transaction within a global transaction is a limitation of the current MySQL XA implementation. It is not part of the XA specification.

If an XA transaction has reached the PREPARED state and the MySQL server is killed (for example, with kill -9 on Unix) or shuts down abnormally, the transaction can be continued after the server restarts. However, if the client reconnects and commits the transaction, the transaction will be absent from the binary log even though it has been committed. This means the data and the binary log have gone out of synchrony. An implication is that XA cannot be used safely together with replication.

It is possible that the server will roll back a pending XA transaction, even one that has reached the PREPARED state. This happens if a client connection terminates and the server continues to run, or if clients are connected and the server shuts down gracefully. (In the latter case, the server marks each connection to be terminated, and then rolls back the PREPARED XA transaction associated with it.) It should be possible to commit or roll back a PREPARED XA transaction, but this cannot be done without changes to the binary logging mechanism.

E.7 Restrictions on Character Sets

- Identifiers are stored in mysql database tables (user, db, and so forth) using utf8, but identifiers can contain only characters in the Basic Multilingual Plane (BMP). Supplementary characters are not permitted in identifiers.
- The ucs2, utf16, utf16le, and utf32 character sets have the following restrictions:
  - They cannot be used as a client character set, which means that they do not work for SET NAMES or SET CHARACTER SET. (See Section 10.1.4, “Connection Character Sets and Collations”.)
  - It is currently not possible to use LOAD DATA INFILE to load data files that use these character sets.
  - FULLTEXT indexes cannot be created on a column that uses any of these character sets. However, you can perform IN BOOLEAN MODE searches on the column without an index.
  - The use of ENCRYPT() with these character sets is not recommended because the underlying system call expects a string terminated by a zero byte.
- The REGEXP and RLIKE operators work in byte-wise fashion, so they are not multi-byte safe and may produce unexpected results with multi-byte character sets. In addition, these operators compare characters by their byte values and accented characters may not compare as equal even if a given collation treats them as equal.

E.8 Restrictions on Performance Schema

The Performance Schema avoids using mutexes to collect or produce data, so there are no guarantees of consistency and results can sometimes be incorrect. Event values in `performance_schema` tables are nondeterministic and nonrepeatable.

If you save event information in another table, you should not assume that the original events will still be available later. For example, if you select events from a `performance_schema` table into a temporary table, intending to join that table with the original table later, there might be no matches.

`mysqldump` and `BACKUP DATABASE` ignore tables in the `performance_schema` database.

Tables in the `performance_schema` database cannot be locked with `LOCK TABLES`, except the `setup_xxx` tables.

Tables in the `performance_schema` database cannot be indexed.

Results for queries that refer to tables in the `performance_schema` database are not saved in the query cache.

Tables in the `performance_schema` database are not replicated.

The Performance Schema is not available in `libmysqld`, the embedded server.

The types of timers might vary per platform. The `performance_timers` table shows which event timers are available. If the values in this table for a given timer name are `NULL`, that timer is not supported on your platform.

Instruments that apply to storage engines might not be implemented for all storage engines. Instrumentation of each third-party engine is the responsibility of the engine maintainer.

E.9 Restrictions on Pluggable Authentication

The first part of this section describes general restrictions on the applicability of the pluggable authentication framework described at Section 6.3.8, “Pluggable Authentication”. The second part describes how third-party connector developers can determine the extent to which a connector can take advantage of pluggable authentication capabilities and what steps to take to become more compliant.

The term “native authentication” used here refers to authentication against passwords stored in the `Password` column of the `mysql.user` table. This is the same authentication method provided by older MySQL servers, before pluggable authentication was implemented. It remains the default method, although now it is implemented using plugins. “Windows native authentication” refers to authentication using the credentials of a user who has already logged in to Windows, as implemented by the Windows Native Authentication plugin (“Windows plugin” for short).

General Pluggable Authentication Restrictions

- **Connector/C, Connector/C++:** Clients that use these connectors can connect to the server only through accounts that use native authentication.

  Exception: A connector supports pluggable authentication if it was built to link to `libmysqlclient` dynamically (rather than statically) and it loads the current version of `libmysqlclient` if that version is installed, or if the connector is recompiled from source to link against the current `libmysqlclient`.

- **Connector/J:** Clients that use this connector can connect to the server only through accounts that use native authentication.

- **Connector/Net:** Before Connector/Net 6.4.4, clients that use this connector can connect to the server only through accounts that use native authentication. As of 6.4.4, clients can also connect to the server through accounts that use the Windows plugin.

- **Connector/ODBC:** Before Connector/ODBC 3.51.29 and 5.1.9, clients that use this connector can connect to the server only through accounts that use native authentication. As of 3.51.29 and 5.1.9, clients that use binary releases of this connector for Windows can also connect to the server through accounts that use the PAM or Windows plugins. (These capabilities result from linking the Connector/ODBC binaries against the MySQL 5.5.16 `libmysqlclient` rather than the MySQL 5.1 `libmysqlclient` used previously. The newer `libmysqlclient` includes the client-side support needed for the server-side PAM and Windows authentication plugins.)

- **Connector/PHP:** Clients that use this connector can connect to the server only through accounts that use native authentication, when compiled using the MySQL native driver for PHP (`mysqlnd`).

- **MySQL Proxy:** Before MySQL Proxy 0.8.2, clients can connect to the server only through accounts that use native authentication. As of 0.8.2, clients can also connect to the server through accounts that use the PAM plugin. As of 0.8.3, clients can also connect to the server through accounts that use the Windows plugin.

- **MySQL Enterprise Backup:** MySQL Enterprise Backup before version 3.6.1 supports connections to the server only through accounts that use native authentication. As of 3.6.1, MySQL Enterprise Backup can connect to the server through accounts that use nonnative authentication.

- **Windows native authentication:** Connecting through an account that uses the Windows plugin requires Windows Domain setup. Without it, NTLM authentication is used and then only local connections are possible; that is, the client and server must run on the same computer.

- **Proxy users:** Proxy user support is available to the extent that clients can connect through accounts authenticated with plugins that implement proxy user capability (that is, plugins that can return a user name different from that of the connecting user). For example, the native authentication plugins do not support proxy users, whereas the PAM and Windows plugins do.

- **Replication**: Replication slaves can employ not only master accounts using native authentication, but can also connect through master accounts that use nonnative authentication if the required client-side plugin is available. If the plugin is built into `libmysqlclient`, it is available by default. Otherwise, the plugin must be installed on the slave side in the directory named by the slave `plugin_dir` system variable.

- **`FEDERATED` tables:** A `FEDERATED` table can access the remote table only through accounts on the remote server that use native authentication.

Pluggable Authentication and Third-Party Connectors

Third-party connector developers can use the following guidelines to determine readiness of a connector to take advantage of pluggable authentication capabilities and what steps to take to become more compliant:

- An existing connector to which no changes have been made uses native authentication and clients that use the connector can connect to the server only through accounts that use native authentication. *However, you should test the connector against a recent version of the server to verify that such connections still work without problem.*

  Exception: A connector might work with pluggable authentication without any changes if it links to `libmysqlclient` dynamically (rather than statically) and it loads the current version of `libmysqlclient` if that version is installed.

- To take advantage of pluggable authentication capabilities, a connector that is `libmysqlclient`-based should be relinked against the current version of `libmysqlclient`. This enables the connector to support connections though accounts that require client-side plugins now built into `libmysqlclient` (such as the cleartext plugin needed for PAM authentication and the Windows plugin needed for

Windows native authentication). Linking with a current libmysqlclient also enables the connector to access client-side plugins installed in the default MySQL plugin directory (typically the directory named by the default value of the local server's plugin_dir system variable).

If a connector links to libmysqlclient dynamically, it must be ensured that the newer version of libmysqlclient is installed on the client host and that the connector loads it at runtime.

- Another way for a connector to support a given authentication method is to implement it directly in the client/server protocol. Connector/Net uses this approach to provide support for Windows native authentication.

- If a connector should be able to load client-side plugins from a directory different from the default plugin directory, it must implement some means for client users to specify the directory. Possibilities for this include a command-line option or environment variable from which the connector can obtain the directory name. Standard MySQL client programs such as mysql and mysqladmin implement a --plugin-dir option. See also Section 21.8.14, “C API Client Plugin Functions”.

- Proxy user support by a connector depends, as described earlier in this section, on whether the authentication methods that it supports permit proxy users.

E.10 Limits in MySQL

This section lists current limits in MySQL 5.7.

E.10.1 Limits of Joins

The maximum number of tables that can be referenced in a single join is 61. This also applies to the number of tables that can be referenced in the definition of a view.

E.10.2 Limits on Number of Databases and Tables

MySQL has no limit on the number of databases. The underlying file system may have a limit on the number of directories.

MySQL has no limit on the number of tables. The underlying file system may have a limit on the number of files that represent tables. Individual storage engines may impose engine-specific constraints. InnoDB permits up to 4 billion tables.

E.10.3 Limits on Table Size

The effective maximum table size for MySQL databases is usually determined by operating system constraints on file sizes, not by MySQL internal limits. The following table lists some examples of operating system file-size limits. This is only a rough guide and is not intended to be definitive. For the most up-to-date information, be sure to check the documentation specific to your operating system.

Operating System	File-size Limit
Win32 w/ FAT/FAT32	2GB/4GB
Win32 w/ NTFS	2TB (possibly larger)
Linux 2.2-Intel 32-bit	2GB (LFS: 4GB)
Linux 2.4+	(using ext3 file system) 4TB
Solaris 9/10	16TB
Mac OS X w/ HFS+	2TB

Windows users, please note that FAT and VFAT (FAT32) are *not* considered suitable for production use with MySQL. Use NTFS instead.

On Linux 2.2, you can get MyISAM tables larger than 2GB in size by using the Large File Support (LFS) patch for the ext2 file system. Most current Linux distributions are based on kernel 2.4 or higher and include all the required LFS patches. On Linux 2.4, patches also exist for ReiserFS to get support for big files (up to 2TB). With JFS and XFS, petabyte and larger files are possible on Linux.

For a detailed overview about LFS in Linux, have a look at Andreas Jaeger's *Large File Support in Linux* page at http://www.suse.de/~aj/linux_lfs.html.

If you do encounter a full-table error, there are several reasons why it might have occurred:

- The disk might be full.

- The InnoDB storage engine maintains InnoDB tables within a tablespace that can be created from several files. This enables a table to exceed the maximum individual file size. The tablespace can include raw disk partitions, which permits extremely large tables. The maximum tablespace size is 64TB.

  If you are using InnoDB tables and run out of room in the InnoDB tablespace. In this case, the solution is to extend the InnoDB tablespace. See Section 14.2.5.7, “Changing the Number or Size of InnoDB Log Files and Resizing the InnoDB Tablespace”.

- You are using MyISAM tables on an operating system that supports files only up to 2GB in size and you have hit this limit for the data file or index file.

- You are using a MyISAM table and the space required for the table exceeds what is permitted by the internal pointer size. MyISAM permits data and index files to grow up to 256TB by default, but this limit can be changed up to the maximum permissible size of 65,536TB (256^7 – 1 bytes).

  If you need a MyISAM table that is larger than the default limit and your operating system supports large files, the CREATE TABLE statement supports AVG_ROW_LENGTH and MAX_ROWS options. See Section 13.1.14, “CREATE TABLE Syntax”. The server uses these options to determine how large a table to permit.

  If the pointer size is too small for an existing table, you can change the options with ALTER TABLE to increase a table's maximum permissible size. See Section 13.1.6, “ALTER TABLE Syntax”.

  ```
  ALTER TABLE tbl_name MAX_ROWS=1000000000 AVG_ROW_LENGTH=nnn;
  ```

  You have to specify AVG_ROW_LENGTH only for tables with BLOB or TEXT columns; in this case, MySQL can't optimize the space required based only on the number of rows.

  To change the default size limit for MyISAM tables, set the myisam_data_pointer_size, which sets the number of bytes used for internal row pointers. The value is used to set the pointer size for new tables if you do not specify the MAX_ROWS option. The value of myisam_data_pointer_size can be from 2 to 7. A value of 4 permits tables up to 4GB; a value of 6 permits tables up to 256TB.

  You can check the maximum data and index sizes by using this statement:

  ```
  SHOW TABLE STATUS FROM db_name LIKE 'tbl_name';
  ```

  You also can use myisamchk -dv /path/to/table-index-file. See Section 13.7.5, “SHOW Syntax”, or Section 4.6.3, “myisamchk — MyISAM Table-Maintenance Utility”.

  Other ways to work around file-size limits for MyISAM tables are as follows:

- If your large table is read only, you can use myisampack to compress it. myisampack usually compresses a table by at least 50%, so you can have, in effect, much bigger tables. myisampack also can merge multiple tables into a single table. See Section 4.6.5, “myisampack — Generate Compressed, Read-Only MyISAM Tables”.
- MySQL includes a MERGE library that enables you to handle a collection of MyISAM tables that have identical structure as a single MERGE table. See Section 14.8, “The MERGE Storage Engine”.

- You are using the MEMORY (HEAP) storage engine; in this case you need to increase the value of the max_heap_table_size system variable. See Section 5.1.4, “Server System Variables”.

E.10.4 Limits on Table Column Count and Row Size

There is a hard limit of 4096 columns per table, but the effective maximum may be less for a given table. The exact limit depends on several interacting factors.

- Every table (regardless of storage engine) has a maximum row size of 65,535 bytes. Storage engines may place additional constraints on this limit, reducing the effective maximum row size.

  The maximum row size constrains the number (and possibly size) of columns because the total length of all columns cannot exceed this size. For example, utf8 characters require up to three bytes per character, so for a CHAR(255) CHARACTER SET utf8 column, the server must allocate 255 × 3 = 765 bytes per value. Consequently, a table cannot contain more than 65,535 / 765 = 85 such columns.

  Storage for variable-length columns includes length bytes, which are assessed against the row size. For example, a VARCHAR(255) CHARACTER SET utf8 column takes two bytes to store the length of the value, so each value can take up to 767 bytes.

  BLOB and TEXT columns count from one to four plus eight bytes each toward the row-size limit because their contents are stored separately from the rest of the row.

  Declaring columns NULL can reduce the maximum number of columns permitted. For MyISAM tables, NULL columns require additional space in the row to record whether their values are NULL. Each NULL column takes one bit extra, rounded up to the nearest byte. The maximum row length in bytes can be calculated as follows:

  ```
  row length = 1
               + (sum of column lengths)
               + (number of NULL columns + delete_flag + 7)/8
               + (number of variable-length columns)
  ```

  delete_flag is 1 for tables with static row format. Static tables use a bit in the row record for a flag that indicates whether the row has been deleted. *delete_flag* is 0 for dynamic tables because the flag is stored in the dynamic row header. For information about MyISAM table formats, see Section 14.3.3, “MyISAM Table Storage Formats”.

  For InnoDB tables, storage size is the same for NULL and NOT NULL columns, so the preceding calculations do not apply.

  The following statement to create table t1 succeeds because the columns require 32,765 + 2 bytes and 32,766 + 2 bytes, which falls within the maximum row size of 65,535 bytes:

  ```
  mysql> CREATE TABLE t1
      -> (c1 VARCHAR(32765) NOT NULL, c2 VARCHAR(32766) NOT NULL)
      -> ENGINE = MyISAM CHARACTER SET latin1;
  ```

```
Query OK, 0 rows affected (0.02 sec)
```

The following statement to create table t2 fails because the columns are NULL and MyISAM requires additional space that causes the row size to exceed 65,535 bytes:

```
mysql> CREATE TABLE t2
    -> (c1 VARCHAR(32765) NULL, c2 VARCHAR(32766) NULL)
    -> ENGINE = MyISAM CHARACTER SET latin1;
ERROR 1118 (42000): Row size too large. The maximum row size for the
used table type, not counting BLOBs, is 65535. You have to change some
columns to TEXT or BLOBs
```

The following statement to create table t3 fails because, although the column length is within the maximum length of 65,535 bytes, two additional bytes are required to record the length, which causes the row size to exceed 65,535 bytes:

```
mysql> CREATE TABLE t3
    -> (c1 VARCHAR(65535) NOT NULL)
    -> ENGINE = MyISAM CHARACTER SET latin1;
ERROR 1118 (42000): Row size too large. The maximum row size for the
used table type, not counting BLOBs, is 65535. You have to change some
columns to TEXT or BLOBs
```

Reducing the column length to 65,533 or less permits the statement to succeed.

- Individual storage engines might impose additional restrictions that limit table column count. Examples:
  - InnoDB permits up to 1000 columns.
  - InnoDB restricts row size to something less than half a database page (approximately 8000 bytes), not including VARBINARY, VARCHAR, BLOB, or TEXT columns.
  - Different InnoDB storage formats (COMPRESSED, REDUNDANT) use different amounts of page header and trailer data, which affects the amount of storage available for rows.
- Each table has an .frm file that contains the table definition. The definition affects the content of this file in ways that may affect the number of columns permitted in the table. For more information, see Section E.10.5, “Limits Imposed by .frm File Structure”.

E.10.5 Limits Imposed by .frm File Structure

Each table has an .frm file that contains the table definition. The server uses the following expression to check some of the table information stored in the file against an upper limit of 64KB:

```
if (info_length+(ulong) create_fields.elements*FCOMP+288+
    n_length+int_length+com_length > 65535L || int_count > 255)
```

The portion of the information stored in the .frm file that is checked against the expression cannot grow beyond the 64KB limit, so if the table definition reaches this size, no more columns can be added.

The relevant factors in the expression are:

- info_length is space needed for “screens.” This is related to MySQL's Unireg heritage.
- create_fields.elements is the number of columns.
- FCOMP is 17.

- `n_length` is the total length of all column names, including one byte per name as a separator.
- `int_length` is related to the list of values for `ENUM` and `SET` columns. In this context, “int” does not mean “integer.” It means “interval,” a term that refers collectively to `ENUM` and `SET` columns.
- `int_count` is the number of unique `ENUM` and `SET` definitions.
- `com_length` is the total length of column comments.

The expression just described has several implications for permitted table definitions:

- Using long column names can reduce the maximum number of columns, as can the inclusion of `ENUM` or `SET` columns, or use of column comments.
- A table can have no more than 255 unique `ENUM` and `SET` definitions. Columns with identical element lists are considered the same against this limt. For example, if a table contains these two columns, they count as one (not two) toward this limit because the definitions are identical:

  ```
  e1 ENUM('a','b','c')
  e2 ENUM('a','b','c')
  ```
- The sum of the length of element names in the unique `ENUM` and `SET` definitions counts toward the 64KB limit, so although the theoretical limit on number of elements in a given `ENUM` column is 65,535, the practical limit is less than 3000.

E.10.6 Windows Platform Limitations

The following limitations apply to use of MySQL on the Windows platform:

- **Process memory**

  On Windows 32-bit platforms, it is not possible by default to use more than 2GB of RAM within a single process, including MySQL. This is because the physical address limit on Windows 32-bit is 4GB and the default setting within Windows is to split the virtual address space between kernel (2GB) and user/applications (2GB).

  Some versions of Windows have a boot time setting to enable larger applications by reducing the kernel application. Alternatively, to use more than 2GB, use a 64-bit version of Windows.
- **File system aliases**

  When using `MyISAM` tables, you cannot use aliases within Windows link to the data files on another volume and then link back to the main MySQL `datadir` location.

  This facility is often used to move the data and index files to a RAID or other fast solution, while retaining the main `.frm` files in the default data directory configured with the `datadir` option.
- **Limited number of ports**

  Windows systems have about 4,000 ports available for client connections, and after a connection on a port closes, it takes two to four minutes before the port can be reused. In situations where clients connect to and disconnect from the server at a high rate, it is possible for all available ports to be used up before closed ports become available again. If this happens, the MySQL server appears to be unresponsive even though it is running. Note that ports may be used by other applications running on the machine as well, in which case the number of ports available to MySQL is lower.

  For more information about this problem, see http://support.microsoft.com/default.aspx?scid=kb;en-us;196271.

- **`DATA DIRECTORY` and `INDEX DIRECTORY`**

  The `DATA DIRECTORY` option for `CREATE TABLE` is supported on Windows only for `InnoDB` tables, as described in Section 14.2.5.4, “Specifying the Location of a Tablespace”. For `MyISAM` and other storage engines, the `DATA DIRECTORY` and `INDEX DIRECTORY` options for `CREATE TABLE` are ignored on Windows and any other platforms with a nonfunctional `realpath()` call.

- **`DROP DATABASE`**

  You cannot drop a database that is in use by another session.

- **Case-insensitive names**

  File names are not case sensitive on Windows, so MySQL database and table names are also not case sensitive on Windows. The only restriction is that database and table names must be specified using the same case throughout a given statement. See Section 9.2.2, “Identifier Case Sensitivity”.

- **Directory and file names**

  On Windows, MySQL Server supports only directory and file names that are compatible with the current ANSI code pages. For example, the following Japanese directory name will not work in the Western locale (code page 1252):

  ```
  datadir="C:/私たちのプロジェクトのデータ"
  ```

  The same limitation applies to directory and file names referred to in SQL statements, such as the data file path name in `LOAD DATA INFILE`.

- **The “`\`” path name separator character**

  Path name components in Windows are separated by the “`\`” character, which is also the escape character in MySQL. If you are using `LOAD DATA INFILE` or `SELECT ... INTO OUTFILE`, use Unix-style file names with “`/`” characters:

  ```
  mysql> LOAD DATA INFILE 'C:/tmp/skr.txt' INTO TABLE skr;
  mysql> SELECT * INTO OUTFILE 'C:/tmp/skr.txt' FROM skr;
  ```

  Alternatively, you must double the “`\`” character:

  ```
  mysql> LOAD DATA INFILE 'C:\\tmp\\skr.txt' INTO TABLE skr;
  mysql> SELECT * INTO OUTFILE 'C:\\tmp\\skr.txt' FROM skr;
  ```

- **Problems with pipes**

  Pipes do not work reliably from the Windows command-line prompt. If the pipe includes the character `^Z` / `CHAR(24)`, Windows thinks that it has encountered end-of-file and aborts the program.

  This is mainly a problem when you try to apply a binary log as follows:

  ```
  C:\> mysqlbinlog binary_log_file | mysql --user=root
  ```

  If you have a problem applying the log and suspect that it is because of a `^Z` / `CHAR(24)` character, you can use the following workaround:

  ```
  C:\> mysqlbinlog binary_log_file --result-file=/tmp/bin.sql
  ```

```
C:\> mysql --user=root --execute "source /tmp/bin.sql"
```

The latter command also can be used to reliably read in any SQL file that may contain binary data.

MySQL Glossary

These terms are commonly used in information about the MySQL database server. This glossary originated as a reference for terminology about the InnoDB storage engine, and the majority of definitions are InnoDB-related.

A

.ARM file

Metadata for ARCHIVE tables. Contrast with **.ARZ file**. Files with this extension are always included in backups produced by the `mysqlbackup` command of the **MySQL Enterprise Backup** product.
See Also .ARZ file, MySQL Enterprise Backup, mysqlbackup command.

.ARZ file

Data for ARCHIVE tables. Contrast with **.ARM file**. Files with this extension are always included in backups produced by the `mysqlbackup` command of the **MySQL Enterprise Backup** product.
See Also .ARM file, MySQL Enterprise Backup, mysqlbackup command.

ACID

An acronym standing for atomicity, consistency, isolation, and durability. These properties are all desirable in a database system, and are all closely tied to the notion of a **transaction**. The transactional features of InnoDB adhere to the ACID principles.

Transactions are **atomic** units of work that can be **committed** or **rolled back**. When a transaction makes multiple changes to the database, either all the changes succeed when the transaction is committed, or all the changes are undone when the transaction is rolled back.

The database remains in a consistent state at all times -- after each commit or rollback, and while transactions are in progress. If related data is being updated across multiple tables, queries see either all old values or all new values, not a mix of old and new values.

Transactions are protected (isolated) from each other while they are in progress; they cannot interfere with each other or see each other's uncommitted data. This isolation is achieved through the **locking** mechanism. Experienced users can adjust the **isolation level**, trading off less protection in favor of increased performance and **concurrency**, when they can be sure that the transactions really do not interfere with each other.

The results of transactions are durable: once a commit operation succeeds, the changes made by that transaction are safe from power failures, system crashes, race conditions, or other potential dangers that many non-database applications are vulnerable to. Durability typically involves writing to disk storage, with a certain amount of redundancy to protect against power failures or software crashes during write operations. (In InnoDB, the **doublewrite buffer** assists with durability.)
See Also atomic, commit, concurrency, doublewrite buffer, isolation level, locking, rollback, transaction.

adaptive flushing

An algorithm for **InnoDB** tables that smooths out the I/O overhead introduced by **checkpoints**. Instead of **flushing** all modified **pages** from the **buffer pool** to the **data files** at once, MySQL periodically flushes small sets of modified pages. The adaptive flushing algorithm extends this process by estimating the optimal rate to perform these periodic flushes, based on the rate of flushing and how fast **redo** information is generated. First introduced in MySQL 5.1, in the InnoDB Plugin.
See Also buffer pool, checkpoint, data files, flush, InnoDB, page, redo log.

adaptive hash index

An optimization for InnoDB tables that can speed up lookups using `=` and `IN` operators, by constructing a **hash index** in memory. MySQL monitors index searches for InnoDB tables, and if queries could benefit from a hash index, it builds one automatically for index **pages** that are frequently accessed. In a sense, the adaptive hash index configures MySQL at runtime to take advantage of ample main memory, coming closer to the architecture of main-memory databases. This feature is controlled by the `innodb_adaptive_hash_index` configuration option. Because this feature benefits some workloads and not others, and the memory used for the hash index is reserved in the **buffer pool**, typically you should benchmark with this feature both enabled and disabled.

The hash index is always built based on an existing InnoDB **secondary index**, which is organized as a **B-tree** structure. MySQL can build a hash index on a prefix of any length of the key defined for the B-tree, depending on the pattern of searches against the index. A hash index can be partial; the whole B-tree index does not need to be cached in the buffer pool.

In MySQL 5.6 and higher, another way to take advantage of fast single-value lookups with InnoDB tables is to use the **memcached** interface to InnoDB. See Section 14.2.16, “`InnoDB` Integration with memcached” for details.
See Also B-tree, buffer pool, hash index, memcached, page, secondary index.

AHI

Acronym for **adaptive hash index**.
See Also adaptive hash index.

AIO

Acronym for **asynchronous I/O**. You might see this acronym in InnoDB messages or keywords.
See Also asynchronous I/O.

Antelope

The code name for the original InnoDB **file format**. It supports the **redundant** and **compact** row formats, but not the newer **dynamic** and **compressed** row formats available in the **Barracuda** file format.

If your application could benefit from InnoDB table **compression**, or uses BLOBs or large text columns that could benefit from the dynamic row format, you might switch some tables to Barracuda format. You select the file format to use by setting the `innodb_file_format` option before creating the table.
See Also Barracuda, compact row format, compressed row format, dynamic row format, file format, innodb_file_format, redundant row format.

application programming interface (API)

A set of functions or procedures. An API provides a stable set of names and types for functions, procedures, parameters, and return values.

apply

When a backup produced by the **MySQL Enterprise Backup** product does not include the most recent changes that occurred while the backup was underway, the process of updating the backup files to include those changes is known as the **apply** step. It is specified by the `apply-log` option of the `mysqlbackup` command.

Before the changes are applied, we refer to the files as a **raw backup**. After the changes are applied, we refer to the files as a **prepared backup**. The changes are recorded in the **ibbackup_logfile** file; once the apply step is finished, this file is no longer necessary.
See Also hot backup, ibbackup_logfile, MySQL Enterprise Backup, prepared backup, raw backup.

asynchronous I/O

A type of I/O operation that allows other processing to proceed before the I/O is completed. Also known as **non-blocking I/O** and abbreviated as **AIO**. InnoDB uses this type of I/O for certain operations that can run in parallel without affecting the reliability of the database, such as reading pages into the **buffer pool** that have not actually been requested, but might be needed soon.

Historically, InnoDB has used asynchronous I/O on Windows systems only. Starting with the InnoDB Plugin 1.1, InnoDB uses asynchronous I/O on Linux systems. This change introduces a dependency on `libaio`. On other Unix-like systems, InnoDB uses synchronous I/O only.
See Also buffer pool, non-blocking I/O.

atomic

In the SQL context, **transactions** are units of work that either succeed entirely (when **committed**) or have no effect at all (when **rolled back**). The indivisible ("atomic") property of transactions is the "A" in the acronym **ACID**.

See Also ACID, commit, rollback, transaction.

atomic instruction

Special instructions provided by the CPU, to ensure that critical low-level operations cannot be interrupted.

auto-increment

A property of a table column (specified by the `AUTO_INCREMENT` keyword) that automatically adds an ascending sequence of values in the column. InnoDB supports auto-increment only for **primary key** columns.

It saves work for the developer, not to have to produce new unique values when inserting new rows. It provides useful information for the query optimizer, because the column is known to be not null and with unique values. The values from such a column can be used as lookup keys in various contexts, and because they are auto-generated there is no reason to ever change them; for this reason, primary key columns are often specified as auto-incrementing.

Auto-increment columns can be problematic with statement-based replication, because replaying the statements on a slave might not produce the same set of column values as on the master, due to timing issues. When you have an auto-incrementing primary key, you can use statement-based replication only with the setting `innodb_autoinc_lock_mode=1`. If you have `innodb_autoinc_lock_mode=2`, which allows higher concurrency for insert operations, use **row-based replication** rather than **statement-based replication**. The setting `innodb_autoinc_lock_mode=0` is the previous (traditional) default setting and should not be used except for compatibility purposes.
See Also auto-increment locking, innodb_autoinc_lock_mode, primary key, row-based replication, statement-based replication.

auto-increment locking

The convenience of an **auto-increment** primary key involves some tradeoff with concurrency. In the simplest case, if one transaction is inserting values into the table, any other transactions must wait to do their own inserts into that table, so that rows inserted by the first transaction receive consecutive primary key values. InnoDB includes optimizations, and the `innodb_autoinc_lock_mode` option, so that you can choose how to trade off between predictable sequences of auto-increment values and maximum **concurrency** for insert operations.
See Also auto-increment, concurrency, innodb_autoinc_lock_mode.

autocommit

A setting that causes a **commit** operation after each **SQL** statement. This mode is not recommended for working with InnoDB tables with **transactions** that span several statements. It can help performance for **read-only transactions** on InnoDB tables, where it minimizes overhead from **locking** and generation of **undo** data, especially in MySQL 5.6.4 and up. It is also appropriate for working with MyISAM tables, where transactions are not applicable.
See Also commit, locking, read-only transaction, SQL, transaction, undo.

availability

The ability to cope with, and if necessary recover from, failures on the host, including failures of MySQL, the operating system, or the hardware and maintenance activity that may otherwise cause downtime. Often paired with **scalability** as critical aspects of a large-scale deployment.
See Also scalability.

B

B-tree

A tree data structure that is popular for use in database indexes. The structure is kept sorted at all times, enabling fast lookup for exact matches (equals operator) and ranges (for example, greater than, less than, and `BETWEEN` operators). This type of index is available for most storage engines, such as InnoDB and MyISAM.

Because B-tree nodes can have many children, a B-tree is not the same as a binary tree, which is limited to 2 children per node.

Contrast with **hash index**, which is only available in the MEMORY storage engine. The MEMORY storage engine can also use B-tree indexes, and you should choose B-tree indexes for MEMORY tables if some queries use range operators.
See Also hash index.

backticks

Identifiers within MySQL SQL statements must be quoted using the backtick character (`` ` ``) if they contain special characters or reserved words. For example, to refer to a table named `FOO#BAR` or a column named `SELECT`, you would specify the identifiers as `` `FOO#BAR` `` and `` `SELECT` ``. Since the backticks provide an extra level of safety, they are used extensively in program-generated SQL statements, where the identifier names might not be known in advance.

Many other database systems use double quotation marks (`"`) around such special names. For portability, you can enable `ANSI_QUOTES` mode in MySQL and use double quotation marks instead of backticks to qualify identifier names.
See Also SQL.

backup

The process of copying some or all table data and metadata from a MySQL instance, for safekeeping. Can also refer to the set of copied files. This is a crucial task for DBAs. The reverse of this process is the **restore** operation.

With MySQL, **physical backups** are performed by the **MySQL Enterprise Backup** product, and **logical backups** are performed by the `mysqldump` command. These techniques have different characteristics in terms of size and representation of the backup data, and speed (especially speed of the restore operation).

Backups are further classified as **hot**, **warm**, or **cold** depending on how much they interfere with normal database operation. (Hot backups have the least interference, cold backups the most.)
See Also cold backup, hot backup, logical backup, MySQL Enterprise Backup, mysqldump, physical backup, warm backup.

Barracuda

The code name for an InnoDB **file format** that supports compression for table data. This file format was first introduced in the InnoDB Plugin. It supports the **compressed** row format that enables InnoDB table compression, and the **dynamic** row format that improves the storage layout for BLOB and large text columns. You can select it through the `innodb_file_format` option.

Because the InnoDB **system tablespace** is stored in the original **Antelope** file format, to use the Barracuda file format you must also enable the **file-per-table** setting, which puts newly created tables in their own tablespaces separate from the system tablespace.

The **MySQL Enterprise Backup** product version 3.5 and above supports backing up tablespaces that use the Barracuda file format.
See Also Antelope, compact row format, compressed row format, dynamic row format, file format, file-per-table, innodb_file_format, MySQL Enterprise Backup, row format, system tablespace.

beta

An early stage in the life of a software product, when it is available only for evaluation, typically without a definite release number or a number less than 1. InnoDB does not use the beta designation, preferring an **early adopter** phase that can extend over several point releases, leading to a **GA** release.
See Also early adopter, GA.

binary log

A file containing a record of all statements that attempt to change table data. These statements can be replayed to bring slave servers up to date in a **replication** scenario, or to bring a database up to date after restoring table data from a backup. The binary logging feature can be turned on and off, although Oracle recommends always enabling it if you use replication or perform backups.

You can examine the contents of the binary log, or replay those statements during replication or recovery, by using the `mysqlbinlog` command. For full information about the binary log, see Section 5.2.4, “The Binary Log”. For MySQL configuration options related to the binary log, see Section 16.1.4.4, “Binary Log Options and Variables”.

For the **MySQL Enterprise Backup** product, the file name of the binary log and the current position within the file are important details. To record this information for the master server when taking a backup in a replication context, you can specify the `--slave-info` option.

Prior to MySQL 5.0, a similar capability was available, known as the update log. In MySQL 5.0 and higher, the binary log replaces the update log.
See Also binlog, MySQL Enterprise Backup, replication.

binlog
An informal name for the **binary log** file. For example, you might see this abbreviation used in e-mail messages or forum discussions.
See Also binary log.

blind query expansion
A special mode of **full-text search** enabled by the `WITH QUERY EXPANSION` clause. It performs the search twice, where the search phrase for the second search is the original search phrase concatenated with the few most highly relevant documents from the first search. This technique is mainly applicable for short search phrases, perhaps only a single word. It can uncover relevant matches where the precise search term does not occur in the document.
See Also full-text search.

bottleneck
A portion of a system that is constrained in size or capacity, that has the effect of limiting overall throughput. For example, a memory area might be smaller than necessary; access to a single required resource might prevent multiple CPU cores from running simultaneously; or waiting for disk I/O to complete might prevent the CPU from running at full capacity. Removing bottlenecks tends to improve **concurrency**. For example, the ability to have multiple InnoDB **buffer pool** instances reduces contention when multiple sessions read from and write to the buffer pool simultaneously.
See Also buffer pool, concurrency.

bounce
A **shutdown** operation immediately followed by a restart. Ideally with a relatively short **warmup** period so that performance and throughput quickly return to a high level.
See Also shutdown.

buddy allocator
A mechanism for managing different-sized **pages** in the InnoDB **buffer pool**.
See Also buffer pool, page, page size.

buffer
A memory or disk area used for temporary storage. Data is buffered in memory so that it can be written to disk efficiently, with a few large I/O operations rather than many small ones. Data is buffered on disk for greater reliability, so that it can be recovered even when a **crash** or other failure occurs at the worst possible time. The main types of buffers used by InnoDB are the **buffer pool**, the **doublewrite buffer**, and the **insert buffer**.
See Also buffer pool, crash, doublewrite buffer, insert buffer.

buffer pool
The memory area that holds cached InnoDB data for both tables and indexes. For efficiency of high-volume read operations, the buffer pool is divided into **pages** that can potentially hold multiple rows. For efficiency of cache management, the buffer pool is implemented as a linked list of pages; data that is rarely used is aged out of the cache, using a variation of the **LRU** algorithm. On systems with large memory, you can improve concurrency by dividing the buffer pool into multiple **buffer pool instances**.

Several InnoDB status variables, information_schema tables, and performance_schema tables help to monitor the internal workings of the buffer pool. Starting in MySQL 5.6, you can also dump and restore the contents of the buffer pool, either automatically during shutdown and restart, or manually at any time, through a set of InnoDB configuration variables such as innodb_buffer_pool_dump_at_shutdown and innodb_buffer_pool_load_at_startup.
See Also buffer pool instance, LRU, page, warm up.

buffer pool instance
Any of the multiple regions into which the **buffer pool** can be divided, controlled by the innodb_buffer_pool_instances configuration option. The total memory size specified by the innodb_buffer_pool_size is divided among all the instances. Typically, multiple buffer pool instances are appropriate for systems devoting multiple gigabytes to the InnoDB buffer pool, with each instance 1 gigabyte or larger. On systems loading or looking up large amounts of data in the buffer pool from many concurrent sessions, having multiple instances reduces the contention for exclusive access to the data structures that manage the buffer pool.
See Also buffer pool.

built-in
The built-in InnoDB storage engine within MySQL is the original form of distribution for the storage engine. Contrast with the **InnoDB Plugin**. Starting with MySQL 5.5, the InnoDB Plugin is merged back into the MySQL code base as the built-in InnoDB storage engine (known as InnoDB 1.1).

This distinction is important mainly in MySQL 5.1, where a feature or bug fix might apply to the InnoDB Plugin but not the built-in InnoDB, or vice versa.
See Also InnoDB, plugin.

business rules
The relationships and sequences of actions that form the basis of business software, used to run a commercial company. Sometimes these rules are dictated by law, other times by company policy. Careful planning ensures that the relationships encoded and enforced by the database, and the actions performed through application logic, accurately reflect the real policies of the company and can handle real-life situations.

For example, an employee leaving a company might trigger a sequence of actions from the human resources department. The human resources database might also need the flexibility to represent data about a person who has been hired, but not yet started work. Closing an account at an online service might result in data being removed from a database, or the data might be moved or flagged so that it could be recovered if the account is re-opened. A company might establish policies regarding salary maximums, minimums, and adjustments, in addition to basic sanity checks such as the salary not being a negative number. A retail database might not allow a purchase with the same serial number to be returned more than once, or might not allow credit card purchases above a certain value, while a database used to detect fraud might allow these kinds of things.
See Also relational.

C

.cfg file
A metadata file used with the InnoDB **transportable tablespace** feature. It is produced by the command FLUSH TABLES ... FOR EXPORT, puts one or more tables in a consistent state that can be copied to another server. The .cfg file is copied along with the corresponding **.ibd file**, and used to adjust the internal values of the .ibd file, such as the **space ID**, during the ALTER TABLE ... IMPORT TABLESPACE step.
See Also .ibd file, space ID, transportable tablespace.

cache
The general term for any memory area that stores copies of data for frequent or high-speed retrieval. In InnoDB, the primary kind of cache structure is the **buffer pool**.

See Also buffer, buffer pool.

cardinality

The number of different values in a table **column**. When queries refer to columns that have an associated **index**, the cardinality of each column influences which access method is most efficient. For example, for a column with a **unique constraint**, the number of different values is equal to the number of rows in the table. If a table has a million rows but only 10 different values for a particular column, each value occurs (on average) 100,000 times. A query such as `SELECT c1 FROM t1 WHERE c1 = 50;` thus might return 1 row or a huge number of rows, and the database server might process the query differently depending on the cardinality of `c1`.

If the values in a column have a very uneven distribution, the cardinality might not be a good way to determine the best query plan. For example, `SELECT c1 FROM t1 WHERE c1 = x;` might return 1 row when `x=50` and a million rows when `x=30`. In such a case, you might need to use **index hints** to pass along advice about which lookup method is more efficient for a particular query.

Cardinality can also apply to the number of distinct values present in multiple columns, as in a **composite index**.

For InnoDB, the process of estimating cardinality for indexes is influenced by the `innodb_stats_sample_pages` and the `innodb_stats_on_metadata` configuration options. The estimated values are more stable when the **persistent statistics** feature is enabled (in MySQL 5.6 and higher).
See Also column, composite index, index, index hint, persistent statistics, random dive, selectivity, unique constraint.

change buffer

A special data structure that records changes to **pages** in **secondary indexes**. These values could result from SQL `INSERT`, `UPDATE`, or `DELETE` statements (**DML**). The set of features involving the change buffer is known collectively as **change buffering**, consisting of **insert buffering**, **delete buffering**, and **purge buffering**.

Changes are only recorded in the change buffer when the relevant page from the secondary index is not in the **buffer pool**. When the relevant index page is brought into the buffer pool while associated changes are still in the change buffer, the changes for that page are applied in the buffer pool (**merged**) using the data from the change buffer. Periodically, the **purge** operation that runs during times when the system is mostly idle, or during a slow shutdown, writes the new index pages to disk. The purge operation can write the disk blocks for a series of index values more efficiently than if each value were written to disk immediately.

Physically, the change buffer is part of the **system tablespace**, so that the index changes remain buffered across database restarts. The changes are only applied (**merged**) when the pages are brought into the buffer pool due to some other read operation.

The kinds and amount of data stored in the change buffer are governed by the `innodb_change_buffering` and `innodb_change_buffer_max_size` configuration options. To see information about the current data in the change buffer, issue the `SHOW ENGINE INNODB STATUS` command.

Formerly known as the **insert buffer**.
See Also buffer pool, change buffering, delete buffering, DML, insert buffer, insert buffering, merge, page, purge, purge buffering, secondary index, system tablespace.

change buffering

The general term for the features involving the **change buffer**, consisting of **insert buffering**, **delete buffering**, and **purge buffering**. Index changes resulting from SQL statements, which could normally involve random I/O operations, are held back and performed periodically by a background **thread**. This sequence of operations can write the disk blocks for a series of index values more efficiently than if each value were written to disk immediately. Controlled by the `innodb_change_buffering` and `innodb_change_buffer_max_size` configuration options.
See Also change buffer, delete buffering, insert buffering, purge buffering.

checkpoint

As changes are made to data pages that are cached in the **buffer pool**, those changes are written to the **data files** sometime later, a process known as **flushing**. The checkpoint is a record of the latest changes (represented by an **LSN** value) that have been successfully written to the data files.
See Also buffer pool, data files, flush, fuzzy checkpointing, LSN.

checksum

In `InnoDB`, a validation mechanism to detect corruption when a **page** in a **tablespace** is read from disk into the InnoDB **buffer pool**. This feature is turned on and off by the `innodb_checksums` configuration option. In MySQL 5.6, you can enable a faster checksum algorithm by also specifying the configuration option `innodb_checksum_algorithm=crc32`.

The `innochecksum` command helps to diagnose corruption problems by testing the checksum values for a specified **tablespace** file while the MySQL server is shut down.

MySQL also uses checksums for replication purposes. For details, see the configuration options `binlog_checksum`, `master_verify_checksum`, and `slave_sql_verify_checksum`.
See Also buffer pool, page, tablespace.

child table

In a **foreign key** relationship, a child table is one whose rows refer (or point) to rows in another table with an identical value for a specific column. This is the table that contains the `FOREIGN KEY ... REFERENCES` clause and optionally `ON UPDATE` and `ON DELETE` clauses. The corresponding row in the **parent table** must exist before the row can be created in the child table. The values in the child table can prevent delete or update operations on the parent table, or can cause automatic deletion or updates in the child table, based on the `ON CASCADE` option used when creating the foreign key.
See Also foreign key, parent table.

clean page

A **page** in the InnoDB **buffer pool** where all changes made in memory have also been written (**flushed**) to the data files. The opposite of a **dirty page**.
See Also buffer pool, data files, dirty page, flush, page.

clean shutdown

A **shutdown** that completes without errors and applies all changes to InnoDB tables before finishing, as opposed to a **crash** or a **fast shutdown**. Synonym for **slow shutdown**.
See Also crash, fast shutdown, shutdown, slow shutdown.

client

A type of program that sends requests to a server, and interprets or processes the results. The client software might run only some of the time (such as a mail or chat program), and might run interactively (such as the `mysql` command processor).
See Also mysql, server.

clustered index

The InnoDB term for a **primary key** index. InnoDB table storage is organized based on the values of the primary key columns, to speed up queries and sorts involving the primary key columns. For best performance, choose the primary key columns carefully based on the most performance-critical queries. Because modifying the columns of the clustered index is an expensive operation, choose primary columns that are rarely or never updated.

In the Oracle Database product, this type of table is known as an **index-organized table**.
See Also index, primary key, secondary index.

cold backup

A **backup** taken while the database is shut down. For busy applications and web sites, this might not be practical, and you might prefer a **warm backup** or a **hot backup**.

See Also backup, hot backup, warm backup.

column

A data item within a **row**, whose storage and semantics are defined by a data type. Each **table** and **index** is largely defined by the set of columns it contains.

Each column has a **cardinality** value. A column can be the **primary key** for its table, or part of the primary key. A column can be subject to a **unique constraint**, a **NOT NULL constraint**, or both. Values in different columns, even across different tables, can be linked by a **foreign key** relationship.

In discussions of MySQL internal operations, sometimes **field** is used as a synonym.
See Also cardinality, foreign key, index, primary key, row, SQL, table, unique constraint.

column index

An **index** on a single column.
See Also composite index, index.

column prefix

When an index is created with a length specification, such as `CREATE INDEX idx ON t1 (c1(N))`, only the first N characters of the column value are stored in the index. Keeping the index prefix small makes the index compact, and the memory and disk I/O savings help performance. (Although making the index prefix too small can hinder query optimization by making rows with different values appear to the query optimizer to be duplicates.)

For columns containing binary values or long text strings, where sorting is not a major consideration and storing the entire value in the index would waste space, the index automatically uses the first N (typically 768) characters of the value to do lookups and sorts.
See Also index.

commit

A **SQL** statement that ends a **transaction**, making permanent any changes made by the transaction. It is the opposite of **rollback**, which undoes any changes made in the transaction.

InnoDB uses an **optimistic** mechanism for commits, so that changes can be written to the data files before the commit actually occurs. This technique makes the commit itself faster, with the tradeoff that more work is required in case of a rollback.

By default, MySQL uses the **autocommit** setting, which automatically issues a commit following each SQL statement.
See Also autocommit, optimistic, rollback, SQL, transaction.

compact row format

The default `InnoDB` **row format** since MySQL 5.0.3. Available for tables that use the **Antelope file format**. It has a more compact representation for nulls and variable-length fields than the prior default (**redundant row format**).

Because of the **B-tree** indexes that make row lookups so fast in InnoDB, there is little if any performance benefit to keeping all rows the same size.

For additional information about `InnoDB` `COMPACT` row format, see Section 14.2.9.4, “`COMPACT` and `REDUNDANT` Row Formats”.
See Also Antelope, file format, redundant row format, row format.

composite index

An **index** that includes multiple columns.
See Also index, index prefix.

compressed backup

The compression feature of the **MySQL Enterprise Backup** product makes a compressed copy of each tablespace, changing the extension from `.ibd` to `.ibz`. Compressing the backup data allows you to keep more backups on hand, and reduces the time to transfer backups to a different server. The data is uncompressed during the restore operation. When a compressed backup operation processes a table that is already compressed, it skips the compression step for that table, because compressing again would result in little or no space savings.

A set of files produced by the **MySQL Enterprise Backup** product, where each **tablespace** is compressed. The compressed files are renamed with a `.ibz` file extension.

Applying **compression** right at the start of the backup process helps to avoid storage overhead during the compression process, and to avoid network overhead when transferring the backup files to another server. The process of **applying** the **binary log** takes longer, and requires uncompressing the backup files.
See Also apply, binary log, compression, hot backup, MySQL Enterprise Backup, tablespace.

compressed row format

A **row format** that enables data and index **compression** for `InnoDB` tables. It was introduced in the `InnoDB` Plugin, available as part of the **Barracuda** file format. Large fields are stored away from the page that holds the rest of the row data, as in **dynamic row format**. Both index pages and the large fields are compressed, yielding memory and disk savings. Depending on the structure of the data, the decrease in memory and disk usage might or might not outweigh the performance overhead of uncompressing the data as it is used. See Section 14.2.7, “`InnoDB` Compressed Tables” for usage details.

For additional information about `InnoDB` `COMPRESSED` row format, see Section 14.2.9.3, “`DYNAMIC` and `COMPRESSED` Row Formats”.
See Also Barracuda, compression, dynamic row format, row format.

compression

A feature with wide-ranging benefits from using less disk space, performing less I/O, and using less memory for caching. InnoDB table and index data can be kept in a compressed format during database operation.

The data is uncompressed when needed for queries, and re-compressed when changes are made by **DML** operations. After you enable compression for a table, this processing is transparent to users and application developers. DBAs can consult **information_schema** tables to monitor how efficiently the compression parameters work for the MySQL instance and for particular compressed tables.

When InnoDB table data is compressed, the compression applies to the **table** itself, any associated **index** data, and the pages loaded into the **buffer pool**. Compression does not apply to pages in the **undo buffer**.

The table compression feature requires using MySQL 5.5 or higher, or the InnoDB Plugin in MySQL 5.1 or earlier, and creating the table using the **Barracuda** file format and **compressed row format**, with the **innodb_file_per_table** setting turned on. The compression for each table is influenced by the `KEY_BLOCK_SIZE` clause of the `CREATE TABLE` and `ALTER TABLE` statements. In MySQL 5.6 and higher, compression is also affected by the server-wide configuration options `innodb_compression_failure_threshold_pct`, `innodb_compression_level`, and `innodb_compression_pad_pct_max`. See Section 14.2.7, “`InnoDB` Compressed Tables” for usage details.

Another type of compression is the **compressed backup** feature of the **MySQL Enterprise Backup** product.
See Also Barracuda, buffer pool, compressed row format, DML, hot backup, index, INFORMATION_SCHEMA, innodb_file_per_table, plugin, table, undo buffer.

compression failure

Not actually an error, rather an expensive operation that can occur when using **compression** in combination with **DML** operations. It occurs when: updates to a compressed **page** overflow the area on the page reserved for recording modifications; the page is compressed again, with all changes applied to the table data; the re-compressed data does not fit on the original page, requiring MySQL to split the data into

two new pages and compress each one separately. To check the frequency of this condition, query the table INFORMATION_SCHEMA.INNODB_CMP and check how much the value of the COMPRESS_OPS column exceeds the value of the COMPRESS_OPS_OK column. Ideally, compression failures do not occur often; when they do, you can adjust the configuration options innodb_compression_level, innodb_compression_failure_threshold_pct, and innodb_compression_pad_pct_max.
See Also compression, DML, page.

concatenated index
See composite index.

concurrency
The ability of multiple operations (in database terminology, **transactions**) to run simultaneously, without interfering with each other. Concurrency is also involved with performance, because ideally the protection for multiple simultaneous transactions works with a minimum of performance overhead, using efficient mechanisms for **locking**.
See Also ACID, locking, transaction.

configuration file
The file that holds the **option** values used by MySQL at startup. Traditionally, on Linux and UNIX this file is named my.cnf, and on Windows it is named my.ini. You can set a number of options related to InnoDB under the [mysqld] section of the file.

Typically, this file is searched for in the locations /etc/my.cnf /etc/mysql/my.cnf /usr/local/mysql/etc/my.cnf and ~/.my.cnf. See Section 4.2.3.3, “Using Option Files” for details about the search path for this file.

When you use the **MySQL Enterprise Backup** product, you typically use two configuration files: one that specifies where the data comes from and how it is structured (which could be the original configuration file for your real server), and a stripped-down one containing only a small set of options that specify where the backup data goes and how it is structured. The configuration files used with the **MySQL Enterprise Backup** product must contain certain options that are typically left out of regular configuration files, so you might need to add some options to your existing configuration file for use with **MySQL Enterprise Backup**.
See Also my.cnf, option file.

consistent read
A read operation that uses snapshot information to present query results based on a point in time, regardless of changes performed by other transactions running at the same time. If queried data has been changed by another transaction, the original data is reconstructed based on the contents of the **undo log**. This technique avoids some of the **locking** issues that can reduce **concurrency** by forcing transactions to wait for other transactions to finish.

With the **repeatable read** isolation level, the snapshot is based on the time when the first read operation is performed. With the **read committed** isolation level, the snapshot is reset to the time of each consistent read operation.

Consistent read is the default mode in which InnoDB processes SELECT statements in **READ COMMITTED** and **REPEATABLE READ** isolation levels. Because a consistent read does not set any locks on the tables it accesses, other sessions are free to modify those tables while a consistent read is being performed on the table.

For technical details about the applicable isolation levels, see Section 14.2.2.4, “Consistent Nonlocking Reads”.
See Also ACID, concurrency, isolation level, locking, MVCC, READ COMMITTED, READ UNCOMMITTED, REPEATABLE READ, SERIALIZABLE, transaction, undo log.

constraint
An automatic test that can block database changes to prevent data from becoming inconsistent. (In computer science terms, a kind of assertion related to an invariant condition.) Constraints are a crucial component of the **ACID** philosophy, to maintain data consistency. Constraints supported by MySQL include **FOREIGN KEY constraints** and **unique constraints**.

See Also ACID, foreign key, relational, unique constraint.

counter

A value that is incremented by a particular kind of `InnoDB` operation. Useful for measuring how busy a server is, troubleshooting the sources of performance issues, and testing whether changes (for example, to configuration settings or indexes used by queries) have the desired low-level effects. Different kinds of counters are available through **performance_schema** tables and **information_schema** tables, particularly `information_schema.innodb_metrics`.
See Also INFORMATION_SCHEMA, metrics counter, Performance Schema.

covering index

An **index** that includes all the columns retrieved by a query. Instead of using the index values as pointers to find the full table rows, the query returns values from the index structure, saving disk I/O. InnoDB can apply this optimization technique to more indexes than MyISAM can, because InnoDB **secondary indexes** also include the primary key columns. InnoDB cannot apply this technique for queries against tables modified by a transaction, until that transaction ends.

Any **column index** or **composite index** could act as a covering index, given the right query. Design your indexes and queries to take advantage of this optimization technique wherever possible.
See Also column index, composite index, index, secondary index.

CPU-bound

A type of **workload** where the primary **bottleneck** is CPU operations in memory. Typically involves read-intensive operations where the results can all be cached in the **buffer pool**.
See Also bottleneck, buffer pool, CPU-bound, workload.

crash

MySQL uses the term "crash" to refer generally to any unexpected shutdown operation where the server cannot do its normal cleanup. For example, a crash could happen due to a hardware fault on the database server machine or storage device; a power failure; a potential data mismatch that causes the MySQL server to halt; a **fast shutdown** initiated by the DBA; or many other reasons. The robust, automatic **crash recovery** for **InnoDB** tables ensures that data is made consistent when the server is restarted, without any extra work for the DBA.
See Also crash recovery, fast shutdown, InnoDB, redo log, shutdown.

crash recovery

The cleanup activities that occur when MySQL is started again after a **crash**. For **InnoDB** tables, changes from incomplete transactions are replayed using data from the **redo log**. Changes that were **committed** before the crash, but not yet written into the data files, are reconstructed from the **doublewrite buffer**. When the database is shut down normally, this type of activity is performed during shutdown by the **purge** operation.

During normal operation, committed data can be stored in the **change buffer** for a period of time before being written to the data files. There is always a tradeoff between keeping the data files up-to-date, which introduces performance overhead during normal operation, and buffering the data, which can make shutdown and crash recovery take longer.
See Also change buffer, commit, crash, data files, doublewrite buffer, InnoDB, purge, redo log.

CRUD

Acronym for "create, read, update, delete", a common sequence of operations in database applications. Often denotes a class of applications with relatively simple database usage (basic **DDL**, **DML** and **query** statements in **SQL**) that can be implemented quickly in any language.
See Also DDL, DML, query, SQL.

cursor

An internal data structure that is used to represent the result set of a **query**, or other operation that performs a search using an SQL `WHERE` clause. It works like an iterator in other high-level languages, producing each value from the result set as requested.

Although usually SQL handles the processing of cursors for you, you might delve into the inner workings when dealing with performance-critical code.
See Also query.

D

data definition language
See DDL.

data dictionary
Metadata that keeps track of InnoDB-related objects such as **tables**, **indexes**, and table **columns**. This metadata is physically located in the InnoDB **system tablespace**. For historical reasons, it overlaps to some degree with information stored in the **.frm files**.

Because the **MySQL Enterprise Backup** product always backs up the system tablespace, all backups include the contents of the data dictionary.
See Also column, .frm file, hot backup, index, MySQL Enterprise Backup, system tablespace, table.

data directory
The directory under which each MySQL **instance** keeps the **data files** for InnoDB and the directories representing individual databases. Controlled by the `datadir` configuration option.
See Also data files, instance.

data files
The files that physically contain the InnoDB **table** and **index** data. There can be a one-to-many relationship between data files and tables, as in the case of the **system tablespace**, which can hold multiple InnoDB tables as well as the **data dictionary**. There can also be a one-to-one relationship between data files and tables, as when the **file-per-table** setting is enabled, causing each newly created table to be stored in a separate **tablespace**.
See Also data dictionary, file-per-table, index, system tablespace, table, tablespace.

data manipulation language
See DML.

data warehouse
A database system or application that primarily runs large **queries**. The read-only or read-mostly data might be organized in **denormalized** form for query efficiency. Can benefit from the optimizations for **read-only transactions** in MySQL 5.6 and higher. Contrast with **OLTP**.
See Also denormalized, OLTP, query, read-only transaction.

database
Within the MySQL **data directory**, each database is represented by a separate directory. The InnoDB **system tablespace**, which can hold table data from multiple databases within a MySQL **instance**, is kept in its **data files** that reside outside the individual database directories. When **file-per-table** mode is enabled, the **.ibd files** representing individual InnoDB tables are stored inside the database directories.

For long-time MySQL users, a database is a familiar notion. Users coming from an Oracle Database background will find that the MySQL meaning of a database is closer to what Oracle Database calls a **schema**.
See Also data files, file-per-table, .ibd file, instance, schema, system tablespace.

DCL
Data control language, a set of **SQL** statements for managing privileges. In MySQL, consists of the `GRANT` and `REVOKE` statements. Contrast with **DDL** and **DML**.
See Also DDL, DML, SQL.

DDL

Data definition language, a set of **SQL** statements for manipulating the database itself rather than individual table rows. Includes all forms of the `CREATE`, `ALTER`, and `DROP` statements. Also includes the `TRUNCATE` statement, because it works differently than a `DELETE FROM table_name` statement, even though the ultimate effect is similar.

DDL statements automatically **commit** the current **transaction**; they cannot be **rolled back**.

InnoDB's online DDL feature enhances performance for `CREATE INDEX`, `DROP INDEX`, and many types of `ALTER TABLE` operations. See Section 14.2.11, “InnoDB and Online DDL” for more information. Also, the InnoDB's file-per-table setting can affect the behaviour of `DROP TABLE` and `TRUNCATE TABLE` operations.

Contrast with **DML** and **DCL**.
See Also commit, DCL, DML, file-per-table, rollback, SQL, transaction.

deadlock

A situation where different **transactions** are unable to proceed, because each holds a **lock** that the other needs. Because both transactions are waiting for a resource to become available, neither will ever release the locks it holds.

A deadlock can occur when the transactions lock rows in multiple tables (through statements such as `UPDATE` or `SELECT ... FOR UPDATE`), but in the opposite order. A deadlock can also occur when such statements lock ranges of index records and **gaps**, with each transaction acquiring some locks but not others due to a timing issue.

To reduce the possibility of deadlocks, use transactions rather than `LOCK TABLE` statements; keep transactions that insert or update data small enough that they do not stay open for long periods of time; when different transactions update multiple tables or large ranges of rows, use the same order of operations (such as `SELECT ... FOR UPDATE`) in each transaction; create indexes on the columns used in `SELECT ... FOR UPDATE` and `UPDATE ... WHERE` statements. The possibility of deadlocks is not affected by the **isolation level**, because the isolation level changes the behavior of read operations, while deadlocks occur because of write operations.

If a deadlock does occur, InnoDB detects the condition and **rolls back** one of the transactions (the **victim**). Thus, even if your application logic is perfectly correct, you must still handle the case where a transaction must be retried. To see the last deadlock in an InnoDB user transaction, use the command `SHOW ENGINE INNODB STATUS`. If frequent deadlocks highlight a problem with transaction structure or application error handling, run with the `innodb_print_all_deadlocks` setting enabled to print information about all deadlocks to the `mysqld` error log.

For background information on how deadlocks are automatically detected and handled, see Section 14.2.2.10, “Deadlock Detection and Rollback”. For tips on avoiding and recovering from deadlock conditions, see Section 14.2.2.11, “How to Cope with Deadlocks”.
See Also concurrency, gap, isolation level, lock, locking, rollback, transaction, victim.

deadlock detection

A mechanism that automatically detects when a **deadlock** occurs, and automatically **rolls back** one of the **transactions** involved (the **victim**).
See Also deadlock, rollback, transaction, victim.

delete

When InnoDB processes a `DELETE` statement, the rows are immediately marked for deletion and no longer are returned by queries. The storage is reclaimed sometime later, during the periodic garbage collection known as the **purge** operation, performed by a separate thread. For removing large quantities of data, related operations with their own performance characteristics are **truncate** and **drop**.
See Also drop, purge, truncate.

delete buffering

The technique of storing index changes due to DELETE operations in the **insert buffer** rather than writing them immediately, so that the physical writes can be performed to minimize random I/O. (Because delete operations are a two-step process, this operation buffers the write that normally marks an index record for deletion.) It is one of the types of **change buffering**; the others are **insert buffering** and **purge buffering**.
See Also change buffer, change buffering, insert buffer, insert buffering, purge buffering.

denormalized

A data storage strategy that duplicates data across different tables, rather than linking the tables with **foreign keys** and **join** queries. Typically used in **data warehouse** applications, where the data is not updated after loading. In such applications, query performance is more important than making it simple to maintain consistent data during updates. Contrast with **normalized**.
See Also data warehouse, normalized.

descending index

A type of index available with some database systems, where index storage is optimized to process ORDER BY *column* DESC clauses. Currently, although MySQL allows the DESC keyword in the CREATE TABLE statement, it does not use any special storage layout for the resulting index.
See Also index.

dirty page

A **page** in the InnoDB **buffer pool** that has been updated in memory, where the changes are not yet written (**flushed**) to the data files. The opposite of a **clean page**.
See Also buffer pool, clean page, data files, flush, page.

dirty read

An operation that retrieves unreliable data, data that was updated by another transaction but not yet **committed**. It is only possible with the **isolation level** known as **read uncommitted**.

This kind of operation does not adhere to the **ACID** principle of database design. It is considered very risky, because the data could be **rolled back**, or updated further before being committed; then, the transaction doing the dirty read would be using data that was never confirmed as accurate.

Its polar opposite is **consistent read**, where InnoDB goes to great lengths to ensure that a transaction does not read information updated by another transaction, even if the other transaction commits in the meantime.
See Also ACID, commit, consistent read, isolation level, READ COMMITTED, READ UNCOMMITTED, rollback.

disk-based

A kind of database that primarily organizes data on disk storage (hard drives or equivalent). Data is brought back and forth between disk and memory to be operated upon. It is the opposite of an **in-memory database**. Although InnoDB is disk-based, it also contains features such as **the buffer** pool, multiple buffer pool instances, and the **adaptive hash index** that allow certain kinds of workloads to work primarily from memory.
See Also adaptive hash index, buffer pool, in-memory database.

disk-bound

A type of **workload** where the primary **bottleneck** is disk I/O. (Also known as **I/O-bound**.) Typically involves frequent writes to disk, or random reads of more data than can fit into the **buffer pool**.
See Also bottleneck, buffer pool, CPU-bound, workload.

DML

Data manipulation language, a set of **SQL** statements for performing insert, update, and delete operations. The SELECT statement is sometimes considered as a DML statement, because the SELECT ... FOR UPDATE form is subject to the same considerations for **locking** as INSERT, UPDATE, and DELETE.

DML statements for an InnoDB table operate in the context of a **transaction**, so their effects can be **committed** or **rolled back** as a single unit.

Contrast with **DDL** and **DCL**.
See Also commit, DCL, DDL, locking, rollback, SQL, transaction.

document id

In the InnoDB **full-text search** feature, a special column in the table containing the **FULLTEXT index**, to uniquely identify the document associated with each **ilist** value. Its name is `FTS_DOC_ID` (uppercase required). The column itself must be of `BIGINT UNSIGNED NOT NULL` type, with a unique index named `FTS_DOC_ID_INDEX`. Preferably, you define this column when creating the table. If InnoDB must add the column to the table while creating a `FULLTEXT` index, the indexing operation is considerably more expensive.
See Also full-text search, FULLTEXT index, ilist.

doublewrite buffer

InnoDB uses a novel file flush technique called doublewrite. Before writing **pages** to the **data files**, InnoDB first writes them to a contiguous area called the doublewrite buffer. Only after the write and the flush to the doublewrite buffer have completed, does InnoDB write the pages to their proper positions in the data file. If the operating system crashes in the middle of a page write, InnoDB can later find a good copy of the page from the doublewrite buffer during **crash recovery**.

Although data is always written twice, the doublewrite buffer does not require twice as much I/O overhead or twice as many I/O operations. Data is written to the buffer itself as a large sequential chunk, with a single `fsync()` call to the operating system.

To turn off the doublewrite buffer, specify the option `innodb_doublewrite=0`.
See Also crash recovery, data files, page, purge.

drop

A kind of **DDL** operation that removes a schema object, through a statement such as `DROP TABLE` or `DROP INDEX`. It maps internally to an `ALTER TABLE` statement. From an InnoDB perspective, the performance considerations of such operations involve the time that the **data dictionary** is locked to ensure that interrelated objects are all updated, and the time to update memory structures such as the **buffer pool**. For a **table**, the drop operation has somewhat different characteristics than a **truncate** operation (`TRUNCATE TABLE` statement).
See Also buffer pool, data dictionary, DDL, table, truncate.

dynamic row format

A row format introduced in the `InnoDB` Plugin, available as part of the **Barracuda file format**. Because `TEXT` and `BLOB` fields are stored outside of the rest of the page that holds the row data, it is very efficient for rows that include large objects. Since the large fields are typically not accessed to evaluate query conditions, they are not brought into the **buffer pool** as often, resulting in fewer I/O operations and better utilization of cache memory.

For additional information about `InnoDB` `DYNAMIC` row format, see Section 14.2.9.3, “`DYNAMIC` and `COMPRESSED` Row Formats”.
See Also Barracuda, buffer pool, file format, row format.

E

early adopter

A stage similar to beta, when a software product is typically evaluated for performance, functionality, and compatibility in a non-mission-critical setting. InnoDB uses the **early adopter** designation rather than **beta**, through a succession of point releases leading up to a **GA** release.
See Also beta, GA.

error log

A type of **log** showing information about MySQL startup and critical runtime errors and **crash** information. For details, see Section 5.2.2, “The Error Log”.
See Also crash, log.

eviction

The process of removing an item from a cache or other temporary storage area, such as the InnoDB **buffer pool**. Often, but not always, uses the **LRU** algorithm to determine which item to remove. When a **dirty page** is evicted, its contents are **flushed** to disk, and any **dirty neighbor** pages might be flushed also.
See Also buffer pool, dirty page, flush, LRU.

exclusive lock

A kind of **lock** that prevents any other **transaction** from locking the same row. Depending on the transaction **isolation level**, this kind of lock might block other transactions from writing to the same row, or might also block other transactions from reading the same row. The default InnoDB isolation level, **REPEATABLE READ**, enables higher **concurrency** by allowing transactions to read rows that have exclusive locks, a technique known as **consistent read**.
See Also concurrency, consistent read, isolation level, lock, REPEATABLE READ, shared lock, transaction.

extent

A group of **pages** within a **tablespace** totaling 1 megabyte. With the default **page size** of 16KB, an extent contains 64 pages. In MySQL 5.6, the page size can also be 4KB or 8KB, in which case an extent contains more pages, still adding up to 1MB.

InnoDB features such as **segments**, **read-ahead** requests and the **doublewrite buffer** use I/O operations that read, write, allocate, or free data one extent at a time.
See Also doublewrite buffer, neighbor page, page, page size, read-ahead, segment, tablespace.

F

.frm file

A file containing the metadata, such as the table definition, of a MySQL table.

For backups, you must always keep the full set of `.frm` files along with the backup data to be able to restore tables that are altered or dropped after the backup.

Although each InnoDB table has a `.frm` file, InnoDB maintains its own table metadata in the system tablespace; the `.frm` files are not needed for InnoDB to operate on InnoDB tables.

These files are backed up by the **MySQL Enterprise Backup** product. These files must not be modified by an `ALTER TABLE` operation while the backup is taking place, which is why backups that include non-InnoDB tables perform a `FLUSH TABLES WITH READ LOCK` operation to freeze such activity while backing up the `.frm` files. Restoring a backup can result in `.frm` files being created, changed, or removed to match the state of the database at the time of the backup.
See Also MySQL Enterprise Backup.

Fast Index Creation

A capability first introduced in the InnoDB Plugin, now part of the MySQL server in 5.5 and higher, that speeds up creation of InnoDB **secondary indexes** by avoiding the need to completely rewrite the associated table. The speedup applies to dropping secondary indexes also.

Because index maintenance can add performance overhead to many data transfer operations, consider doing operations such as `ALTER TABLE ... ENGINE=INNODB` or `INSERT INTO ... SELECT * FROM ...` without any secondary indexes in place, and creating the indexes afterward.

In MySQL 5.6, this feature becomes more general: you can read and write to tables while an index is being created, and many more kinds of `ALTER TABLE` operations can be performed without copying the table, without blocking **DML** operations, or both. Thus in MySQL 5.6 and higher, we typically refer to this set of features as **online DDL** rather than Fast Index Creation.
See Also DML, index, online DDL, secondary index.

fast shutdown

The default **shutdown** procedure for InnoDB, based on the configuration setting `innodb_fast_shutdown=1`. To save time, certain **flush** operations are skipped. This type of shutdown is safe during normal usage, because the flush operations are performed during the next startup, using the same mechanism as in **crash recovery**. In cases where the database is being shut down for an upgrade or downgrade, do a **slow shutdown** instead to ensure that all relevant changes are applied to the **data files** during the shutdown.
See Also crash recovery, data files, flush, shutdown, slow shutdown.

file format

The format used by InnoDB for each table, typically with the **file-per-table** setting enabled so that each table is stored in a separate **`.ibd` file**. Currently, the file formats available in InnoDB are known as **Antelope** and **Barracuda**. Each file format supports one or more **row formats**. The row formats available for Barracuda tables, **COMPRESSED** and **DYNAMIC**, enable important new storage features for InnoDB tables.
See Also Antelope, Barracuda, file-per-table, .ibd file, ibdata file, row format.

file-per-table

A general name for the setting controlled by the `innodb_file_per_table` option. That is a very important configuration option that affects many aspects of InnoDB file storage, availability of features, and I/O characteristics. In MySQL 5.6.7 and higher, it is enabled by default. Prior to MySQL 5.6.7, it is disabled by default.

For each table created while this setting is in effect, the data is stored in a separate **.ibd file** rather than in the **ibdata files** of the **system tablespace**. When table data is stored in individual files, you have more flexibility to choose nondefault **file formats** and **row formats**, which are required for features such as data **compression**. The `TRUNCATE TABLE` operation is also much faster, and the reclaimed space can be used by the operating system rather than remaining reserved for InnoDB.

The **MySQL Enterprise Backup** product is more flexible for tables that are in their own files. For example, tables can be excluded from a backup, but only if they are in separate files. Thus, this setting is suitable for tables that are backed up less frequently or on a different schedule.
See Also compressed row format, compression, file format, .ibd file, ibdata file, innodb_file_per_table, row format, system tablespace.

fill factor

In an InnoDB **index**, the proportion of a **page** that is taken up by index data before the page is split. The unused space when index data is first divided between pages allows for rows to be updated with longer string values without requiring expensive index maintenance operations. If the fill factor is too low, the index consumes more space than needed, causing extra I/O overhead when reading the index. If the fill factor is too high, any update that increases the length of column values can cause extra I/O overhead for index maintenance. See Physical Structure of an `InnoDB` Index for more information.
See Also index, page.

fixed row format

This row format is used by the MyISAM storage engine, not by InnoDB. If you create an InnoDB table with the option `row_format=fixed`, InnoDB translates this option to use the **compact row format** instead, although the `fixed` value might still show up in output such as `SHOW TABLE STATUS` reports.
See Also compact row format, row format.

flush

To write changes to the database files, that had been buffered in a memory area or a temporary disk storage area. The InnoDB storage structures that are periodically flushed include the **redo log**, the **undo log**, and the **buffer pool**.

Flushing can happen because a memory area becomes full and the system needs to free some space, because a **commit** operation means the changes from a transaction can be finalized, or because a **slow shutdown** operation means that all outstanding work should be finalized. When it is not critical to flush all the buffered data

at once, InnoDB can use a technique called **fuzzy checkpointing** to flush small batches of pages to spread out the I/O overhead.
See Also buffer pool, commit, fuzzy checkpointing, neighbor page, redo log, slow shutdown, undo log.

flush list
An internal InnoDB data structure that tracks **dirty pages** in the **buffer pool**: that is, **pages** that have been changed and need to be written back out to disk. This data structure is updated frequently by InnoDB's internal **mini-transactions**, and so is protected by its own **mutex** to allow concurrent access to the buffer pool.
See Also buffer pool, dirty page, LRU, mini-transaction, mutex, page, page cleaner.

foreign key
A type of pointer relationship, between rows in separate InnoDB tables. The foreign key relationship is defined on one column in both the **parent table** and the **child table**.

In addition to enabling fast lookup of related information, foreign keys help to enforce **referential integrity**, by preventing any of these pointers from becoming invalid as data is inserted, updated, and deleted. This enforcement mechanism is a type of **constraint**. A row that points to another table cannot be inserted if the associated foreign key value does not exist in the other table. If a row is deleted or its foreign key value changed, and rows in another table point to that foreign key value, the foreign key can be set up to prevent the deletion, cause the corresponding column values in the other table to become **null**, or automatically delete the corresponding rows in the other table.

One of the stages in designing a **normalized** database is to identify data that is duplicated, separate that data into a new table, and set up a foreign key relationship so that the multiple tables can be queried like a single table, using a **join** operation.
See Also child table, FOREIGN KEY constraint, join, normalized, NULL, parent table, referential integrity, relational.

FOREIGN KEY constraint
The type of **constraint** that maintains database consistency through a **foreign key** relationship. Like other kinds of constraints, it can prevent data from being inserted or updated if data would become inconsistent; in this case, the inconsistency being prevented is between data in multiple tables. Alternatively, when a **DML** operation is performed, FOREIGN KEY constraints can cause data in **child rows** to be deleted, changed to different values, or set to **null**, based on the ON CASCADE option specified when creating the foreign key.
See Also child table, constraint, DML, foreign key, NULL.

FTS
In most contexts, an acronym for **full-text search**. Sometimes in performance discussions, an acronym for **full table scan**.
See Also full table scan, full-text search.

full backup
A **backup** that includes all the **tables** in each MySQL **database**, and all the databases in a MySQL **instance**. Contrast with **partial backup**.
See Also backup, database, instance, partial backup, table.

full table scan
An operation that requires reading the entire contents of a table, rather than just selected portions using an index. Typically performed either with small lookup tables, or in data warehousing situations with large tables where all available data is aggregated and analyzed. How frequently these operations occur, and the sizes of the tables relative to available memory, have implications for the algorithms used in query optimization and managing the buffer pool.

The purpose of **indexes** is to allow lookups for specific values or ranges of values within a large table, thus avoiding full table scans when practical.
See Also buffer pool, index, LRU.

full-text search

The MySQL feature for finding words, phrases, Boolean combinations of words, and so on within table data, in a faster, more convenient, and more flexible way than using the SQL `LIKE` operator or writing your own application-level search algorithm. It uses the SQL function `MATCH()` [1271] and **FULLTEXT indexes**.
See Also FULLTEXT index.

FULLTEXT index

The special kind of **index** that holds the **search index** in the MySQL **full-text search** mechanism. Represents the words from values of a column, omitting any that are specified as **stopwords**. Originally, only available for `MyISAM` tables. Starting in MySQL 5.6.4, it is also available for **InnoDB** tables.
See Also full-text search, index, InnoDB, search index, stopword.

fuzzy checkpointing

A technique that **flushes** small batches of **dirty pages** from the **buffer pool**, rather than flushing all dirty pages at once which would disrupt database processing.
See Also buffer pool, dirty page, flush.

G

GA

"Generally available", the stage when a software product leaves beta and is available for sale, official support, and production use.
See Also beta, early adopter.

gap

A place in an InnoDB **index** data structure where new values could be inserted. When you lock a set of rows with a statement such as `SELECT ... FOR UPDATE`, InnoDB can create locks that apply to the gaps as well as the actual values in the index. For example, if you select all values greater than 10 for update, a gap lock prevents another transaction from inserting a new value that is greater than 10. The **supremum record** and **infimum record** represent the gaps containing all values greater than or less than all the current index values.
See Also concurrency, gap lock, index, infimum record, isolation level, supremum record.

gap lock

A **lock** on a **gap** between index records, or a lock on the gap before the first or after the last index record. For example, `SELECT c1 FOR UPDATE FROM t WHERE c1 BETWEEN 10 and 20;` prevents other transactions from inserting a value of 15 into the column `t.c1`, whether or not there was already any such value in the column, because the gaps between all existing values in the range are locked. Contrast with **record lock** and **next-key lock**.

Gap locks are part of the tradeoff between performance and **concurrency**, and are used in some transaction **isolation levels** and not others.
See Also gap, infimum record, lock, next-key lock, record lock, supremum record.

general log

See general query log.

general query log

A type of **log** used for diagnosis and troubleshooting of SQL statements processed by the MySQL server. Can be stored in a file or in a database table. You must enable this feature through the `general_log` configuration option to use it. You can disable it for a specific connection through the `sql_log_off` configuration option.

Records a broader range of queries than the **slow query log**. Unlike the **binary log**, which is used for replication, the general query log contains `SELECT` statements and does not maintain strict ordering. For more information, see Section 5.2.3, “The General Query Log”.
See Also binary log, general query log, log.

global_transaction

A type of **transaction** involved in **XA** operations. It consists of several actions that are transactional in themselves, but that all must either complete successfully as a group, or all be rolled back as a group. In essence, this extends **ACID** properties "up a level" so that multiple ACID transactions can be executed in concert as components of a global operation that also has ACID properties. For this type of distributed transaction, you must use the **SERIALIZABLE** isolation level to achieve ACID properties.
See Also ACID, SERIALIZABLE, transaction, XA.

group commit

An `InnoDB` optimization that performs some low-level I/O operations (**log write**) once for a set of **commit** operations, rather than flushing and syncing separately for each commit.

When the binlog is enabled, you typically also set the configuration option `sync_binlog=0`, because group commit for the binary log is only supported if it is set to 0.
See Also commit, plugin, XA.

H

hash index

A type of **index** intended for queries that use equality operators, rather than range operators such as greater-than or `BETWEEN`. It is available for MEMORY tables. Although hash indexes are the default for MEMORY tables for historic reasons, that storage engine also supports **B-tree** indexes, which are often a better choice for general-purpose queries.

MySQL includes a variant of this index type, the **adaptive hash index**, that is constructed automatically for InnoDB tables if needed based on runtime conditions.
See Also adaptive hash index, B-tree, index, InnoDB.

HDD

Acronym for "hard disk drive". Refers to storage media using spinning platters, usually when comparing and contrasting with **SSD**. Its performance characteristics can influence the throughput of a **disk-based** workload.
See Also disk-based, SSD.

heartbeat

A periodic message that is sent to indicate that a system is functioning properly. In a **replication** context, if the **master** stops sending such messages, one of the **slaves** can take its place. Similar techniques can be used between the servers in a cluster environment, to confirm that all of them are operating properly.
See Also replication.

high-water mark

A value representing an upper limit, either a hard limit that should not be exceeded at runtime, or a record of the maximum value that was actually reached. Contrast with **low-water mark**.
See Also low-water mark.

history list

A list of **transactions** with delete-marked records scheduled to be processed by the `InnoDB` **purge** operation. Recorded in the **undo log**. The length of the history list is reported by the command `SHOW ENGINE INNODB STATUS`. If the history list grows longer than the value of the `innodb_max_purge_lag` configuration option, each **DML** operation is delayed slightly to allow the purge operation to finish **flushing** the deleted records.

Also known as **purge lag**.
See Also flush, purge, purge lag, rollback segment, transaction, undo log.

hot

A condition where a row, table, or internal data structure is accessed so frequently, requiring some form of locking or mutual exclusion, that it results in a performance or scalability issue.

Although "hot" typically indicates an undesirable condition, a **hot backup** is the preferred type of backup.
See Also hot backup.

hot backup

A backup taken while the database and is running and applications are reading and writing to it. The backup involves more than simply copying data files: it must include any data that was inserted or updated while the backup was in process; it must exclude any data that was deleted while the backup was in process; and it must ignore any changes that were not committed.

The Oracle product that performs hot backups, of InnoDB tables especially but also tables from MyISAM and other storage engines, is known as **MySQL Enterprise Backup**.

The hot backup process consists of two stages. The initial copying of the data files produces a **raw backup**. The **apply** step incorporates any changes to the database that happened while the backup was running. Applying the changes produces a **prepared** backup; these files are ready to be restored whenever necessary.
See Also apply, MySQL Enterprise Backup, prepared backup, raw backup.

I

.ibd file

Each InnoDB **table** created using the **file-per-table** mode goes into its own **tablespace** file, with a `.ibd` extension, inside the **database** directory. This file contains the table data and any **indexes** for the table. File-per-table mode, controlled by the **innodb_file_per_table** option, affects many aspects of InnoDB storage usage and performance, and is enabled by default in MySQL 5.6.7 and higher.

This extension does not apply to the **system tablespace**, which consists of the **ibdata files**.

When a `.ibd` file is included in a compressed backup by the **MySQL Enterprise Backup** product, the compressed equivalent is a `.ibz` file.

If a table is create with the `DATA DIRECTORY =` clause in MySQL 5.6 and higher, the `.ibd` file is located outside the normal database directory, and is pointed to by a **.isl file**.
See Also database, file-per-table, ibdata file, .ibz file, index, innodb_file_per_table, .isl file, MySQL Enterprise Backup, system tablespace, table, tablespace.

.ibz file

When the **MySQL Enterprise Backup** product performs a **compressed backup**, it transforms each **tablespace** file that is created using the **file-per-table** setting from a `.ibd` extension to a `.ibz` extension.

The compression applied during backup is distinct from the **compressed row format** that keeps table data compressed during normal operation. A compressed backup operation skips the compression step for a tablespace that is already in compressed row format, as compressing a second time would slow down the backup but produce little or no space savings.
See Also compressed backup, compressed row format, file-per-table, .ibd file, MySQL Enterprise Backup, tablespace.

.isl file

A file that specifies the location of a **.ibd file** for an InnoDB table created with the `DATA DIRECTORY =` clause in MySQL 5.6 and higher. It functions like a symbolic link, without the platform restrictions of the actual symbolic link mechanism. You can store InnoDB **tablespaces** outside the **database** directory, for example, on an especially large or fast storage device depending on the usage of the table. For details, see Section 14.2.5.4, “Specifying the Location of a Tablespace”.
See Also database, .ibd file, table, tablespace.

I/O-bound

See disk-bound.

ib-file set

The set of files managed by InnoDB within a MySQL database: the **system tablespace**, any **file-per-table** tablespaces, and the (typically 2) **redo log** files. Used sometimes in detailed discussions of InnoDB file structures and formats, to avoid ambiguity between the meanings of **database** between different DBMS products, and the non-InnoDB files that may be part of a MySQL database.
See Also database, file-per-table, redo log, system tablespace.

ibbackup_logfile

A supplemental backup file created by the **MySQL Enterprise Backup** product during a **hot backup** operation. It contains information about any data changes that occurred while the backup was running. The initial backup files, including `ibbackup_logfile`, are known as a **raw backup**, because the changes that occurred during the backup operation are not yet incorporated. After you perform the **apply** step to the raw backup files, the resulting files do include those final data changes, and are known as a **prepared backup**. At this stage, the `ibbackup_logfile` file is no longer necessary.
See Also apply, hot backup, MySQL Enterprise Backup, prepared backup, raw backup.

ibdata file

A set of files with names such as `ibdata1`, `ibdata2`, and so on, that make up the InnoDB **system tablespace**. These files contain metadata about InnoDB tables, (the **data dictionary**), and the storage areas for the **undo log**, the **change buffer**, and the **doublewrite buffer**. They also can contain some or all of the table data also (depending on whether the **file-per-table** mode is in effect when each table is created). When the **innodb_file_per_table** option is enabled, data and indexes for newly created tables are stored in separate **.ibd files** rather than in the system tablespace.

The growth of the `ibdata` files is influenced by the `innodb_autoextend_increment` configuration option.
See Also change buffer, data dictionary, doublewrite buffer, file-per-table, .ibd file, innodb_file_per_table, system tablespace, undo log.

ibtmp file

The InnoDB temporary tablespace data file for non-compressed `InnoDB` temporary tables and related objects. The configuration file option, `innodb_temp_data_file_path`, allows users to define a relative path for the temporary data file. If `innodb_temp_data_file_path` is not specified, the default behavior is to create a single auto- extending 12MB data file named `ibtmp1` in the data directory, alongside `ibdata1`.
See Also temporary tablespace.

ib_logfile

A set of files, typically named `ib_logfile0` and `ib_logfile1`, that form the **redo log**. Also sometimes referred to as the **log group**. These files record statements that attempt to change data in InnoDB tables. These statements are replayed automatically to correct data written by incomplete transactions, on startup following a crash.

This data cannot be used for manual recovery; for that type of operation, use the **binary log**.
See Also binary log, log group, redo log.

ilist

Within an InnoDB **FULLTEXT index**, the data structure consisting of a document ID and positional information for a token (that is, a particular word).
See Also FULLTEXT index.

implicit row lock

A row lock that InnoDB acquires to ensure consistency, without you specifically requesting it.
See Also row lock.

in-memory database

A type of database system that maintains data in memory, to avoid overhead due to disk I/O and translation between disk blocks and memory areas. Some in-memory databases sacrifice durability (the "D" in the **ACID** design philosophy) and are vulnerable to hardware, power, and other types of failures, making them more suitable for read-only operations. Other in-memory databases do use durability mechanisms such as logging changes to disk or using non-volatile memory.

MySQL features that are address the same kinds of memory-intensive processing include the InnoDB **buffer pool**, **adaptive hash index**, and **read-only transaction** optimization, the MEMORY storage engine, the MyISAM key cache, and the MySQL **query cache**.
See Also ACID, adaptive hash index, buffer pool, disk-based, read-only transaction.

incremental backup

A type of **hot backup**, performed by the **MySQL Enterprise Backup** product, that only saves data changed since some point in time. Having a full backup and a succession of incremental backups lets you reconstruct backup data over a long period, without the storage overhead of keeping several full backups on hand. You can restore the full backup and then apply each of the incremental backups in succession, or you can keep the full backup up-to-date by applying each incremental backup to it, then perform a single restore operation.

The granularity of changed data is at the **page** level. A page might actually cover more than one row. Each changed page is included in the backup.
See Also hot backup, MySQL Enterprise Backup, page.

index

A data structure that provides a fast lookup capability for **rows** of a **table**, typically by forming a tree structure (**B-tree)** representing all the values of a particular **column** or set of columns.

InnoDB tables always have a **clustered index** representing the **primary key**. They can also have one or more **secondary indexes** defined on one or more columns. Depending on their structure, secondary indexes can be classified as **partial**, **column**, or **composite** indexes.

Indexes are a crucial aspect of **query** performance. Database architects design tables, queries, and indexes to allow fast lookups for data needed by applications. The ideal database design uses a **covering index** where practical; the query results are computed entirely from the index, without reading the actual table data. Each **foreign key** constraint also requires an index, to efficiently check whether values exist in both the **parent** and **child** tables.

Although a B-tree index is the most common, a different kind of data structure is used for **hash indexes**, as in the `MEMORY` storage engine and the InnoDB **adaptive hash index**.
See Also adaptive hash index, B-tree, child table, clustered index, column index, composite index, covering index, foreign key, hash index, parent table, partial index, primary key, query, row, secondary index, table.

index cache

A memory area that holds the token data for InnoDB **full-text search**. It buffers the data to minimize disk I/O when data is inserted or updated in columns that are part of a **FULLTEXT index**. The token data is written to disk when the index cache becomes full. Each InnoDB `FULLTEXT` index has its own separate index cache, whose size is controlled by the configuration option `innodb_ft_cache_size`.
See Also full-text search, FULLTEXT index.

index hint

Extended SQL syntax for overriding the **indexes** recommended by the optimizer. For example, the `FORCE INDEX`, `USE INDEX`, and `IGNORE INDEX` clauses. Typically used when indexed columns have unevenly distributed values, resulting in inaccurate **cardinality** estimates.
See Also cardinality, index.

index prefix

In an **index** that applies to multiple columns (known as a **composite index**), the initial or leading columns of the index. A query that references the first 1, 2, 3, and so on columns of a composite index can use the index, even if the query does not reference all the columns in the index.
See Also composite index, index.

index statistics

See statistics.

infimum record

A **pseudo-record** in an **index**, representing the **gap** below the smallest value in that index. If a transaction has a statement such as `SELECT ... FOR UPDATE ... WHERE col < 10;`, and the smallest value in the column is 5, it is a lock on the infimum record that prevents other transactions from inserting even smaller values such as 0, -10, and so on.

See Also gap, index, pseudo-record, supremum record.

INFORMATION_SCHEMA

The name of the **database** that provides a query interface to the MySQL **data dictionary**. (This name is defined by the ANSI SQL standard.) To examine information (metadata) about the database, you can query tables such as `INFORMATION_SCHEMA.TABLES` and `INFORMATION_SCHEMA.COLUMNS`, rather than using `SHOW` commands that produce unstructured output.

The information schema contains some tables that are specific to **InnoDB**, such as `INNODB_LOCKS` and `INNODB_TRX`. You use these tables not to see how the database is structured, but to get real-time information about the workings of InnoDB tables to help with performance monitoring, tuning, and troubleshooting. In particular, these tables provide information about MySQL features related to **compression**, and **transactions** and their associated **locks**.

See Also compression, data dictionary, database, InnoDB, lock, transaction.

InnoDB

A MySQL component that combines high performance with **transactional** capability for reliability, robustness, and concurrent access. It embodies the **ACID** design philosophy. Represented as a **storage engine**; it handles tables created or altered with the `ENGINE=INNODB` clause. See Section 14.2, "The `InnoDB` Storage Engine" for architectural details and administration procedures, and Section 8.5, "Optimizing for `InnoDB` Tables" for performance advice.

In MySQL 5.5 and higher, InnoDB is the default storage engine for new tables and the `ENGINE=INNODB` clause is not required. In MySQL 5.1 only, many of the advanced InnoDB features require enabling the component known as the InnoDB Plugin. See Section 14.2.1.1, "`InnoDB` as the Default MySQL Storage Engine" for the considerations involved in transitioning to recent releases where InnoDB tables are the default.

InnoDB tables are ideally suited for **hot backups**. See Section 23.2, "MySQL Enterprise Backup" for information about the **MySQL Enterprise Backup** product for backing up MySQL servers without interrupting normal processing.

See Also ACID, hot backup, storage engine, transaction.

innodb_autoinc_lock_mode

The `innodb_autoinc_lock_mode` option controls the algorithm used for **auto-increment locking**. When you have an auto-incrementing **primary key**, you can use statement-based replication only with the setting `innodb_autoinc_lock_mode=1`. This setting is known as **consecutive** lock mode, because multi-row inserts within a transaction receive consecutive auto-increment values. If you have `innodb_autoinc_lock_mode=2`, which allows higher concurrency for insert operations, use row-based replication rather than statement-based replication. This setting is known as **interleaved** lock mode, because multiple multi-row insert statements running at the same time can receive autoincrement values that are interleaved. The setting `innodb_autoinc_lock_mode=0` is the previous (traditional) default setting and should not be used except for compatibility purposes.

See Also auto-increment locking, mixed-mode insert, primary key.

innodb_file_format

The `innodb_file_format` option determines the **file format** for all InnoDB **tablespaces** created after you specify a value for this option. To create tablespaces other than the **system tablespace**, you must also use the **file-per-table** option. Currently, you can specify the **Antelope** and **Barracuda** file formats.

See Also Antelope, Barracuda, file format, file-per-table, innodb_file_per_table, system tablespace, tablespace.

innodb_file_per_table

A very important configuration option that affects many aspects of InnoDB file storage, availability of features, and I/O characteristics. In MySQL 5.6.7 and higher, it is enabled by default. Prior to MySQL 5.6.7, it is disabled by default. The `innodb_file_per_table` option turns on **file-per-table** mode, which stores each newly created InnoDB table and its associated index in its own **.ibd file**, outside the **system tablespace**.

This option affects the performance and storage considerations for a number of SQL statements, such as `DROP TABLE` and `TRUNCATE TABLE`.

This option is needed to take full advantage of many other InnoDB features, such as such as table **compression**, or backups of named tables in **MySQL Enterprise Backup**.

This option was once static, but can now be set using the `SET GLOBAL` command.

For reference information, see `innodb_file_per_table`. For usage information, see Section 14.2.5.2, “InnoDB File-Per-Table Mode”.
See Also compression, file-per-table, .ibd file, MySQL Enterprise Backup, system tablespace.

innodb_lock_wait_timeout

The `innodb_lock_wait_timeout` option sets the balance between **waiting** for shared resources to become available, or giving up and handling the error, retrying, or doing alternative processing in your application. Rolls back any InnoDB transaction that waits more than a specified time to acquire a **lock**. Especially useful if **deadlocks** are caused by updates to multiple tables controlled by different storage engines; such deadlocks are not **detected** automatically.
See Also deadlock, deadlock detection, lock, wait.

innodb_strict_mode

The `innodb_strict_mode` option controls whether InnoDB operates in **strict mode**, where conditions that are normally treated as warnings, cause errors instead (and the underlying statements fail).

This mode is the default setting in MySQL 5.5.5 and higher.
See Also strict mode.

insert

One of the primary **DML** operations in **SQL**. The performance of inserts is a key factor in **data warehouse** systems that load millions of rows into tables, and **OLTP** systems where many concurrent connections might insert rows into the same table, in arbitrary order. If insert performance is important to you, you should learn about **InnoDB** features such as the **insert buffer** used in **change buffering**, and **auto-increment** columns.
See Also auto-increment, change buffering, data warehouse, DML, InnoDB, insert buffer, OLTP, SQL.

insert buffer

Former name for the **change buffer**. Now that **change buffering** includes delete and update operations as well as inserts, "change buffer" is the preferred term.
See Also change buffer, change buffering.

insert buffering

The technique of storing secondary index changes due to `INSERT` operations in the **insert buffer** rather than writing them immediately, so that the physical writes can be performed to minimize random I/O. It is one of the types of **change buffering**; the others are **delete buffering** and **purge buffering**.

Insert buffering is not used if the secondary index is **unique**, because the uniqueness of new values cannot be verified before the new entries are written out. Other kinds of change buffering do work for unique indexes.
See Also change buffer, change buffering, delete buffering, insert buffer, purge buffering, unique index.

instance

A single **mysqld** daemon managing a **data directory** representing one or more **databases** with a set of **tables**. It is common in development, testing, and some **replication** scenarios to have multiple instances on the same **server** machine, each managing its own data directory and listening on its own port or socket. With one instance running a **disk-bound** workload, the server might still have extra CPU and memory capacity to run additional instances.

See Also data directory, database, disk-bound, mysqld, replication, server.

instrumentation

Modifications at the source code level to collect performance data for tuning and debugging. In MySQL, data collected by instrumentation is exposed through a SQL interface using the `INFORMATION_SCHEMA` and `PERFORMANCE_SCHEMA` databases.

See Also INFORMATION_SCHEMA, Performance Schema.

intention exclusive lock

See intention lock.

intention lock

A kind of **lock** that applies to the table level, used to indicate what kind of lock the transaction intends to acquire on rows in the table. Different transactions can acquire different kinds of intention locks on the same table, but the first transaction to acquire an **intention exclusive** (IX) lock on a table prevents other transactions from acquiring any S or X locks on the table. Conversely, the first transaction to acquire an **intention shared** (IS) lock on a table prevents other transactions from acquiring any X locks on the table. The two-phase process allows the lock requests to be resolved in order, without blocking locks and corresponding operations that are compatible. For more details on this locking mechanism, see Section 14.2.2.3, “`InnoDB` Lock Modes”.

See Also lock, lock mode, locking.

intention shared lock

See intention lock.

inverted index

A data structure optimized for document retrieval systems, used in the implementation of InnoDB **full-text search**. The InnoDB **FULLTEXT index**, implemented as an inverted index, records the position of each word within a document, rather than the location of a table row. A single column value (a document stored as a text string) is represented by many entries in the inverted index.

See Also full-text search, FULLTEXT index, ilist.

IOPS

Acronym for **I/O operations per second**. A common measurement for busy systems, particularly **OLTP** applications. If this value is near the maximum that the storage devices can handle, the application can become **disk-bound**, limiting **scalability**.

See Also disk-bound, OLTP, scalability.

isolation level

One of the foundations of database processing. Isolation is the **I** in the acronym **ACID**; the isolation level is the setting that fine-tunes the balance between performance and reliability, consistency, and reproducibility of results when multiple **transactions** are making changes and performing queries at the same time.

From highest amount of consistency and protection to the least, the isolation levels supported by InnoDB are: **SERIALIZABLE**, **REPEATABLE READ**, **READ COMMITTED**, and **READ UNCOMMITTED**.

With InnoDB tables, many users can keep the default isolation level (**REPEATABLE READ**) for all operations. Expert users might choose the **read committed** level as they push the boundaries of scalability with OLTP processing, or during data warehousing operations where minor inconsistencies do not affect the aggregate results of large amounts of data. The levels on the edges (**SERIALIZABLE** and **READ UNCOMMITTED**) change the processing behavior to such an extent that they are rarely used.

See Also ACID, READ COMMITTED, READ UNCOMMITTED, REPEATABLE READ, SERIALIZABLE, transaction.

J

join

A **query** that retrieves data from more than one table, by referencing columns in the tables that hold identical values. Ideally, these columns are part of an InnoDB **foreign key** relationship, which ensures **referential integrity** and that the join columns are **indexed**. Often used to save space and improve query performance by replacing repeated strings with numeric IDs, in a **normalized** data design.
See Also foreign key, index, normalized, query, referential integrity.

K

KEY_BLOCK_SIZE

An option to specify the size of data pages within an InnoDB table that uses **compressed row format**. The default is 8 kilobytes. Lower values risk hitting internal limits that depend on the combination of row size and compression percentage.
See Also compressed row format.

L

latch

A lightweight structure used by InnoDB to implement a **lock** for its own internal memory structures, typically held for a brief time measured in milliseconds or microseconds. A general term that includes both **mutexes** (for exclusive access) and **rw-locks** (for shared access). Certain latches are the focus of InnoDB performance tuning, such as the **data dictionary** mutex. Statistics about latch use and contention are available through the **Performance Schema** interface.
See Also data dictionary, lock, locking, mutex, Performance Schema, rw-lock.

list

The InnoDB **buffer pool** is represented as a list of memory **pages**. The list is reordered as new pages are accessed and enter the buffer pool, as pages within the buffer pool are accessed again and are considered newer, and as pages that are not accessed for a long time are **evicted** from the buffer pool. The buffer pool is actually divided into **sublists**, and the replacement policy is a variation of the familiar **LRU** technique.
See Also buffer pool, eviction, LRU, sublist.

lock

The high-level notion of an object that controls access to a resource, such as a table, row, or internal data structure, as part of a **locking** strategy. For intensive performance tuning, you might delve into the actual structures that implement locks, such as **mutexes** and **latches**.
See Also latch, lock mode, locking, mutex.

lock escalation

An operation used in some database systems that converts many row locks into a single table lock, saving memory space but reducing concurrent access to the table. InnoDB uses a space-efficient representation for row locks, so that lock escalation is not needed.
See Also locking, row lock, table lock.

lock mode

A shared (S) lock allows a transaction to read a row. Multiple transactions can acquire an S lock on that same row at the same time.

An exclusive (X) lock allows a transaction to update or delete a row. No other transaction can acquire any kind of lock on that same row at the same time.

Intention locks apply to the table level, and are used to indicate what kind of lock the transaction intends to acquire on rows in the table. Different transactions can acquire different kinds of intention locks on the same table, but the first transaction to acquire an intention exclusive (IX) lock on a table prevents other transactions from acquiring any S or X locks on the table. Conversely, the first transaction to acquire an intention shared (IS) lock on a table prevents other transactions from acquiring any X locks on the table. The two-phase process allows the lock requests to be resolved in order, without blocking locks and corresponding operations that are compatible.
See Also intention lock, lock, locking.

locking

The system of protecting a **transaction** from seeing or changing data that is being queried or changed by other transactions. The locking strategy must balance reliability and consistency of database operations (the principles of the **ACID** philosophy) against the performance needed for good **concurrency**. Fine-tuning the locking strategy often involves choosing an **isolation level** and ensuring all your database operations are safe and reliable for that isolation level.
See Also ACID, concurrency, isolation level, latch, lock, mutex, transaction.

locking read

A `SELECT` statement that also performs a **locking** operation on an `InnoDB` table. Either `SELECT ... FOR UPDATE` or `SELECT ... LOCK IN SHARE MODE`. It has the potential to produce a **deadlock**, depending on the **isolation level** of the transaction. The opposite of a **non-locking read**. Not allowed for global tables in a **read-only transaction**.
See Also deadlock, isolation level, locking, non-locking read, read-only transaction.

log

In the InnoDB context, "log" or "log files" typically refers to the **redo log** represented by the **ib_logfile*** files. Another log area, which is physically part of the **system tablespace**, is the **undo log**.

Other kinds of logs that are important in MySQL are the **error log** (for diagnosing startup and runtime problems), **binary log** (for working with replication and performing point-in-time restores), the **general query log** (for diagnosing application problems), and the **slow query log** (for diagnosing performance problems).
See Also binary log, error log, general query log, ib_logfile, redo log, slow query log, system tablespace, undo log.

log buffer

The memory area that holds data to be written to the **log files** that make up the **redo log**. It is controlled by the `innodb_log_buffer_size` configuration option.
See Also log file, redo log.

log file

One of the `ib_logfileN` files that make up the **redo log**. Data is written to these files from the **log buffer** memory area.
See Also ib_logfile, log buffer, redo log.

log group

The set of files that make up the **redo log**, typically named `ib_logfile0` and `ib_logfile1`. (For that reason, sometimes referred to collectively as **ib_logfile**.)
See Also ib_logfile, redo log.

logical

A type of operation that involves high-level, abstract aspects such as tables, queries, indexes, and other SQL concepts. Typically, logical aspects are important to make database administration and application development convenient and usable. Contrast with **physical**.

See Also logical backup, physical.

logical backup

A **backup** that reproduces table structure and data, without copying the actual data files. For example, the **mysqldump** command produces a logical backup, because its output contains statements such as CREATE TABLE and INSERT that can re-create the data. Contrast with **physical backup**. A logical backup offers flexibility (for example, you could edit table definitions or insert statements before restoring), but can take substantially longer to **restore** than a physical backup.
See Also backup, mysqldump, physical backup, restore.

loose_

In MySQL 5.1, a prefix added to InnoDB configuration options when installing the InnoDB **Plugin** after server startup, so any new configuration options not recognized by the current level of MySQL do not cause a startup failure. MySQL processes configuration options that start with this prefix, but gives a warning rather than a failure if the part after the prefix is not a recognized option.
See Also plugin.

low-water mark

A value representing a lower limit, typically a threshold value at which some corrective action begins or becomes more aggressive. Contrast with **high-water mark**.
See Also high-water mark.

LRU

An acronym for "least recently used", a common method for managing storage areas. The items that have not been used recently are **evicted** when space is needed to cache newer items. InnoDB uses the LRU mechanism by default to manage the **pages** within the **buffer pool**, but makes exceptions in cases where a page might be read only a single time, such as during a **full table scan**. This variation of the LRU algorithm is called the **midpoint insertion strategy**. The ways in which the buffer pool management differs from the traditional LRU algorithm is fine-tuned by the options innodb_old_blocks_pct, innodb_old_blocks_time, and the new MySQL 5.6 options innodb_lru_scan_depth and innodb_flush_neighbors.
See Also buffer pool, eviction, full table scan, midpoint insertion strategy, page.

LSN

Acronym for "log sequence number". This arbitrary, ever-increasing value represents a point in time corresponding to operations recorded in the **redo log**. (This point in time is regardless of **transaction** boundaries; it can fall in the middle of one or more transactions.) It is used internally by InnoDB during **crash recovery** and for managing the buffer pool.

Prior to MySQL 5.6.3, the LSN was a 4-byte unsigned integer. The LSN became an 8-byte unsigned integer in MySQL 5.6.3 when the redo log file size limit increased from 4GB to 512GB, as additional bytes were required to store extra size information. Applications built on MySQL 5.6.3 or later that use LSN values should use 64-bit rather than 32-bit variables to store and compare LSN values.

In the **MySQL Enterprise Backup** product, you can specify an LSN to represent the point in time from which to take an **incremental backup**. The relevant LSN is displayed by the output of the mysqlbackup command. Once you have the LSN corresponding to the time of a full backup, you can specify that value to take a subsequent incremental backup, whose output contains another LSN for the next incremental backup.
See Also crash recovery, incremental backup, MySQL Enterprise Backup, redo log, transaction.

M

.MRG file

A file containing references to other tables, used by the MERGE storage engine. Files with this extension are always included in backups produced by the mysqlbackup command of the **MySQL Enterprise Backup** product.

See Also MySQL Enterprise Backup, mysqlbackup command.

.MYD file

A file that MySQL uses to store data for a MyISAM table.
See Also .MYI file, MySQL Enterprise Backup, mysqlbackup command.

.MYI file

A file that MySQL uses to store indexes for a MyISAM table.
See Also .MYD file, MySQL Enterprise Backup, mysqlbackup command.

master server

Frequently shortened to "master". A database server machine in a **replication** scenario that processes the initial insert, update, and delete requests for data. These changes are propagated to, and repeated on, other servers known as **slave servers**.
See Also replication, slave server.

master thread

An InnoDB **thread** that performs various tasks in the background. Most of these tasks are I/O related, such as writing changes from the **insert buffer** to the appropriate secondary indexes.

To improve **concurrency**, sometimes actions are moved from the master thread to separate background threads. For example, in MySQL 5.6 and higher, **dirty pages** are **flushed** from the **buffer pool** by the **page cleaner** thread rather than the master thread.
See Also buffer pool, dirty page, flush, insert buffer, page cleaner, thread.

MDL

Acronym for "metadata lock".
See Also metadata lock.

memcached

A popular component of many MySQL and **NoSQL** software stacks, allowing fast reads and writes for single values and caching the results entirely in memory. Traditionally, applications required extra logic to write the same data to a MySQL database for permanent storage, or to read data from a MySQL database when it was not cached yet in memory. Now, applications can use the simple `memcached` protocol, supported by client libraries for many languages, to communicate directly with MySQL servers using **InnoDB** or MySQL Cluster tables. These NoSQL interfaces to MySQL tables allow applications to achieve higher read and write performance than by issuing SQL commands directly, and can simplify application logic and deployment configurations for systems that already incorporated `memcached` for in-memory caching.

The `memcached` interface to InnoDB tables is available in MySQL 5.6 and higher; see Section 14.2.16, "`InnoDB` Integration with memcached" for details. The `memcached` interface to MySQL Cluster tables is available in MySQL Cluster 7.2; see http://dev.mysql.com/doc/ndbapi/en/ndbmemcache.html for details.
See Also InnoDB, NoSQL.

merge

To apply changes to data cached in memory, such as when a page is brought into the **buffer pool**, and any applicable changes recorded in the **change buffer** are incorporated into the page in the buffer pool. The updated data is eventually written to the **tablespace** by the **flush** mechanism.
See Also buffer pool, change buffer, flush, tablespace.

metadata lock

A type of **lock** that prevents **DDL** operations on a table that is being used at the same time by another **transaction**. For details, see Section 8.10.4, "Metadata Locking".

Enhancements to **online** operations, particularly in MySQL 5.6 and higher, are focused on reducing the amount of metadata locking. The objective is for DDL operations that do not change the table structure (such as `CREATE`

INDEX and DROP INDEX for InnoDB tables) to proceed while the table is being queried, updated, and so on by other transactions.
See Also DDL, lock, online, transaction.

metrics counter
A feature implemented by the innodb_metrics table in the **information_schema**, in MySQL 5.6 and higher. You can query **counts** and totals for low-level InnoDB operations, and use the results for performance tuning in combination with data from the **performance_schema**.
See Also counter, INFORMATION_SCHEMA, Performance Schema.

midpoint insertion strategy
The technique of initially bringing **pages** into the InnoDB **buffer pool** not at the "newest" end of the list, but instead somewhere in the middle. The exact location of this point can vary, based on the setting of the innodb_old_blocks_pct option. The intent is that blocks that are only read once, such as during a **full table scan**, can be aged out of the buffer pool sooner than with a strict **LRU** algorithm.
See Also buffer pool, full table scan, LRU, page.

mini-transaction
An internal phase of InnoDB processing, when making changes at the **physical** level to internal data structures during **DML** operations. A mini-transaction has no notion of **rollback**; multiple mini-transactions can occur within a single **transaction**. Mini-transactions write information to the **redo log** that is used during **crash recovery**. A mini-transaction can also happen outside the context of a regular transaction, for example during **purge** processing by background threads.
See Also commit, crash recovery, DML, physical, purge, redo log, rollback, transaction.

mixed-mode insert
An INSERT statement where **auto-increment** values are specified for some but not all of the new rows. For example, a multi-value INSERT could specify a value for the auto-increment column in some cases and NULL in other cases. InnoDB generates auto-increment values for the rows where the column value was specified as NULL. Another example is an INSERT ... ON DUPLICATE KEY UPDATE statement, where auto-increment values might be generated but not used, for any duplicate rows that are processed as UPDATE rather than INSERT statements.

Can cause consistency issues between **master** and **slave** servers in a **replication** configuration. Can require adjusting the value of the **innodb_autoinc_lock_mode** configuration option.
See Also auto-increment, innodb_autoinc_lock_mode, master server, replication, slave server.

multi-core
A type of processor that can take advantage of multi-threaded programs, such as the MySQL server.

multiversion concurrency control
See MVCC.

mutex
Informal abbreviation for "mutex variable". (Mutex itself is short for "mutual exclusion".) The low-level object that InnoDB uses to represent and enforce exclusive-access **locks** to internal in-memory data structures. Once the lock is acquired, any other process, thread, and so on is prevented from acquiring the same lock. Contrast with **rw-locks**, which allow shared access. Mutexes and rw-locks are known collectively as **latches**.
See Also latch, lock, Performance Schema, Pthreads, rw-lock.

MVCC
Acronym for "multiversion concurrency control". This technique lets InnoDB **transactions** with certain **isolation levels** to perform **consistent read** operations; that is, to query rows that are being updated by other transactions, and see the values from before those updates occurred. This is a powerful technique to increase **concurrency**, by allowing queries to proceed without waiting due to **locks** held by the other transactions.

This technique is not universal in the database world. Some other database products, and some other MySQL storage engines, do not support it.
See Also ACID, concurrency, consistent read, isolation level, lock, transaction.

my.cnf
The name, on UNIX or Linux systems, of the MySQL option file.
See Also my.ini, option file.

my.ini
The name, on Windows systems, of the MySQL option file.
See Also my.cnf, option file.

mysql
The `mysql` program is the command-line interpreter for the MySQL database. It processes **SQL** statements, and also MySQL-specific commands such as `SHOW TABLES`, by passing requests to the **`mysqld`** daemon.
See Also mysqld, SQL.

MySQL Enterprise Backup
A licensed product that performs **hot backups** of MySQL databases. It offers the most efficiency and flexibility when backing up **InnoDB** tables, but can also back up MyISAM and other kinds of tables.
See Also hot backup, InnoDB.

mysqlbackup command
A command-line tool of the **MySQL Enterprise Backup** product. It performs a **hot backup** operation for InnoDB tables, and a warm backup for MyISAM and other kinds of tables. See Section 23.2, “MySQL Enterprise Backup” for more information about this command.
See Also hot backup, MySQL Enterprise Backup, warm backup.

mysqld
The `mysqld` program is the database engine for the MySQL database. It runs as a UNIX daemon or Windows service, constantly waiting for requests and performing maintenance work in the background.
See Also mysql.

mysqldump
A command that performs a **logical backup** of some combination of databases, tables, and table data. The results are SQL statements that reproduce the original schema objects, data, or both. For substantial amounts of data, a **physical backup** solution such as **MySQL Enterprise Backup** is faster, particularly for the **restore** operation.
See Also logical backup, MySQL Enterprise Backup, physical backup, restore.

N

natural key
A indexed column, typically a **primary key**, where the values have some real-world significance. Usually advised against because:

- If the value should ever change, there is potentially a lot of index maintenance to re-sort the **clustered index** and update the copies of the primary key value that are repeated in each **secondary index**.

- Even seemingly stable values can change in unpredictable ways that are difficult to represent correctly in the database. For example, one country can change into two or several, making the original country code obsolete. Or, rules about unique values might have exceptions. For example, even if taxpayer IDs are intended to be unique to a single person, a database might have to handle records that violate that rule, such as in cases of identity theft. Taxpayer IDs and other sensitive ID numbers also make poor primary keys, because they may need to be secured, encrypted, and otherwise treated differently than other columns.

Thus, it is typically better to use arbitrary numeric values to form a **synthetic key**, for example using an **auto-increment** column.
See Also auto-increment, primary key, secondary index, synthetic key.

neighbor page
Any **page** in the same **extent** as a particular page. When a page is selected to be **flushed**, any neighbor pages that are **dirty** are typically flushed as well, as an I/O optimization for traditional hard disks. In MySQL 5.6 and up, this behavior can be controlled by the configuration variable `innodb_flush_neighbors`; you might turn that setting off for SSD drives, which do not have the same overhead for writing smaller batches of data at random locations.
See Also dirty page, extent, flush, page.

next-key lock
A combination of a **record lock** on the index record and a gap lock on the gap before the index record.
See Also gap lock, locking, record lock.

non-blocking I/O
An industry term that means the same as **asynchronous I/O**.
See Also asynchronous I/O.

non-locking read
A **query** that does not use the `SELECT ... FOR UPDATE` or `SELECT ... LOCK IN SHARE MODE` clauses. The only kind of query allowed for global tables in a **read-only transaction**. The opposite of a **locking read**.
See Also locking read, query, read-only transaction.

non-repeatable read
The situation when a query retrieves data, and a later query within the same **transaction** retrieves what should be the same data, but the queries return different results (changed by another transaction committing in the meantime).

This kind of operation goes against the **ACID** principle of database design. Within a transaction, data should be consistent, with predictable and stable relationships.

Among different **isolation levels**, non-repeatable reads are prevented by the **serializable read** and **repeatable read** levels, and allowed by the **consistent read**, and **read uncommitted** levels.
See Also ACID, consistent read, isolation level, READ UNCOMMITTED, REPEATABLE READ, SERIALIZABLE, transaction.

normalized
A database design strategy where data is split into multiple tables, and duplicate values condensed into single rows represented by an ID, to avoid storing, querying, and updating redundant or lengthy values. It is typically used in **OLTP** applications.

For example, an address might be given a unique ID, so that a census database could represent the relationship **lives at this address** by associating that ID with each member of a family, rather than storing multiple copies of a complex value such as **123 Main Street, Anytown, USA**.

For another example, although a simple address book application might store each phone number in the same table as a person's name and address, a phone company database might give each phone number a special ID, and store the numbers and IDs in a separate table. This normalized representation could simplify large-scale updates when area codes split apart.

Normalization is not always recommended. Data that is primarily queried, and only updated by deleting entirely and reloading, is often kept in fewer, larger tables with redundant copies of duplicate values. This data representation is referred to as **denormalized**, and is frequently found in data warehousing applications.

See Also denormalized, foreign key, OLTP, relational.

NoSQL

A broad term for a set of data access technologies that do not use the **SQL** language as their primary mechanism for reading and writing data. Some NoSQL technologies act as key-value stores, only accepting single-value reads and writes; some relax the restrictions of the **ACID** methodology; still others do not require a pre-planned **schema**. MySQL users can combine NoSQL-style processing for speed and simplicity with SQL operations for flexibility and convenience, by using the **memcached** API to directly access some kinds of MySQL tables. The `memcached` interface to InnoDB tables is available in MySQL 5.6 and higher; see Section 14.2.16, “`InnoDB` Integration with memcached” for details. The `memcached` interface to MySQL Cluster tables is available in MySQL Cluster 7.2; see http://dev.mysql.com/doc/ndbapi/en/ndbmemcache.html for details.
See Also ACID, InnoDB, memcached, schema, SQL.

NOT NULL constraint

A type of **constraint** that specifies that a **column** cannot contain any **NULL** values. It helps to preserve **referential integrity**, as the database server can identify data with erroneous missing values. It also helps in the arithmetic involved in query optimization, allowing the optimizer to predict the number of entries in an index on that column.
See Also column, constraint, NULL, primary key, referential integrity.

NULL

A special value in **SQL**, indicating the absence of data. Any arithmetic operation or equality test involving a `NULL` value, in turn produces a `NULL` result. (Thus it is similar to the IEEE floating-point concept of NaN, "not a number".) Any aggregate calculation such as `AVG()` ignores rows with `NULL` values, when determining how many rows to divide by. The only test that works with `NULL` values uses the SQL idioms `IS NULL` or `IS NOT NULL`.

`NULL` values play a part in index operations, because for performance a database must minimize the overhead of keeping track of missing data values. Typically, `NULL` values are not stored in an index, because a query that tests an indexed column using a standard comparison operator could never match a row with a `NULL` value for that column. For the same reason, unique indexes do not prevent `NULL` values; those values simply are not represented in the index. Declaring a `NOT NULL` constraint on a column provides reassurance that there are no rows left out of the index, allowing for better query optimization (accurate counting of rows and estimation of whether to use the index).

Because the **primary key** must be able to uniquely identify every row in the table, a single-column primary key cannot contain any `NULL` values, and a multi-column primary key cannot contain any rows with `NULL` values in all columns.

Although the Oracle database allows a `NULL` value to be concatenated with a string, InnoDB treats the result of such an operation as `NULL`.
See Also index, primary key, SQL.

O

.OPT file

A file containing database configuration information. Files with this extension are always included in backups produced by the `mysqlbackup` command of the **MySQL Enterprise Backup** product.
See Also MySQL Enterprise Backup, mysqlbackup command.

off-page column

A column containing variable-length data (such as `BLOB` and `VARCHAR`) that is too long to fit on a **B-tree** page. The data is stored in **overflow pages**. The `DYNAMIC` row format in the InnoDB **Barracuda** file format is more efficient for such storage than the older `COMPACT` row format.
See Also B-tree, Barracuda, overflow page.

OLTP

Acronym for "Online Transaction Processing". A database system, or a database application, that runs a workload with many **transactions**, with frequent writes as well as reads, typically affecting small amounts of data at a time. For example, an airline reservation system or an application that processes bank deposits. The data might be organized in **normalized** form for a balance between **DML** (insert/update/delete) efficiency and **query** efficiency. Contrast with **data warehouse**.

With its **row-level locking** and **transactional** capability, **InnoDB** is the ideal storage engine for MySQL tables used in OLTP applications.
See Also data warehouse, DML, InnoDB, query, row lock, transaction.

online

A type of operation that involves no downtime, blocking, or restricted operation for the database. Typically applied to **DDL**. Operations that shorten the periods of restricted operation, such as **fast index creation**, have evolved into a wider set of **online DDL** operations in MySQL 5.6.

In the context of backups, a **hot backup** is an online operation and a **warm backup** is partially an online operation.
See Also DDL, Fast Index Creation, hot backup, online DDL, warm backup.

online DDL

A feature that improves the performance, concurrency, and availability of InnoDB tables during **DDL** (primarily `ALTER TABLE`) operations. See Section 14.2.11, "`InnoDB` and Online DDL" for details.

The details vary according to the type of operation. In some cases, the table can be modified concurrently while the `ALTER TABLE` is in progress. The operation might be able to be performed without doing a table copy, or using a specially optimized type of table copy. Space usage is controlled by the `innodb_online_alter_log_max_size` configuration option.

This feature is an enhancement of the **Fast Index Creation** feature in MySQL 5.5 and the InnoDB Plugin for MySQL 5.1.
See Also DDL, Fast Index Creation, online.

optimistic

A methodology that guides low-level implementation decisions for a relational database system. The requirements of performance and **concurrency** in a relational database mean that operations must be started or dispatched quickly. The requirements of consistency and **referential integrity** mean that any operation could fail: a transaction might be rolled back, a **DML** operation could violate a constraint, a request for a lock could cause a deadlock, a network error could cause a timeout. An optimistic strategy is one that assumes most requests or attempts will succeed, so that relatively little work is done to prepare for the failure case. When this assumption is true, the database does little unnecessary work; when requests do fail, extra work must be done to clean up and undo changes.

InnoDB uses optimistic strategies for operations such as **locking** and **commits**. For example, data changed by a transaction can be written to the data files before the commit occurs, making the commit itself very fast, but requiring more work to undo the changes if the transaction is rolled back.

The opposite of an optimistic strategy is a **pessimistic** one, where a system is optimized to deal with operations that are unreliable and frequently unsuccessful. This methodology is rare in a database system, because so much care goes into choosing reliable hardware, networks, and algorithms.
See Also commit, concurrency, DML, locking, pessimistic.

optimizer

The MySQL component that determines the best **indexes** and **join** order to use for a **query**, based on characteristics and data distribution of the relevant **tables**.
See Also index, join, query, table.

option

A configuration parameter for MySQL, either stored in the **option file** or passed on the command line.

For the options that apply to **InnoDB** tables, each option name starts with the prefix `innodb_`.
See Also InnoDB, option file.

option file

The file that holds the configuration **options** for the MySQL instance. Traditionally, on Linux and UNIX this file is named `my.cnf`, and on Windows it is named `my.ini`.
See Also configuration file, my.cnf, option.

overflow page

Separately allocated disk **pages** that hold variable-length columns (such as `BLOB` and `VARCHAR`) that are too long to fit on a **B-tree** page. The associated columns are known as **off-page columns**.
See Also B-tree, off-page column, page.

P

.PAR file

A file containing partition definitions. Files with this extension are always included in backups produced by the `mysqlbackup` command of the **MySQL Enterprise Backup** product.
See Also MySQL Enterprise Backup, mysqlbackup command.

page

A unit representing how much data InnoDB transfers at any one time between disk (the **data files**) and memory (the **buffer pool**). A page can contain one or more **rows**, depending on how much data is in each row. If a row does not fit entirely into a single page, InnoDB sets up additional pointer-style data structures so that the information about the row can be stored in one page.

One way to fit more data in each page is to use **compressed row format**. For tables that use BLOBs or large text fields, **compact row format** allows those large columns to be stored separately from the rest of the row, reducing I/O overhead and memory usage for queries that do not reference those columns.

When InnoDB reads or writes sets of pages as a batch to increase I/O throughput, it reads or writes an **extent** at a time.

All the InnoDB disk data structures within a MySQL instance share the same **page size**.
See Also buffer pool, compact row format, compressed row format, data files, extent, page size, row.

page cleaner

An InnoDB background **thread** that **flushes dirty pages** from the **buffer pool**. Prior to MySQL 5.6, this activity was performed by the **master thread**
See Also buffer pool, dirty page, flush, master thread, thread.

page size

For releases up to and including MySQL 5.5, the size of each InnoDB **page** is fixed at 16 kilobytes. This value represents a balance: large enough to hold the data for most rows, yet small enough to minimize the performance overhead of transferring unneeded data to memory. Other values are not tested or supported.

Starting in MySQL 5.6, the page size for an InnoDB **instance** can be either 4KB, 8KB, or 16KB, controlled by the `innodb_page_size` configuration option. You set the size when creating the MySQL instance, and it remains constant afterwards. The same page size applies to all InnoDB **tablespaces**, both the **system tablespace** and any separate tablespaces created in **file-per-table** mode.

Smaller page sizes can help performance with storage devices that use small block sizes, particularly for **SSD** devices in **disk-bound** workloads, such as for **OLTP** applications. As individual rows are updated, less data is copied into memory, written to disk, reorganized, locked, and so on.

See Also disk-bound, file-per-table, instance, OLTP, page, SSD, system tablespace, tablespace.

parent table

The table in a **foreign key** relationship that holds the initial column values pointed to from the **child table**. The consequences of deleting, or updating rows in the parent table depend on the `ON UPDATE` and `ON DELETE` clauses in the foreign key definition. Rows with corresponding values in the child table could be automatically deleted or updated in turn, or those columns could be set to `NULL`, or the operation could be prevented.
See Also child table, foreign key.

partial backup

A **backup** that contains some of the **tables** in a MySQL database, or some of the databases in a MySQL instance. Contrast with **full backup**.
See Also backup, full backup, table.

partial index

An **index** that represents only part of a column value, typically the first N characters (the **prefix**) of a long `VARCHAR` value.
See Also index, index prefix.

Performance Schema

The `performance_schema` schema, in MySQL 5.5 and up, presents a set of tables that you can query to get detailed information about the performance characteristics of many internal parts of the MySQL server.
See Also latch, mutex, rw-lock.

persistent statistics

A feature in MySQL 5.6 that stores **index** statistics for InnoDB **tables** on disk, providing better **plan stability** for **queries**.
See Also index, optimizer, plan stability, query, table.

pessimistic

A methodology that sacrifices performance or concurrency in favor of safety. It is appropriate if a high proportion of requests or attempts might fail, or if the consequences of a failed request are severe. InnoDB uses what is known as a pessimistic **locking** strategy, to minimize the chance of **deadlocks**. At the application level, you might avoid deadlocks by using a pessimistic strategy of acquiring all locks needed by a transaction at the very beginning.

Many built-in database mechanisms use the opposite **optimistic** methodology.
See Also deadlock, locking, optimistic.

phantom

A row that appears in the result set of a query, but not in the result set of an earlier query. For example, if a query is run twice within a **transaction**, and in the meantime, another transaction commits after inserting a new row or updating a row so that it matches the `WHERE` clause of the query.

This occurrence is known as a phantom read. It is harder to guard against than a **non-repeatable read**, because locking all the rows from the first query result set does not prevent the changes that cause the phantom to appear.

Among different **isolation levels**, phantom reads are prevented by the **serializable read** level, and allowed by the **repeatable read**, **consistent read**, and **read uncommitted** levels.
See Also consistent read, isolation level, non-repeatable read, READ UNCOMMITTED, REPEATABLE READ, SERIALIZABLE, transaction.

physical

A type of operation that involves hardware-related aspects such as disk blocks, memory pages, files, bits, disk reads, and so on. Typically, physical aspects are important during expert-level performance tuning and problem diagnosis. Contrast with **logical**.

See Also logical, physical backup.

physical backup

A **backup** that copies the actual data files. For example, the `mysqlbackup` command of the **MySQL Enterprise Backup** product produces a physical backup, because its output contains data files that can be used directly by the `mysqld` server, resulting in a faster **restore** operation. Contrast with **logical backup**.
See Also backup, logical backup, MySQL Enterprise Backup, restore.

PITR

Acronym for **point-in-time recovery**.
See Also point-in-time recovery.

plan stability

A property of a **query execution plan**, where the optimizer makes the same choices each time for a given **query**, so that performance is consistent and predictable.
See Also query, query execution plan.

plugin

In MySQL 5.1 and earlier, a separately installable form of the **InnoDB** storage engine that includes features and performance enhancements not included in the **built-in** InnoDB for those releases.

For MySQL 5.5 and higher, the MySQL distribution includes the very latest InnoDB features and performance enhancements, known as InnoDB 1.1, and there is no longer a separate InnoDB Plugin.

This distinction is important mainly in MySQL 5.1, where a feature or bug fix might apply to the InnoDB Plugin but not the built-in InnoDB, or vice versa.
See Also built-in, InnoDB.

point-in-time recovery

The process of restoring a **backup** to recreate the state of the database at a specific date and time. Commonly abbreviated **PITR**. Because it is unlikely that the specified time corresponds exactly to the time of a backup, this technique usually requires a combination of a **physical backup** and a **logical backup**. For example, with the **MySQL Enterprise Backup** product, you restore the last backup that you took before the specified point in time, then replay changes from the **binary log** between the time of the backup and the PITR time.
See Also backup, logical backup, MySQL Enterprise Backup, physical backup, PITR.

prefix

See index prefix.

prepared backup

A set of backup files, produced by the **MySQL Enterprise Backup** product, after all the stages of applying **binary logs** and **incremental backups** are finished. The resulting files are ready to be **restored**. Prior to the apply steps, the files are known as a **raw backup**.
See Also binary log, hot backup, incremental backup, MySQL Enterprise Backup, raw backup, restore.

primary key

A set of columns -- and by implication, the index based on this set of columns -- that can uniquely identify every row in a table. As such, it must be a unique index that does not contain any `NULL` values.

InnoDB requires that every table has such an index (also called the **clustered index** or **cluster index**), and organizes the table storage based on the column values of the primary key.

When choosing primary key values, consider using arbitrary values (a **synthetic key**) rather than relying on values derived from some other source (a **natural key**).
See Also clustered index, index, natural key, synthetic key.

process

An instance of an executing program. The operating system switches between multiple running processes, allowing for a certain degree of **concurrency**. On most operating systems, processes can contain multiple **threads** of execution that share resources. Context-switching between threads is faster than the equivalent switching between processes.
See Also concurrency, thread.

pseudo-record

An artificial record in an index, used for **locking** key values or ranges that do not currently exist.
See Also infimum record, locking, supremum record.

Pthreads

The POSIX threads standard, which defines an API for threading and locking operations on UNIX and Linux systems. On UNIX and Linux systems, InnoDB uses this implementation for **mutexes**.
See Also mutex.

purge

A type of garbage collection performed by a separate thread, running on a periodic schedule. The purge includes these actions: removing obsolete values from indexes; physically removing rows that were marked for deletion by previous `DELETE` statements.
See Also crash recovery, delete, doublewrite buffer.

purge buffering

The technique of storing index changes due to `DELETE` operations in the **insert buffer** rather than writing them immediately, so that the physical writes can be performed to minimize random I/O. (Because delete operations are a two-step process, this operation buffers the write that normally purges an index record that was previously marked for deletion.) It is one of the types of **change buffering**; the others are **insert buffering**. and **delete buffering**
See Also change buffer, change buffering, delete buffering, insert buffer, insert buffering.

purge lag

Another name for the `InnoDB` **history list**. Related to the `innodb_max_purge_lag` configuration option.
See Also history list, purge.

purge thread

A **thread** within the InnoDB process that is dedicated to performing the periodic **purge** operation. In MySQL 5.6 and higher, multiple purge threads are enabled by the `innodb_purge_threads` configuration option.
See Also purge, thread.

Q

query

In **SQL**, an operation that reads information from one or more **tables**. Depending on the organization of data and the parameters of the query, the lookup might be optimized by consulting an **index**. If multiple tables are involved, the query is known as a **join**.

For historical reasons, sometimes discussions of internal processing for statements use "query" in a broader sense, including other types of MySQL statements such as **DDL** and **DML** statements.
See Also DDL, DML, index, join, SQL, table.

query execution plan

The set of decisions made by the optimizer about how to perform a **query** most efficiently, including which **index** or indexes to use, and the order in which to **join** tables. **Plan stability** involves the same choices being made consistently for a given query.
See Also index, join, plan stability, query.

query log
See general query log.

quiesce
To reduce the amount of database activity, often in preparation for an operation such as an ALTER TABLE, a **backup**, or a **shutdown**. Might or might not involve doing as much **flushing** as possible, so that **InnoDB** does not continue doing background I/O.

In MySQL 5.6 and higher, the syntax FLUSH TABLES ... FOR EXPORT writes some data to disk for InnoDB tables that make it simpler to back up those tables by copying the data files.
See Also backup, flush, InnoDB, shutdown.

R

RAID
Acronym for "Redundant Array of Inexpensive Drives". Spreading I/O operations across multiple drives enables greater **concurrency** at the hardware level, and improves the efficiency of low-level write operations that otherwise would be performed in sequence.
See Also concurrency.

random dive
A technique for quickly estimating the number of different values in a column (the column's cardinality). InnoDB samples pages at random from the index and uses that data to estimate the number of different values. This operation occurs when each table is first opened.

Originally, the number of sampled pages was fixed at 8; now, it is determined by the setting of the innodb_stats_sample_pages parameter.

The way the random pages are picked depends on the setting of the innodb_use_legacy_cardinality_algorithm parameter. The default setting (OFF) has better randomness than in older releases.
See Also cardinality.

raw backup
The initial set of backup files produced by the **MySQL Enterprise Backup** product, before the changes reflected in the **binary log** and any **incremental backups** are applied. At this stage, the files are not ready to **restore**. After these changes are applied, the files are known as a **prepared backup**.
See Also binary log, hot backup, ibbackup_logfile, incremental backup, MySQL Enterprise Backup, prepared backup, restore.

READ COMMITTED
An **isolation level** that uses a **locking** strategy that relaxes some of the protection between **transactions**, in the interest of performance. Transactions cannot see uncommitted data from other transactions, but they can see data that is committed by another transaction after the current transaction started. Thus, a transaction never sees any bad data, but the data that it does see may depend to some extent on the timing of other transactions.

When a transaction with this isolation level performs UPDATE ... WHERE or DELETE ... WHERE operations, other transactions might have to wait. The transaction can perform SELECT ... FOR UPDATE, and LOCK IN SHARE MODE operations without making other transactions wait.
See Also ACID, isolation level, locking, REPEATABLE READ, SERIALIZABLE, transaction.

READ UNCOMMITTED
The **isolation level** that provides the least amount of protection between transactions. Queries employ a **locking** strategy that allows them to proceed in situations where they would normally wait for another transaction. However, this extra performance comes at the cost of less reliable results, including data that has been changed by other transactions and not committed yet (known as **dirty read**). Use this isolation level only with great caution,

and be aware that the results might not be consistent or reproducible, depending on what other transactions are doing at the same time. Typically, transactions with this isolation level do only queries, not insert, update, or delete operations.
See Also ACID, dirty read, isolation level, locking, transaction.

read view
An internal snapshot used by the **MVCC** mechanism of InnoDB. Certain **transactions**, depending on their **isolation level**, see the data values as they were at the time the transaction (or in some cases, the statement) started. Isolation levels that use a read view are **REPEATABLE READ**, **READ COMMITTED**, and **READ UNCOMMITTED**.
See Also isolation level, MVCC, READ COMMITTED, READ UNCOMMITTED, REPEATABLE READ, transaction.

read-ahead
A type of I/O request that prefetches a group of **pages** (an entire **extent**) into the **buffer pool** asynchronously, in anticipation that these pages will be needed soon. The linear read-ahead technique prefetches all the pages of one extent based on access patterns for pages in the preceding extent, and is part of all MySQL versions starting with the InnoDB Plugin for MySQL 5.1. The random read-ahead technique prefetches all the pages for an extent once a certain number of pages from the same extent are in the buffer pool. Random read-ahead is not part of MySQL 5.5, but is re-introduced in MySQL 5.6 under the control of the `innodb_random_read_ahead` configuration option.
See Also buffer pool, extent, page.

read-only transaction
A type of transaction that can be optimized for `InnoDB` tables by eliminating some of the bookkeeping involved with creating a **read view** for each transaction. Can only perform **non-locking read** queries. It can be started explicitly with the syntax `START TRANSACTION READ ONLY`, or automatically under certain conditions. See Optimizations for Read-Only Transactions for details.
See Also non-locking read, read view, transaction.

record lock
A lock on an index record. For example, `SELECT c1 FOR UPDATE FROM t WHERE c1 = 10;` prevents any other transaction from inserting, updating, or deleting rows where the value of `t.c1` is 10. Contrast with **gap lock** and **next-key lock**.
See Also gap lock, lock, next-key lock.

redo
The data, in units of records, recorded in the **redo log** when DML statements make changes to InnoDB tables. It is used during **crash recovery** to correct data written by incomplete **transactions**. The ever-increasing **LSN** value represents the cumulative amount of redo data that has passed through the redo log.
See Also crash recovery, DML, LSN, redo log, transaction.

redo log
A disk-based data structure used during **crash recovery**, to correct data written by incomplete **transactions**. During normal operation, it encodes requests to change InnoDB table data, which result from SQL statements or low-level API calls through NoSQL interfaces. Modifications that did not finish updating the **data files** before an unexpected **shutdown** are replayed automatically.

The redo log is physically represented as a set of files, typically named `ib_logfile0` and `ib_logfile1`. The data in the redo log is encoded in terms of records affected; this data is collectively referred to as **redo**. The passage of data through the redo logs is represented by the ever-increasing **LSN** value. The original 4GB limit on maximum size for the redo log is raised to 512GB in MySQL 5.6.3.

The disk layout of the redo log is influenced by the configuration options `innodb_log_file_size`, `innodb_log_group_home_dir`, and (rarely) `innodb_log_files_in_group`. The performance of redo

log operations is also affected by the **log buffer**, which is controlled by the `innodb_log_buffer_size` configuration option.
See Also crash recovery, data files, ib_logfile, log buffer, LSN, redo, shutdown, transaction.

redundant row format

The oldest `InnoDB` row format, available for tables using the **Antelope file format**. Prior to MySQL 5.0.3, it was the only row format available in `InnoDB`. In My SQL 5.0.3 and later, the default is **compact row format**. You can still specify redundant row format for compatibility with older `InnoDB` tables.

For additional information about `InnoDB` `REDUNDANT` row format, see Section 14.2.9.4, "`COMPACT` and `REDUNDANT` Row Formats".
See Also Antelope, compact row format, file format, row format.

referential integrity

The technique of maintaining data always in a consistent format, part of the **ACID** philosophy. In particular, data in different tables is kept consistent through the use of **foreign key constraints**, which can prevent changes from happening or automatically propagate those changes to all related tables. Related mechanisms include the **unique constraint**, which prevents duplicate values from being inserted by mistake, and the **NOT NULL constraint**, which prevents blank values from being inserted by mistake.
See Also ACID, FOREIGN KEY constraint, NOT NULL constraint, unique constraint.

relational

An important aspect of modern database systems. The database server encodes and enforces relationships such as one-to-one, one-to-many, many-to-one, and uniqueness. For example, a person might have zero, one, or many phone numbers in an address database; a single phone number might be associated with several family members. In a financial database, a person might be required to have exactly one taxpayer ID, and any taxpayer ID could only be associated with one person.

The database server can use these relationships to prevent bad data from being inserted, and to find efficient ways to look up information. For example, if a value is declared to be unique, the server can stop searching as soon as the first match is found, and it can reject attempts to insert a second copy of the same value.

At the database level, these relationships are expressed through SQL features such as **columns** within a table, unique and `NOT NULL` **constraints**, **foreign keys**, and different kinds of join operations. Complex relationships typically involve data split between more than one table. Often, the data is **normalized**, so that duplicate values in one-to-many relationships are stored only once.

In a mathematical context, the relations within a database are derived from set theory. For example, the `OR` and `AND` operators of a `WHERE` clause represent the notions of union and intersection.
See Also ACID, constraint, foreign key, normalized.

relevance

In the **full-text search** feature, a number signifying the similarity between the search string and the data in the **FULLTEXT index**. For example, when you search for a single word, that word is typically more relevant for a row where if it occurs several times in the text than a row where it appears only once.
See Also full-text search, FULLTEXT index.

REPEATABLE READ

The default **isolation level** for InnoDB. It prevents any rows that are queried from being changed by other transactions, thus blocking **non-repeatable reads** but not **phantom** reads. It uses a moderately strict **locking** strategy so that all queries within a transaction see data from the same snapshot, that is, the data as it was at the time the transaction started.

When a transaction with this isolation level performs `UPDATE ... WHERE`, `DELETE ... WHERE`, `SELECT ... FOR UPDATE`, and `LOCK IN SHARE MODE` operations, other transactions might have to wait.

See Also ACID, consistent read, isolation level, locking, phantom, SERIALIZABLE, transaction.

replication

The practice of sending changes from a **master database**, to one or more **slave databases**, so that all databases have the same data. This technique has a wide range of uses, such as load-balancing for better scalability, disaster recovery, and testing software upgrades and configuration changes. The changes can be sent between the database by methods called **row-based replication** and **statement-based replication**.
See Also row-based replication, statement-based replication.

restore

The process of putting a set of backup files from the **MySQL Enterprise Backup** product in place for use by MySQL. This operation can be performed to fix a corrupted database, to return to some earlier point in time, or (in a **replication** context) to set up a new **slave database**. In the **MySQL Enterprise Backup** product, this operation is performed by the `copy-back` option of the `mysqlbackup` command.
See Also hot backup, MySQL Enterprise Backup, mysqlbackup command, prepared backup, replication.

rollback

A **SQL** statement that ends a **transaction**, undoing any changes made by the transaction. It is the opposite of **commit**, which makes permanent any changes made in the transaction.

By default, MySQL uses the **autocommit** setting, which automatically issues a commit following each SQL statement. You must change this setting before you can use the rollback technique.
See Also ACID, commit, transaction.

rollback segment

The storage area containing the **undo log**, part of the **system tablespace**.
See Also system tablespace, undo log.

row

The logical data structure defined by a set of **columns**. A set of rows makes up a **table**. Within InnoDB **data files**, each **page** can contain one or more rows.

Although InnoDB uses the term **row format** for consistency with MySQL syntax, the row format is a property of each table and applies to all rows in that table.
See Also column, data files, page, row format, table.

row format

The disk storage format for a **row** from an InnoDB **table**. As InnoDB gains new capabilities such as compression, new row formats are introduced to support the resulting improvements in storage efficiency and performance.

Each table has its own row format, specified through the `ROW_FORMAT` option. To see the row format for each InnoDB table, issue the command `SHOW TABLE STATUS`. Because all the tables in the system tablespace share the same row format, to take advantage of other row formats typically requires setting the `innodb_file_per_table` option, so that each table is stored in a separate tablespace.
See Also compact row format, compressed row format, dynamic row format, fixed row format, redundant row format, row, table.

row lock

A **lock** that prevents a row from being accessed in an incompatible way by another **transaction**. Other rows in the same table can be freely written to by other transactions. This is the type of **locking** done by **DML** operations on **InnoDB** tables.

Contrast with **table locks** used by MyISAM, or during **DDL** operations on InnoDB tables that cannot be done with **online DDL**; those locks block concurrent access to the table.
See Also DDL, DML, InnoDB, lock, locking, online DDL, table lock, transaction.

row-based replication

A form of **replication** where events are propagated from the **master** server specifying how to change individual rows on the **slave** server. It is safe to use for all settings of the `innodb_autoinc_lock_mode` option.
See Also auto-increment locking, innodb_autoinc_lock_mode, master server, replication, slave server, statement-based replication.

row-level locking

The **locking** mechanism used for **InnoDB** tables, relying on **row locks** rather than **table locks**. Multiple **transactions** can modify the same table concurrently. Only if two transactions try to modify the same row does one of the transactions wait for the other to complete (and release its row locks).
See Also InnoDB, locking, row lock, table lock, transaction.

rw-lock

The low-level object that InnoDB uses to represent and enforce shared-access **locks** to internal in-memory data structures. Once the lock is acquired, any other process, thread, and so on can read the data structure, but no one else can write to it. Contrast with **mutexes**, which enforce exclusive access. Mutexes and rw-locks are known collectively as **latches**.
See Also latch, lock, mutex, Performance Schema.

S

savepoint

Savepoints help to implement nested **transactions**. They can be used to provide scope to operations on tables that are part of a larger transaction. For example, scheduling a trip in a reservation system might involve booking several different flights; if a desired flight is unavailable, you might **roll back** the changes involved in booking that one leg, without rolling back the earlier flights that were successfully booked.
See Also rollback, transaction.

scalability

The ability to add more work and issue more simultaneous requests to a system, without a sudden drop in performance due to exceeding the limits of system capacity. Software architecture, hardware configuration, application coding, and type of workload all play a part in scalability. When the system reaches its maximum capacity, popular techniques for increasing scalability are **scale up** (increasing the capacity of existing hardware or software) and **scale out** (adding new servers and more instances of MySQL). Often paired with **availability** as critical aspects of a large-scale deployment.
See Also availability, scale out, scale up.

scale out

A technique for increasing **scalability** by adding new servers and more instances of MySQL. For example, setting up replication, MySQL Cluster, connection pooling, or other features that spread work across a group of servers. Contrast with **scale up**.
See Also scalability, scale up.

scale up

A technique for increasing **scalability** by increasing the capacity of existing hardware or software. For example, increasing the memory on a server and adjusting memory-related parameters such as `innodb_buffer_pool_size` and `innodb_buffer_pool_instances`. Contrast with **scale out**.
See Also scalability, scale out.

schema

Conceptually, a schema is a set of interrelated database objects, such as tables, table columns, data types of the columns, indexes, foreign keys, and so on. These objects are connected through SQL syntax, because the columns make up the tables, the foreign keys refer to tables and columns, and so on. Ideally, they are also connected logically, working together as part of a unified application or flexible framework. For example, the

information_schema and **performance_schema** databases use "schema" in their names to emphasize the close relationships between the tables and columns they contain.

In MySQL, physically, a **schema** is synonymous with a **database**. You can substitute the keyword `SCHEMA` instead of `DATABASE` in MySQL SQL syntax, for example using `CREATE SCHEMA` instead of `CREATE DATABASE`.

Some other database products draw a distinction. For example, in the Oracle Database product, a **schema** represents only a part of a database: the tables and other objects owned by a single user.
See Also database, ib-file set, INFORMATION_SCHEMA, Performance Schema.

search index
In MySQL, **full-text search** queries use a special kind of index, the **FULLTEXT** index. In MySQL 5.6.4 and up, `InnoDB` and `MyISAM` tables both support `FULLTEXT` indexes; formerly, these indexes were only available for `MyISAM` tables.
See Also full-text search, FULLTEXT index.

secondary index
A type of InnoDB **index** that represents a subset of table columns. An InnoDB table can have zero, one, or many secondary indexes. (Contrast with the **clustered index**, which is required for each InnoDB table, and stores the data for all the table columns.)

A secondary index can be used to satisfy queries that only require values from the indexed columns. For more complex queries, it can be used to identify the relevant rows in the table, which are then retrieved through lookups using the clustered index.

Creating and dropping secondary indexes has traditionally involved significant overhead from copying all the data in the InnoDB table. The **fast index creation** feature of the InnoDB Plugin makes both `CREATE INDEX` and `DROP INDEX` statements much faster for InnoDB secondary indexes.
See Also clustered index, Fast Index Creation, index.

segment
A division within an InnoDB **tablespace**. If a tablespace is analogous to a directory, the segments are analogous to files within that directory. A segment can grow. New segments can be created.

For example, within a **file-per-table** tablespace, the table data is in one segment and each associated index is in its own segment. The **system tablespace** contains many different segments, because it can hold many tables and their associated indexes. The system tablespace also includes up to 128 **rollback segments** making up the **undo log**.

Segments grow and shrink as data is inserted and deleted. When a segment needs more room, it is extended by one **extent** (1 megabyte) at a time. Similarly, a segment releases one extent's worth of space when all the data in that extent is no longer needed.
See Also extent, file-per-table, rollback segment, system tablespace, tablespace, undo log.

selectivity
A property of data distribution, the number of distinct values in a column (its **cardinality**) divided by the number of records in the table. High selectivity means that the column values are relatively unique, and can retrieved efficiently through an index. If you (or the query optimizer) can predict that a test in a `WHERE` clause only matches a small number (or proportion) of rows in a table, the overall **query** tends to be efficient if it evaluates that test first, using an index.
See Also cardinality, query.

semi-consistent read
A type of read operation used for `UPDATE` statements, that is a combination of **read committed** and **consistent read**. When an `UPDATE` statement examines a row that is already locked, InnoDB returns the latest committed version to MySQL so that MySQL can determine whether the row matches the `WHERE` condition of the `UPDATE`. If

the row matches (must be updated), MySQL reads the row again, and this time InnoDB either locks it or waits for a lock on it. This type of read operation can only happen when the transaction has the read committed **isolation level**, or when the `innodb_locks_unsafe_for_binlog` option is enabled.
See Also consistent read, isolation level, READ COMMITTED.

SERIALIZABLE
The **isolation level** that uses the most conservative locking strategy, to prevent any other transactions from inserting or changing data that was read by this transaction, until it is finished. This way, the same query can be run over and over within a transaction, and be certain to retrieve the same set of results each time. Any attempt to change data that was committed by another transaction since the start of the current transaction, cause the current transaction to wait.

This is the default isolation level specified by the SQL standard. In practice, this degree of strictness is rarely needed, so the default isolation level for InnoDB is the next most strict, **repeatable read**.
See Also ACID, consistent read, isolation level, locking, REPEATABLE READ, transaction.

server
A type of program that runs continuously, waiting to receive and act upon requests from another program (the client). Because often an entire computer is dedicated to running one or more server programs (such as a database server, a web server, an application server, or some combination of these), the term **server** can also refer to the computer that runs the server software.
See Also client, mysqld.

shared lock
A kind of **lock** that allows other **transactions** to read the locked object, and to also acquire other shared locks on it, but not to write to it. The opposite of **exclusive lock**.
See Also exclusive lock, lock, transaction.

shared tablespace
Another way of referring to the **system tablespace**.
See Also system tablespace.

sharp checkpoint
The process of **flushing** to disk all **dirty** buffer pool pages whose redo entries are contained in certain portion of the **redo log**. Occurs before InnoDB reuses a portion of a log file; the log files are used in a circular fashion. Typically occurs with write-intensive **workloads**.
See Also dirty page, flush, redo log, workload.

shutdown
The process of stopping the MySQL server. By default, this process does cleanup operations for **InnoDB** tables, so it can **slow** to shut down, but fast to start up later. If you skip the cleanup operations, it is **fast** to shut down but must do the cleanup during the next restart.

The shutdown mode is controlled by the `innodb_fast_shutdown` option.
See Also fast shutdown, InnoDB, slow shutdown, startup.

slave server
Frequently shortened to "slave". A database **server** machine in a **replication** scenario that receives changes from another server (the **master**) and applies those same changes. Thus it maintains the same contents as the master, although it might lag somewhat behind.

In MySQL, slave servers are commonly used in disaster recovery, to take the place of a master servers that fails. They are also commonly used for testing software upgrades and new settings, to ensure that database configuration changes do not cause problems with performance or reliability.

Slave servers typically have high workloads, because they process all the **DML** (write) operations relayed from the master, as well as user queries. To ensure that slave servers can apply changes from the master fast enough,

they frequently have fast I/O devices and sufficient CPU and memory to run multiple database instances on the same slave server. For example, the master server might use hard drive storage while the slave servers use **SSD**s.
See Also DML, replication, server, SSD.

slow query log
A type of **log** used for performance tuning of SQL statements processed by the MySQL server. The log information is stored in a file. You must enable this feature to use it. You control which categories of "slow" SQL statements are logged. For more information, see Section 5.2.5, “The Slow Query Log”.
See Also general query log, log.

slow shutdown
A type of shutdown that does additional `InnoDB` flushing operations before completing. Also known as a **clean shutdown**. Specified by the configuration parameter `innodb_fast_shutdown=0` or the command `SET GLOBAL innodb_fast_shutdown=0;`. Although the shutdown itself can take longer, that time will be saved on the subsequent startup.
See Also clean shutdown, fast shutdown, shutdown.

snapshot
A representation of data at a particular time, which remains the same even as changes are **committed** by other **transactions**. Used by certain **isolation levels** to allow **consistent reads**.
See Also commit, consistent read, isolation level, transaction.

space ID
An identifier used to uniquely identify an `InnoDB` **tablespace** within a MySQL instance. The space ID for the **system tablespace** is always zero; this same ID applies to all tables within the system tablespace. Each tablespace file created in **file-per-table** mode also has its own space ID.

Prior to MySQL 5.6, this hardcoded value presented difficulties in moving `InnoDB` tablespace files between MySQL instances. Starting in MySQL 5.6, you can copy tablespace files between instances by using the **transportable tablespace** feature involving the statements `FLUSH TABLES ... FOR EXPORT`, `ALTER TABLE ... DISCARD TABLESPACE`, and `ALTER TABLE ... IMPORT TABLESPACE`. The information needed to adjust the space ID is conveyed in the **.cfg file** which you copy along with the tablespace. See Section 14.2.5.5, “Copying Tablespaces to Another Server (Transportable Tablespaces)” for details.
See Also .cfg file, file-per-table, .ibd file, system tablespace, tablespace, transportable tablespace.

spin
A type of **wait** operation that continuously tests whether a resource becomes available. This technique is used for resources that are typically held only for brief periods, where it is more efficient to wait in a "busy loop" than to put the thread to sleep and perform a context switch. If the resource does not become available within a short time, the spin loop ceases and another wait technique is used.
See Also latch, lock, mutex, wait.

SQL
The Structured Query Language that is standard for performing database operations. Often divided into the categories **DDL**, **DML**, and **queries**. MySQL includes some additional statement categories such as **replication**. See Chapter 9, *Language Structure* for the building blocks of SQL syntax, Chapter 11, *Data Types* for the data types to use for MySQL table columns, Chapter 13, *SQL Statement Syntax* for details about SQL statements and their associated categories, and Chapter 12, *Functions and Operators* for standard and MySQL-specific functions to use in queries.
See Also DDL, DML, query, replication.

SSD
Acronym for "solid-state drive". A type of storage device with different performance characteristics than a traditional hard disk drive (**HDD**): smaller storage capacity, faster for random reads, no moving parts, and with

a number of considerations affecting write performance. Its performance characteristics can influence the throughput of a **disk-bound** workload.
See Also disk-bound, SSD.

startup
The process of starting the MySQL server. Typically done by one of the programs listed in Section 4.3, “MySQL Server and Server-Startup Programs”. The opposite of **shutdown**.
See Also shutdown.

statement-based replication
A form of **replication** where SQL statements are sent from the **master** server and replayed on the **slave** server. It requires some care with the setting for the `innodb_autoinc_lock_mode` option, to avoid potential timing problems with **auto-increment locking**.
See Also auto-increment locking, innodb_autoinc_lock_mode, master server, replication, row-based replication, slave server.

statistics
Estimated values relating to each `InnoDB` **table** and **index**, used to construct an efficient **query execution plan**. The main values are the **cardinality** (number of distinct values) and the total number of table rows or index entries. The statistics for the table represent the data in its **primary key** index. The statistics for a **secondary index** represent the rows covered by that index.

The values are estimated rather than counted precisely because at any moment, different **transactions** can be inserting and deleting rows from the same table. To keep the values from being recalculated frequently, you can enable **persistent statistics**, where the values are stored in `InnoDB` system tables, and refreshed only when you issue an `ANALYZE TABLE` statement.

You can control how **NULL** values are treated when calculating statistics through the `innodb_stats_method` configuration option.

Other types of statistics are available for database objects and database activity through the **INFORMATION_SCHEMA** and **PERFORMANCE_SCHEMA** tables.
See Also cardinality, index, INFORMATION_SCHEMA, NULL, Performance Schema, persistent statistics, primary key, query execution plan, secondary index, table, transaction.

stemming
The ability to search for different variations of a word based on a common root word, such as singular and plural, or past, present, and future verb tense. This feature is currently supported in MyISAM **full-text search** feature but not in **FULLTEXT indexes** for InnoDB tables.
See Also full-text search, FULLTEXT index.

stopword
In a **FULLTEXT index**, a word that is considered common or trivial enough that it is omitted from the **search index** and ignored in search queries. Different configuration settings control stopword processing for `InnoDB` and `MyISAM` tables. See Section 12.9.4, “Full-Text Stopwords” for details.
See Also FULLTEXT index, search index.

storage engine
A component of the MySQL database that performs the low-level work of storing, updating, and querying data. In MySQL 5.5 and higher, **InnoDB** is the default storage engine for new tables, superceding MyISAM. Different storage engines are designed with different tradeoffs between factors such as memory usage versus disk usage, read speed versus write speed, and speed versus robustness. Each storage engine manages specific tables, so we refer to `InnoDB` tables, `MyISAM` tables, and so on.

The **MySQL Enterprise Backup** product is optimized for backing up InnoDB tables. It can also back up tables handled by MyISAM and other storage engines.

See Also InnoDB, MySQL Enterprise Backup, table type.

strict mode

The general name for the setting controlled by the `innodb_strict_mode` option. Turning on this setting causes certain conditions that are normally treated as warnings, to be considered errors. For example, certain invalid combinations of options related to **file format** and **row format**, that normally produce a warning and continue with default values, now cause the `CREATE TABLE` operation to fail.

MySQL also has something called strict mode.
See Also file format, innodb_strict_mode, row format.

sublist

Within the list structure that represents the buffer pool, pages that are relatively old and relatively new are represented by different portions of the list. A set of parameters control the size of these portions and the dividing point between the new and old pages.
See Also buffer pool, eviction, list, LRU.

supremum record

A **pseudo-record** in an index, representing the **gap** above the largest value in that index. If a transaction has a statement such as `SELECT ... FOR UPDATE ... WHERE col > 10;`, and the largest value in the column is 20, it is a lock on the supremum record that prevents other transactions from inserting even larger values such as 50, 100, and so on.
See Also gap, infimum record, pseudo-record.

surrogate key

Synonym name for **synthetic key**.
See Also synthetic key.

synthetic key

A indexed column, typically a **primary key**, where the values are assigned arbitrarily. Often done using an **auto-increment** column. By treating the value as completely arbitrary, you can avoid overly restrictive rules and faulty application assumptions. For example, a numeric sequence representing employee numbers might have a gap if an employee was approved for hiring but never actually joined. Or employee number 100 might have a later hiring date than employee number 500, if they left the company and later rejoined. Numeric values also produce shorter values of predictable length. For example, storing numeric codes meaning "Road", "Boulevard", "Expressway", and so on is more space-efficient than repeating those strings over and over.

Also known as a **surrogate key**. Contrast with **natural key**.
See Also auto-increment, natural key, primary key, surrogate key.

system tablespace

A small set of data files (the **ibdata** files) containing the metadata for InnoDB-related objects (the **data dictionary**), and the storage areas for the **undo log**, the **change buffer**, and the **doublewrite buffer**. Depending on the setting of the `innodb_file_per_table`, when tables are created, it might also contain table and index data for some or all InnoDB tables. The data and metadata in the system tablespace apply to all the **databases** in a MySQL **instance**.

Prior to MySQL 5.6.7, the default was to keep all InnoDB tables and indexes inside the system tablespace, often causing this file to become very large. Because the system tablespace never shrinks, storage problems could arise if large amounts of temporary data were loaded and then deleted. In MySQL 5.6.7 and higher, the default is **file-per-table** mode, where each table and its associated indexes are stored in a separate **.ibd file**. This new default makes it easier to use InnoDB features that rely on the **Barracuda** file format, such as table **compression** and the **DYNAMIC** row format.

In MySQL 5.6 and higher, setting a value for the `innodb_undo_tablespaces` option splits the **undo log** into one or more separate tablespace files. These files are still considered part of the system tablespace.

Keeping all table data in the system tablespace or in separate `.ibd` files has implications for storage management in general. The **MySQL Enterprise Backup** product might back up a small set of large files, or many smaller files. On systems with thousands of tables, the filesystem operations to process thousands of `.ibd` files can cause bottlenecks.
See Also Barracuda, change buffer, compression, data dictionary, database, doublewrite buffer, dynamic row format, file-per-table, .ibd file, ibdata file, innodb_file_per_table, instance, MySQL Enterprise Backup, tablespace, undo log.

T

.TRG file
A file containing **trigger** parameters. Files with this extension are always included in backups produced by the `mysqlbackup` command of the **MySQL Enterprise Backup** product.
See Also MySQL Enterprise Backup, mysqlbackup command, .TRN file.

.TRN file
A file containing trigger namespace information. Files with this extension are always included in backups produced by the `mysqlbackup` command of the **MySQL Enterprise Backup** product.
See Also MySQL Enterprise Backup, mysqlbackup command, .TRG file.

table
Each MySQL table is associated with a particular **storage engine**. **InnoDB** tables have particular **physical** and **logical** characteristics that affect performance, **scalability**, **backup**, administration, and application development.

In terms of file storage, each InnoDB table is either part of the single big InnoDB **system tablespace**, or in a separate **`.ibd`** file if the table is created in **file-per-table** mode. The **`.ibd`** file holds data for the table and all its **indexes**, and is known as a **tablespace**.

InnoDB tables created in file-per-table mode can use the **Barracuda** file format. Barracuda tables can use the **DYNAMIC row format** or the **COMPRESSED row format**. These relatively new settings enable a number of InnoDB features, such as **compression**, **fast index creation**, and **off-page columns**

For backward compatibility with MySQL 5.1 and earlier, InnoDB tables inside the system tablespace must use the **Antelope** file format, which supports the **compact row format** and the **redundant row format**.

The **rows** of an InnoDB table are organized into an index structure known as the **clustered index**, with entries sorted based on the **primary key** columns of the table. Data access is optimized for queries that filter and sort on the primary key columns, and each index contains a copy of the associated primary key columns for each entry. Modifying values for any of the primary key columns is an expensive operation. Thus an important aspect of InnoDB table design is choosing a primary key with columns that are used in the most important queries, and keeping the primary key short, with rarely changing values.
See Also Antelope, backup, Barracuda, clustered index, compact row format, compressed row format, compression, dynamic row format, Fast Index Creation, file-per-table, .ibd file, index, off-page column, primary key, redundant row format, row, system tablespace, tablespace.

table lock
A lock that prevents any other **transaction** from accessing a table. InnoDB makes considerable effort to make such locks unnecessary, by using techniques such as **online DDL**, **row locks** and **consistent reads** for processing **DML** statements and **queries**. You can create such a lock through SQL using the `LOCK TABLE` statement; one of the steps in migrating from other database systems or MySQL storage engines is to remove such statements wherever practical.
See Also consistent read, DML, lock, locking, online DDL, query, row lock, table, transaction.

table scan
See full table scan.

table statistics
: See statistics.

table type
: Obsolete synonym for **storage engine**. We refer to `InnoDB` tables, `MyISAM` tables, and so on.
See Also InnoDB, storage engine.

tablespace
: A data file that can hold data for one or more InnoDB **tables** and associated **indexes**. The **system tablespace** contains the tables that make up the **data dictionary**, and prior to MySQL 5.6 holds all the other InnoDB tables by default. Turning on the `innodb_file_per_table` option, the default in MySQL 5.6 and higher, allows newly created tables to each have their own tablespace, with a separate **data file** for each table.

  Using multiple tablespaces, by turning on the `innodb_file_per_table` option, is vital to using many MySQL features such as table compression and transportable tablespaces, and managing disk usage. See Section 14.2.5.2, "InnoDB File-Per-Table Mode" for details.

  Tablespaces created by the built-in InnoDB storage engine are upward compatible with the InnoDB Plugin. Tablespaces created by the InnoDB Plugin are downward compatible with the built-in InnoDB storage engine, if they use the **Antelope** file format.

  MySQL Cluster also groups its tables into tablespaces. See MySQL Cluster Disk Data Objects for details.
See Also Antelope, Barracuda, compressed row format, data dictionary, data files, file-per-table, index, innodb_file_per_table, system tablespace, table.

tablespace dictionary
: A representation of the **data dictionary** metadata for a table, within the InnoDB **tablespace**. This metadata can be checked against the **.frm file** for consistency when the table is opened, to diagnose errors resulting from out-of-date `.frm` files. This information is present for InnoDB tables that are part of the **system tablespace**, as well as for tables that have their own **.ibd file** because of the **file-per-table** option.
See Also data dictionary, file-per-table, .frm file, .ibd file, system tablespace, tablespace.

temporary table
: A table whose data does not need to be truly permanent. For example, temporary tables might be used as storage areas for intermediate results in complicated calculations or transformations; this intermediate data would not need to be recovered after a crash. Database products can take various shortcuts to improve the performance of operations on temporary tables, by being less scrupulous about writing data to disk and other measures to protect the data across restarts.

  Sometimes, the data itself is removed automatically at a set time, such as when the transaction ends or when the session ends. With some database products, the table itself is removed automatically too.
See Also table.

temporary tablespace
: The tablespace for non-compressed `InnoDB` temporary tables and related objects. This tablespace was introduced in MySQL 5.7.1. The configuration file option, `innodb_temp_data_file_path`, allows users to define a relative path for the temporary data file. If `innodb_temp_data_file_path` is not specified, the default behavior is to create a single auto-extending 12MB data file named `ibtmp1` in the data directory, alongside `ibdata1`. The temporary tablespace is recreated on each server start and receives a dynamically generated space-id, which helps avoid conflicts with existing space-ids. The temporary tablespace cannot reside on a raw device. Inability or error creating the temporary table is treated as fatal and server startup will be refused.

  The tablespace is removed on normal shutdown or on init abort, which may occur when a user specifies the wrong startup options, for example. The temporary tablespace is not removed when a crash occurs. In this case, the database administrator can remove the tablespace manually or restart the server with the same configuration, which will remove and recreate the temporary tablespace.
See Also ibtmp file.

text collection

The set of columns included in a **FULLTEXT index**.

See Also FULLTEXT index.

thread

A unit of processing that is typically more lightweight than a **process**, allowing for greater **concurrency**.

See Also concurrency, master thread, process, Pthreads.

torn page

An error condition that can occur due to a combination of I/O device configuration and hardware failure. If data is written out in chunks smaller than the InnoDB **page size** (by default, 16KB), a hardware failure while writing could result in only part of a page being stored to disk. The InnoDB **doublewrite buffer** guards against this possibility.

See Also doublewrite buffer.

TPS

Acronym for "**transactions** per second", a unit of measurement sometimes used in benchmarks. Its value depends on the **workload** represented by a particular benchmark test, combined with factors that you control such as the hardware capacity and database configuration.

See Also transaction, workload.

transaction

Transactions are atomic units of work that can be committed or rolled back. When a transaction makes multiple changes to the database, either all the changes succeed when the transaction is committed, or all the changes are undone when the transaction is rolled back.

Database transactions, as implemented by InnoDB, have properties that are collectively known by the acronym **ACID**, for atomicity, consistency, isolation, and durability.

See Also ACID, commit, isolation level, lock, rollback.

transaction ID

An internal field associated with each row. This field is physically changed by INSERT, UPDATE, and DELETE operations to record which transaction has locked the row.

See Also implicit row lock.

transportable tablespace

A feature that allows a **tablespace** to be moved from one instance to another. Traditionally, this has not been possible for InnoDB tablespaces because all table data was part of the **system tablespace**. In MySQL 5.6 and higher, the `FLUSH TABLES ... FOR EXPORT` syntax prepares an InnoDB table for copying to another server; running `ALTER TABLE ... DISCARD TABLESPACE` and `ALTER TABLE ... IMPORT TABLESPACE` on the other server brings the copied data file into the other instance. A separate `.cfg` file, copied along with the **.ibd file**, is used to update the table metadata (for example the **space ID**) as the tablespace is imported. See Section 14.2.5.5, “Copying Tablespaces to Another Server (Transportable Tablespaces)” for usage information.

See Also .ibd file, space ID, system tablespace, tablespace.

troubleshooting

Resources for troubleshooting InnoDB reliability and performance issues include: the Information Schema tables.

truncate

A **DDL** operation that removes the entire contents of a table, while leaving the table and related indexes intact. Contrast with **drop**. Although conceptually it has the same result as a `DELETE` statement with no `WHERE` clause, it operates differently behind the scenes: InnoDB creates a new empty table, drops the old table, then renames the new table to take the place of the old one. Because this is a DDL operation, it cannot be **rolled back**.

If the table being truncated contains foreign keys that reference another table, the truncation operation uses a slower method of operation, deleting one row at a time so that corresponding rows in the referenced table can be deleted as needed by any `ON DELETE CASCADE` clause. (MySQL 5.5 and higher do not allow this slower form of truncate, and return an error instead if foreign keys are involved. In this case, use a `DELETE` statement instead.

See Also DDL, drop, foreign key, rollback.

tuple

A technical term designating an ordered set of elements. It is an abstract notion, used in formal discussions of database theory. In the database field, tuples are usually represented by the columns of a table row. They could also be represented by the result sets of queries, for example, queries that retrieved only some columns of a table, or columns from joined tables.
See Also cursor.

two-phase commit

An operation that is part of a distributed **transaction**, under the **XA** specification. (Sometimes abbreviated as 2PC.) When multiple databases participate in the transaction, either all databases **commit** the changes, or all databases **roll back** the changes.
See Also commit, rollback, transaction, XA.

U

undo

Data that is maintained throughout the life of a **transaction**, recording all changes so that they can be undone in case of a **rollback** operation. It is stored in the **undo log**, also known as the **rollback segment**, either within the **system tablespace** or in separate **undo tablespaces**.
See Also rollback, rollback segment, system tablespace, transaction, undo log, undo tablespace.

undo buffer

See undo log.

undo log

A storage area that holds copies of data modified by active **transactions**. If another transaction needs to see the original data (as part of a **consistent read** operation), the unmodified data is retrieved from this storage area.

By default, this area is physically part of the **system tablespace**. In MySQL 5.6 and higher, you can use the `innodb_undo_tablespaces` and `innodb_undo_directory` configuration options to split it into one or more separate **tablespace** files, the **undo tablespaces**, optionally stored on another storage device such as an **SSD**.

The undo log is split into separate portions, the **insert undo buffer** and the **update undo buffer**. Collectively, these parts are also known as the **rollback segment**, a familiar term for Oracle DBAs.
See Also consistent read, rollback segment, SSD, system tablespace, transaction, undo tablespace.

undo tablespace

One of a set of files containing the **undo log**, when the undo log is separated from the **system tablespace** by the `innodb_undo_tablespaces` and `innodb_undo_directory` configuration options. Only applies to MySQL 5.6 and higher.
See Also system tablespace, undo log.

unique constraint

A kind of **constraint** that asserts that a column cannot contain any duplicate values. In terms of **relational** algebra, it is used to specify 1-to-1 relationships. For efficiency in checking whether a value can be inserted (that is, the value does not already exist in the column), a unique constraint is supported by an underlying **unique index**.
See Also constraint, relational, unique index.

unique index

An index on a column or set of columns that have a **unique constraint**. Because the index is known not to contain any duplicate values, certain kinds of lookups and count operations are more efficient than in the normal kind of index. Most of the lookups against this type of index are simply to determine if a certain value exists or not.

The number of values in the index is the same as the number of rows in the table, or at least the number of rows with non-null values for the associated columns.

The **insert buffering** optimization does not apply to unique indexes. As a workaround, you can temporarily set `unique_checks=0` while doing a bulk data load into an InnoDB table.
See Also cardinality, insert buffering, unique constraint, unique key.

unique key
The set of columns (one or more) comprising a **unique index**. When you can define a `WHERE` condition that matches exactly one row, and the query can use an associated unique index, the lookup and error handling can be performed very efficiently.
See Also cardinality, unique constraint, unique index.

V

victim
The transaction that is automatically chosen to be rolled back when a **deadlock** is detected. InnoDB rolls back the transaction that has updated the fewest rows.
See Also deadlock, deadlock detection, innodb_lock_wait_timeout.

W

wait
When an operation, such as acquiring a **lock**, **mutex**, or **latch**, cannot be completed immediately, InnoDB pauses and tries again. The mechanism for pausing is elaborate enough that this operation has its own name, the **wait**. Individual threads are paused using a combination of internal InnoDB scheduling, operating system `wait()` calls, and short-duration **spin** loops.

On systems with heavy load and many transactions, you might use the output from the `SHOW INNODB STATUS` command to determine whether threads are spending too much time waiting, and if so, how you can improve **concurrency**.
See Also concurrency, latch, lock, mutex, spin.

warm backup
A **backup** taken while the database is running, but that restricts some database operations during the backup process. For example, tables might become read-only. For busy applications and web sites, you might prefer a **hot backup**.
See Also backup, cold backup, hot backup.

warm up
To run a system under a typical **workload** for some time after startup, so that the **buffer pool** and other memory regions are filled as they would be under normal conditions.

This process happens naturally over time when a MySQL server is restarted or subjected to a new workload. Starting in MySQL 5.6, you can speed up the warmup process by setting the configuration variables `innodb_buffer_pool_dump_at_shutdown=ON` and `innodb_buffer_pool_load_at_startup=ON`, to bring the contents of the buffer pool back into memory after a restart. Typically, you run a workload for some time to warm up the buffer pool before running performance tests, to ensure consistent results across multiple runs; otherwise, performance might be artificially low during the first run.
See Also buffer pool, workload.

Windows
The built-in **InnoDB** storage engine and the InnoDB **Plugin** are supported on all the same Microsoft Windows versions as the MySQL server. The **MySQL Enterprise Backup** product has more comprehensive support for Windows systems than the **InnoDB Hot Backup** product that it supersedes.

See Also InnoDB, MySQL Enterprise Backup, plugin.

workload

The combination and volume of **SQL** and other database operations, performed by a database application during typical or peak usage. You can subject the database to a particular workload during performance testing to identify **bottlenecks**, or during capacity planning.

See Also bottleneck, CPU-bound, disk-bound, SQL.

write combining

An optimization technique that reduces write operations when **dirty pages** are **flushed** from the InnoDB **buffer pool**. If a row in a page is updated multiple times, or multiple rows on the same page are updated, all of those changes are stored to the data files in a single write operation rather than one write for each change.

See Also buffer pool, dirty page, flush.

X

XA

A standard interface for coordinating distributed **transactions**, allowing multiple databases to participate in a transaction while maintaining **ACID** compliance. For full details, see Section 13.3.7, “XA Transactions”.

XA Distributed Transaction support is turned on by default. If you are not using this feature, you can disable the `innodb_support_xa` configuration option, avoiding the performance overhead of an extra fsync for each transaction.

See Also commit, transaction, two-phase commit.

Y

young

A characteristic of a **page** in the `InnoDB` **buffer pool** meaning it has been accessed recently, and so is moved within the buffer pool data structure, so that it will not be **flushed** soon by the **LRU** algorithm. This term is used in some **information schema** column names of tables related to the buffer pool.

See Also buffer pool, flush, INFORMATION_SCHEMA, LRU, page.

General Index

C Function Index

Command Index

Section 21.8.16, “Controlling Automatic Reconnection Behavior”

Section 14.2.12.5, “Controlling Optimizer Statistics Estimation”

Section 14.2.6.4, “Converting Tables from MyISAM to InnoDB”

Section 2.10.5, “Copying MySQL Databases to Another Machine”

Section 3.3.1, “Creating and Selecting a Database”

Section 14.2.5.1, “Creating the InnoDB Tablespace”

Section 2.3.5.6, “Customizing the PATH for MySQL Tools”

Section 22.4.2, “Debugging a MySQL Client”

Section 18.1, “Defining Stored Programs”

Section 4.6.6, “mysql_config_editor — MySQL Configuration Utility”

Section 4.6.7, “mysqlbinlog — Utility for Processing Binary Log Files”

Section 4.5.4, “mysqldump — A Database Backup Program”

Section 4.5.6, “mysqlshow — Display Database, Table, and Column Information”

Section 4.4.3, “mysql_install_db — Initialize MySQL Data Directory”

Section 4.4.6, “mysql_tzinfo_to_sql — Load the Time Zone Tables”

Disabling mysql Auto-Reconnect

Section 2.10.2, “Downgrading MySQL”

Enabling InnoDB Monitors

Section 6.1.2.1, “End-User Guidelines for Password Security”

Section 3.2, “Entering Queries”

Section 2.11, “Environment Variables”

Section 18.4.2, “Event Scheduler Configuration”

Section 7.3, “Example Backup and Recovery Strategy”

Section 21.8.3, “Example C API Client Programs”

Section 3.6, “Examples of Common Queries”

Section 14.2.11.5, “Examples of Online DDL”

Section 17.3.3, “Exchanging Partitions and Subpartitions with Tables”

Section 4.5.1.5, “Executing SQL Statements from a Text File”

Chapter 12, *Functions and Operators*

Section 2.4.1, “General Notes on Installing MySQL on Mac OS X”

Section 14.2.2.11, “How to Cope with Deadlocks”

Section C.5.1, “How to Determine What Is Causing a Problem”

Section 1.7, “How to Report Bugs or Problems”

Section 6.1.5, “How to Run MySQL as a Normal User”

Section 12.14, “Information Functions”

Input-Line Editing

Section 2.5.3, “Installing MySQL on Linux Using RPM Packages”

Section 4.2.1, “Invoking MySQL Programs”

Section 7.4.5.1, “Making a Copy of a Database”

Section 6.1.3, “Making MySQL Secure Against Attackers”

Section 8.12.1, “Measuring the Speed of Expressions and Functions”

Section 15.7, “MySQL Proxy”

Section 10.6, “MySQL Server Time Zone Support”

Section 8.2.1.19, “Optimizing LIMIT Queries”

Section 4.2.3.6, “Option Defaults, Options Expecting Values, and the = Sign”

Section 4.1, “Overview of MySQL Programs”

Section C.5.2.10, “Packet Too Large”

Section 6.1.2.4, “Password Hashing in MySQL”

Section 6.3.8, “Pluggable Authentication”

Section 7.5, “Point-in-Time (Incremental) Recovery Using the Binary Log”

Section 2.9.1.1, “Problems Running mysql_install_db”

Section 4.2.3.2, “Program Option Modifiers”

Section 2.10.4, “Rebuilding or Repairing Tables or Indexes”

Section 7.4.4, “Reloading Delimited-Text Format Backups”

Section 7.4.2, “Reloading SQL-Format Backups”

Section 16.4.1.27, “Replication of Server-Side Help Tables”

Resetting the Root Password: Generic Instructions

Section E.9, “Restrictions on Pluggable Authentication”

Section 2.9.2, “Securing the Initial MySQL Accounts”

Section 6.1.6, “Security Issues with LOAD DATA LOCAL”

Section 5.1.3, “Server Command Options”

Section C.3, “Server Error Codes and Messages”

Section 5.1.4, “Server System Variables”

Section 5.1.10, “Server-Side Help”

Section 12.18.5, “Spatial Analysis Functions”

Section 4.2.3, “Specifying Program Options”

Section 13.5, “SQL Syntax for Prepared Statements”

Section 2.9.1.2, “Starting and Stopping MySQL Automatically”

Section 2.3.5.7, “Starting MySQL as a Windows Service”

Section 9.1.1, “String Literals”

Section 11.4.3, “The BLOB and TEXT Types”

Section 6.3.9.5, “The Cleartext Client-Side Authentication Plugin”

Section 6.3.9.4, “The SHA-256 Authentication Plugin”

Section 6.3.9.6, “The Socket Peer-Credential Authentication Plugin”

Section 18.3.1, “Trigger Syntax and Examples”

Section 14.2.17.3, “Troubleshooting InnoDB Data Dictionary Operations”

Chapter 3, *Tutorial*

Section 1.2, “Typographical and Syntax Conventions”

Unicode Support on Windows

Section 2.10.1.2, “Upgrading from MySQL 5.6 to 5.7”

Function Index

[index top [3243]]

INFORMATION_SCHEMA Index

Join Types Index

Operator Index

Symbols | A | B | C | D | E | I | L | N | O | R | X

Option Index

Privileges Index

SQL Modes Index

Statement/Syntax Index

Section 14.8.2, “MERGE Table Problems”
Section 14.3.1, “MyISAM Startup Options”
Section 14.3.3, “MyISAM Table Storage Formats”
Section 21.8.7.36, “mysql_info()”
Section 13.7.2.4, “OPTIMIZE TABLE Syntax”
Section 17.2.3.1, “RANGE COLUMNS partitioning”
Section 17.2.1, “RANGE Partitioning”
Section 13.1.26, “RENAME TABLE Syntax”
Section 13.7.5.39, “SHOW WARNINGS Syntax”
Section C.5.7.2, “TEMPORARY Table Problems”
Section 11.3.4, “YEAR(2) Limitations and Migrating to YEAR(4)”
Section 16.1.4.4, “Binary Log Options and Variables”
Section 10.1.3.4, “Column Character Set and Collation”
Section 10.1.13, “Column Character Set Conversion”
Section 14.2.11.4, “Combining or Separating DDL Statements”
Section 14.2.2.4, “Consistent Nonlocking Reads”
Controlling Transactional Behavior of the InnoDB memcached Plugin
Section 14.2.6.4, “Converting Tables from MyISAM to InnoDB”
Section 14.2.6.1, “Creating InnoDB Tables”
Section 3.3.2, “Creating a Table”
Section 12.18.4.3, “Creating Spatial Columns”
Section 12.18.6.1, “Creating Spatial Indexes”
Section 14.2.10.4, “Defragmenting a Table”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 14.2.7.2, “Enabling Compression for a Table”
Section 14.2.11.5, “Examples of Online DDL”
Section 17.3.3, “Exchanging Partitions and Subpartitions with Tables”
Section 12.9.6, “Fine-Tuning MySQL Full-Text Search”
Section 12.9, “Full-Text Search Functions”
Section 8.12.5.2, “General Thread States”
Section 14.2.7.5, “How Compression Works for InnoDB Tables”
Section 14.2.11.7, “How Crash Recovery Works with Online DDL”
Section C.5.4.3, “How MySQL Handles a Full Disk”
Section 8.9.3.1, “How the Query Cache Operates”
Section 7.6.3, “How to Repair MyISAM Tables”
Section 14.2.11.6, “Implementation Details of Online DDL”
Section 12.14, “Information Functions”
Section 14.2.5.2, “InnoDB File-Per-Table Mode”
Section C.5.8, “Known Issues in MySQL”
Section 14.2.11.9, “Limitations of Online DDL”
Section E.10.3, “Limits on Table Size”
Section 17.3.4, “Maintenance of Partitions”
Section 17.3.2, “Management of HASH and KEY Partitions”
Section 17.3.1, “Management of RANGE and LIST Partitions”
Section 14.2.6.2, “Moving or Copying InnoDB Tables to Another Machine”
Section 1.8.1, “MySQL Extensions to Standard SQL”
Online DDL
Section 14.2.11.8, “Online DDL for Partitioned InnoDB Tables”
Section 11.2.6, “Out-of-Range and Overflow Handling”
Section 14.2.9.1, “Overview of InnoDB Row Storage”
Section 14.2.11.1, “Overview of Online DDL”
Section 14.2.7.1, “Overview of Table Compression”
Section 17.3, “Partition Management”
Section 17.6.4, “Partitioning and Locking”
Section 17.6.1, “Partitioning Keys, Primary Keys, and Unique Keys”
Section 14.2.11.2, “Performance and Concurrency Considerations for Online DDL”
Persistent Optimizer Statistics for InnoDB Tables
Section 6.2.1, “Privileges Provided by MySQL”
Section C.5.7.1, “Problems with ALTER TABLE”
Section 2.10.4, “Rebuilding or Repairing Tables or Indexes”
Section 16.4.1.1, “Replication and AUTO_INCREMENT”
Section 16.4.1.25, “Replication and Reserved Words”
Replication with More Columns on Master or Slave
Section 17.6, “Restrictions and Limitations on Partitioning”
Section E.5, “Restrictions on Views”
Section 5.2.1, “Selecting General Query and Slow Query Log Output Destinations”
Section 5.1.3, “Server Command Options”
Section 5.1.7, “Server SQL Modes”
Section 5.1.4, “Server System Variables”
Section 5.2.4.2, “Setting The Binary Log Format”
Section 14.1, “Setting the Storage Engine”
Section 13.1.14.3, “Silent Column Specification Changes”
Section 14.2.5.4, “Specifying the Location of a Tablespace”
Section 14.2.9.2, “Specifying the Row Format for a Table”
Section 14.2.7.6, “SQL Compression Syntax Warnings and Errors”
Section 14.2.11.3, “SQL Syntax for Online DDL”
Section 14.2.17.2, “Starting InnoDB on a Corrupted Database”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 11.1.3, “String Type Overview”
Section 10.1.3.3, “Table Character Set and Collation”
Section 19.14, “The INFORMATION_SCHEMA PARTITIONS Table”
Section 14.4, “The MEMORY Storage Engine”
Section 14.3, “The MyISAM Storage Engine”

Section 10.1.4, “Connection Character Sets and Collations”
Section 14.2.2.4, “Consistent Nonlocking Reads”
Section 14.2.6.4, “Converting Tables from MyISAM to InnoDB”
Section 14.9.2.1, “Creating a FEDERATED Table Using CONNECTION”
Section 3.3.1, “Creating and Selecting a Database”
Section 13.6.6.2, “Cursor DECLARE Syntax”
Section 13.6.6.3, “Cursor FETCH Syntax”
Section 14.2.2.10, “Deadlock Detection and Rollback”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.5.7, “mysqlslap — Load Emulation Client”
Section 8.4.3.2, “Disadvantages of Creating Many Tables in the Same Database”
Section 8.11.5.2, “DNS Lookup Optimization and the Host Cache”
Section 5.1.5.2, “Dynamic System Variables”
Section 3.2, “Entering Queries”
Section 18.4.2, “Event Scheduler Configuration”
Section 10.1.7.8, “Examples of the Effect of Collation”
Section 1.8.2.4, “Foreign Key Differences”
Chapter 12, *Functions and Operators*
Section 8.12.5.2, “General Thread States”
Section 14.2.6.3, “Grouping DML Operations with Transactions”
Section 8.2.1.2, “How MySQL Optimizes WHERE Clauses”
Section 17.2.7, “How MySQL Partitioning Handles NULL”
Section 8.4.4, “How MySQL Uses Internal Temporary Tables”
Section 8.9.3.1, “How the Query Cache Operates”
Section 14.2.2.11, “How to Cope with Deadlocks”
Section 1.7, “How to Report Bugs or Problems”
Section 9.2.1, “Identifier Qualifiers”
Section 13.2.9.3, “Index Hint Syntax”
Section 12.14, “Information Functions”
Section 8.10.1, “Internal Locking Methods”
Section C.5.8, “Known Issues in MySQL”
Section 13.6.4.2, “Local Variable Scope and Resolution”
Section 14.2.2.5, “Locking Reads (SELECT ... FOR UPDATE and SELECT ... LOCK IN SHARE MODE)”
Section 14.2.2.8, “Locks Set by Different SQL Statements in InnoDB”
Section 5.2.4.4, “Logging Format for Changes to mysql Database Tables”
Section 6.1.3, “Making MySQL Secure Against Attackers”
Section 17.3.1, “Management of RANGE and LIST Partitions”
Section 8.3.5, “Multiple-Column Indexes”
Section 1.8.1, “MySQL Extensions to Standard SQL”
Chapter 20, *MySQL Performance Schema*
Section 12.9.1, “Natural Language Full-Text Searches”
Section 8.8.3, “Obtaining Execution Plan Information for a Named Connection”
Section 17.3.5, “Obtaining Information About Partitions”
Section 8.3, “Optimization and Indexes”
Optimizations for Read-Only Transactions
Section C.5.6, “Optimizer-Related Issues”
Section 8.5.2, “Optimizing InnoDB Transaction Management”
Section 8.6.1, “Optimizing MyISAM Queries”
Section 8.2.1, “Optimizing SELECT Statements”
Section 8.8.1, “Optimizing Queries with EXPLAIN”
Optimizing Subqueries in the FROM Clause (Derived Tables)
Optimizing Subqueries with EXISTS Strategy
Optimizing Subqueries with Semi-Join Transformations
Section 4.6.3.4, “Other myisamchk Options”
Section 8.2.5, “Other Optimization Tips”
Section 14.2.11.1, “Overview of Online DDL”
Section 17.4, “Partition Pruning”
Section 17.5, “Partition Selection”
Section 17.6.4, “Partitioning and Locking”
Section 20.4, “Performance Schema Instrument Naming Conventions”
Section 20.9.10, “Performance Schema Replication Tables”
Section 2.9.1, “Postinstallation Procedures for Unix-like Systems”
Section 6.2.1, “Privileges Provided by MySQL”
Section C.5.5.2, “Problems Using DATE Columns”
Section C.5.5.8, “Problems with Floating-Point Values”
Section 8.9.3.2, “Query Cache SELECT Options”
Section 8.9.3.4, “Query Cache Status and Maintenance”
Section 8.12.5.3, “Query Cache Thread States”
Section 14.5.1, “Repairing and Checking CSV Tables”
Section 16.2, “Replication Implementation”
Section 16.1.4.2, “Replication Master Options and Variables”
Section 16.4.1.4, “Replication of CREATE ... IF NOT EXISTS Statements”
Section 16.4.1.11, “Replication of Invoked Features”
Section 16.1.4.3, “Replication Slave Options and Variables”
Section 9.3, “Reserved Words”
Section E.1, “Restrictions on Stored Programs”
Section E.5, “Restrictions on Views”
Section 3.3.4, “Retrieving Information from a Table”
Section 3.6.7, “Searching on Two Keys”
Section 2.9.2, “Securing the Initial MySQL Accounts”
Section 3.3.4.1, “Selecting All Data”
Section 3.3.4.2, “Selecting Particular Rows”
Section 5.1.7, “Server SQL Modes”
Section 5.1.6, “Server Status Variables”
Section 5.1.4, “Server System Variables”

System Variable Index

[index top [3383]]

Status Variable Index

Transaction Isolation Level Index